

المحتويات

الفصل الخامس : الاستدعاء الذاتي

166		1-5	مقدمة
166		2-5	تعريف الاستدعاء الذاتي
170		3-5	معالجة برامج الاستدعاء الذاتي
174		4-5	مبنى لنجا الاستدعاء الذاتي
177			برامج الفصل
184			امثلة الفصل

الفصل السادس : هيكل الشجرة

187		1-6	المخطط
188		1-1-6	تمثيل المخطط
198		2-6	هيكل الشجرة
203		3-6	الشجرة الثانية
206		4-6	أنواع أخرى من الأشجار
220		5-6	استعراض (مسح) عقد الشجرة
225		6-6	تمثيل الأشجار
225		1-6-6	تمثيل الأشجار الاعتيادية
229		2-6-6	تمثيل الأشجار الثنائية
237		7-6	تمثيل التعابير الحسابية باستخدام الشجرة الثنائية
243		8-6	تحويل الشجرة الاعتيادية الى شجرة ثنائية
246		9-6	شجرة البحث الثنائية
254			برامج الفصل
258			امثلة الفصل

المحتويات

69		3-3	الطابور
70		1-3-3	تمثيل الطابور باستخدام المصفوفة
72		2-3-3	خوارزميات الطابور
73		3-3-3	البرامج الفرعية لتنفيذ عمليات الطابور
75		4-3-3	تمثيل الطابور باستخدام Structure
77		5-3-3	تطبيقات الطابور
80		4-3	الطابور الدائري
85		5-3	الطابور الثاني (المزدوج)
86			برامج الفصل
99			امثلة الفصل

الفصل الرابع : الهياكل الموصولة

105		1-4	التخصيص الخرجي
105		1-1-4	الخزن التماسلي
105		2-1-4	الخزن الديناميكي
105		3-1-4	المقارنة بين الخزن التماسلي والخزن الديناميكي
106			المؤشرات
107		2-4	القائمة الموصولة
111		3-4	المكس الموصول
124		4-4	الطابور الموصول
128		5-4	القائمة الموصولة الدائرية
134		6-4	القائمة الموصولة المزدوجة
137		7-4	برامج الفصل
140			امثلة الفصل
160			

الفصل الأول مبادئ أساسية

Introduction	1-1 المقدمة
Data Structures	2-1 هياكل البيانات
Types of Data Structures	3-1 أنواع هياكل البيانات
Selection of data Structure	4-1 كيفية اختيار الهيكل البياني
Why we study data structures	5-1 لماذا ندرس هياكل البيانات
Object Oriented Approach	6-1 ماذا يعني
Abstract Data Type	7-1 النوع البياني المجرد

اختبر

الفصل السابع : الترتيب والبحث

263	1-7 الترتيب
263	1-1-1 الغرض من الترتيب
263	2-1-1 خطوات عملية الترتيب
263	3-1-1 أنواع خوارزميات الترتيب
263	1- الترتيب الداخلي
264	ب- الترتيب الخارجي
264	4-1-1 العوامل الرئيسية المحددة لاختيار خوارزمية الترتيب
264	5-1-1 ترتيب بالاضافة
266	6-1-1 الترتيب بالاختيار
268	7-1-1 الترتيب القاعه
270	8-1-1 الترتيب السريع
280	9-1-1 الترتيب الكوسى
286	10-1-1 ترتيب الدمج
288	2-7 البحث
288	1-2-7 أنواع خوارزميات البحث
288	2-2-7 البحث التعملي
290	3-2-7 البحث الثنائي
294	برامج الفصل
305	امثلة الفصل

الفصل الثامن : تحليل الخوارزمية واعداد البرنامج

309	1-8 تحليل الخوارزمية
320	2-8 مزايا البرنامج الجيد
322	3-8 تصميم البرنامج
325	4-8 موارد البرنامج
326	امثلة الفصل
327	المصادر

1-1 مقدمة

من العوامل المهمة في معالجة البيانات والحصول على النتائج المطلوبة بطرق كفوءة هو ضرورة معرفة طرق تمثيلها وأساليب التعامل مع هياكلها التمثيلية لذا فإن هياكل البيانات لا تعني تمثيل البيانات في هياكل معينة فقط بل يجب قياس مخططاتها من حيث المساحة التخزينية (space) والوقت (time) إذ أن لكل طريقة مزايا تختلف عن غيرها مما يستوجب اختيار المناسب منها وفق التطبيق المعني .

تنقسم الهياكل إلى نوعين الأول هو الهيكل الفيزيائي ويقصد به المادي أو الحيز الذي تخزن أو تمثل فيه البيانات في ذاكرة الحاسوب (memory) التي تتعامل معها بصورة مصفوفة أحادية من المواقع التخزينية .

أما الهيكل الثاني فهو الهيكل المنطقي وهو الشكل البرمجي أو الأسلوب الذي يتعامل به المبرمج مع تلك البيانات .

فمثلا عند تعريف مصفوفة ثنائية $A[4][5]$ فإنها تمثل في ذاكرة الحاسوب في (20) موقع متعاقب ، والمبرمج عند استخدامه أو عند تعامله مع بيانات هذه المصفوفة باعتبارها مكونة من أربعة صفوف $[3,2,1,0]$ وخمسة أعمدة $[4,3,2,1,0]$ كما تتعامل معها رياضيا ، فالوصول إلى الموقع $A[2][3]$ لا يعني البحث فيزيائيا في الصف الثالث والعمود الرابع لأن مثل هذه الصورة غير موجودة فيزيائيا بل يجب البحث عن الموقع الثاني عشر (بالقراءن استخدام طريقة الصفوف لتمثيل المصفوفة في لغة ++C) ابتداء من أول موقع خدد لتمثيل المصفوفة ، أي أن المبرمج لم يكن ممثلا بكيفية تمثيل بيانات المصفوفة في ذاكرة الحاسوب (التمثيل الفيزيائي) واستخدم خوارزمية الوصول إلى العناصر البينائية للمصفوفة بصيغ برمجية معينة للوصول إلى الحل .

إن وجهة نظر المبرمج هنا تمثل الهيكل المنطقي ، والترابط بين وجهة نظر المبرمج مع الهيكل الفيزيائي الفعلي فتعالجه لغة البرمجة .

Array Structure	1- المصفوفة
File	2- الملف
Linear Structures	3- الهياكل الخطية
Non-linked structures	4- الهياكل غير الموصولة
Stack	+ المكس
Queue	+ الطابور
Circular Queue	+ الطابور الدائري
Linked Structures	+ الهياكل الموصولة
Linked Stack	+ المكس الموصول
Linked queue	+ الطابور الموصول
Non-linear Structures	5- الهياكل غير الخطية
Graphs	+ المخططات
Directed graph	+ المخطط المتجه
Tree structure	+ هيكل الشجرة
Undirected graph	+ المخطط غير المتجه
Size of the data	1. حجم البيانات
Speed & Method of Using the data	2. سرعة وطريقة استخدام البيانات
Data dynamics	3. الديناميكية للبيانات
Storage Space Required	4. كغيرها وتعديلها دوريا
Retrieval Time required to Access any data item	5. السعة التخزينية المطلوبة
Programming Techniques	6. الزمن اللازم لاسترجاع أية معلومة من الهيكل البياني
	6. أسلوب البرمجة

4-1 كفاءة اختيار الهيكل البياني المناسب

لكل مجموعة من البيانات هناك أكثر من طريقة لتنظيمها ووضعها في هيكل بياني معين ويحدد ذلك وفق عدد من العوامل والاعتبارات لاختيار الهيكل البياني المناسب وهي:

1. حجم البيانات
2. سرعة وطريقة استخدام البيانات
3. الديناميكية للبيانات
4. كغيرها وتعديلها دوريا
5. السعة التخزينية المطلوبة
6. الزمن اللازم لاسترجاع أية معلومة من الهيكل البياني
6. أسلوب البرمجة

2-1 هياكل البيانات Data Structures

يمكن تعريف هياكل البيانات بأنها :

دراسة طرق الترابط بين نظرية المبرمجين للبيانات وعلاقة المعلومات بالأجهزة

(وخصوصا ذاكرة الحاسوب التي تخزن فيها البيانات) .

فهيكل البيانات تشمل طرق تنظيم المعلومات ، والخوارزميات الكفوءة في الوصول لها وطرق التعامل معها أو تدار لها (كالإضافة والحذف والتحديث والتزريب والبحث ... الخ) لذا فإن الاهتمام لا ينحصر فقط بأساليب الخزن وخوارزمياته ، لأن الأهمية الحيوية هي قياس كفاءة كل أسلوب من تلك الأساليب ومدى ملاءمة استخدامها في الحالات المختلفة .

3-1 أنواع هياكل البيانات

توفر لغات البرمجة الصنيع المناسبة لتعريف واستخدام العناصر البيانية ذات القيمة الواحدة (المتفرقة) فمثلا في لغة ++C تستخدم التعريفات :

```
int x
float y
char a
long m
short k
```

لتمثل في ذاكرة الحاسوب ويتم التعامل معها بصيغ برمجية بسيطة مثل :

$x = x + 100$

$y = y + 15.6$

وتكاد تكون هذه الصيغ متوفرة في جميع لغات البرمجة بشكل قياسي شبه موحد.

أما بالنسبة للعناصر البيانية التي تتكون من عدة قيم بيانية فإنها تحتاج لاستخدام هيكل بياني مختلف وفيما يلي نذكر أهم تلك الهياكل البيانية

وبرنامج (C++) يتكون عادة من عدد من هذه الوحدات (objects) التي تتضمن بعضها عبر البرامج الفرعية (functions) المعرفة داخل الـ (object) .
في البرمجة الشيئية (object oriented programming) وعند دراسة مسألة ما لبرنامجها فإنه لا يتم تجزئتها إلى وظائف و افعال (functions) فقط ، وإنما يتم تجزئتها إلى

(objects) التي هي وحدات (entities) كل منها يتكون من :

+ Data التي تشمل الخصائص (attributes) أو المعلومات التي سيتبع عليها تأثير العمل أو الفعل (action).

+ Function الذي يمثل السلوك (Behavior) أو العمل (Action) المتوقع تنفيذه على الـ (Data) .

إن هذا التجميع لـ (Data) و (Functions) معاً "بطريقة وصف ما يسمى (object) ، إذ يتم تعريفه باستخدام Class كنوع بياني (data type) يحتاجه المستخدم وسكون مواصفاته في ضوء الترتيب الذي أعطي لكل من الـ (data) ، (functions) المكونين لـ (object) . أي إن الـ (object) هو مثال من النوع البياني (class) .

إن هذه الفاشية أو الأسلوب الذي تعتمد لغات البرمجة الشيئية ومنها لغة (C++) باستخدام مفهوم (class) و (object) يحقق التطبيق البرمجي مع العالم الحقيقي الذي نعمل على مكنته حاسوبياً ، مثل توصيف المركبات (السيارات) عند تمثيلها في نظام المرور ، و توصيف الطلاب عند تمثيلهم في نظام لإدارة الجامعية أو المدرسية ، وغير ذلك من التوصيفات الكثيرة .

5-1 لماذا ندرس هياكل البيانات

تدرس هياكل البيانات للتمكن من كتابة برامج أكثر كفاءة ، ومع أن أجهزة الحاسوب الحديثة تزداد سرعتها باستمرار فإن الحاجة مستمرة أيضاً لبرامج أكثر كفاءة لتنفيذ وظائف أكثر تعقيداً . إن كتابة البرامج الكفوءة يعتمد على حسن تنظيم المعلومات و كفاءة خوارزمية الحل المعتمدة .

عند التعامل مع هياكل البيانات بأسلوب (Object Oriented Approach) يؤخذ بنظر الاعتبار سلوك هذه الهياكل من حيث وقت التنفيذ (Execution Time) و المساحة التخزينية (storage space) المطلوبين ، ثم يتم البحث عن الهيكل البياني المناسب الذي سيسمح بأفضل أداء للعمليات المطلوبة . إن حل هياكل البيانات يوفر لبرامج تستخدم في البرامج التطبيقية لتحديد الهياكل البيانية التي تحقق تنفيذ سريع دون حصول عيب كبير على المساحة التخزينية ، وأن دراسة هذا الموضوع سيتضمن استخدام و تطبيق صيغ برمجية متقدمة بلغة (C++) باعتبارها إحدى لغات البرمجة الشيئية (object oriented language) مثل المؤشرات (pointers) ، التخصيص الغزني الديناميكي (dynamic storage allocation) و الاستدعاء الذاتي (recursion) .

6-1 ماذا يعني (Object Oriented Approach) .

إن الفكرة الأساسية التي تنبني عليها لغة البرمجة الشيئية (Object Oriented - OOL Language) هي تكوين كيانات أو وحدة (object) التي تتكون من :

- Data أي البيانات

- Function أي البرامج الفرعية الذي يتضمن العمليات والوظائف التي

ستنفذ على تلك البيانات .

الفصل الثاني المصفوفة Array

Array	المصفوفة	1-2
Representation of One - Dimensional Array	تمثيل المصفوفة الأحادية في الذاكرة	2-2
Representation of Two - Dimensional Array	تمثيل المصفوفة الثنائية في الذاكرة	3-2
Row - wise	طريقة الصفوف	1-3-2
Column - wise	طريقة الأعمدة	2-3-2
Representation of Three & Four - Dimensional Arrays	تمثيل المصفوفات المتعددة الأبعاد	4-2

7-1 النوع البياني المجرد (ADT - Abstract Data Type)

عند تصميم نوع بياني معين (data type) فمن المهم التمييز بين مهيتين هما :

+ تعريف وظيفته أو عمله (أي غرض استخدامه).

+ طريقة تمثيله حاسوبياً" (Implementation).

أن النوع البياني المجرد هو المعنى بالمهمة الأولى ويعتبر مجموعة من الأشياء (set of objects) مع مجموعة من العمليات (set of operations) التي ستقوم بها .

أي أنه تصور أو تحقيق (أو بناء) النوع البياني (Data Type) كمكون برمجي (software component) يُعرف النوع البياني و العمليات التي يمكن تنفيذها عليه وفق نوع المدخلات (input) ونوع المخرجات (output).

فمثلاً النوع البياني المجرد للمكس (stack ADT) هو هيكل بياني تنفذ فيه عمليتي الإضافة (push) و الحذف (pop)، أما طريقة تمثيله حاسوبياً" فإنه يمكن أن تتم بعدة طرق (كاستخدام المصفوفة أو استخدام القائمة الموصولة مثلاً) .

في لغة البرمجة السي++ (C++) يمكن تمثيل النوع البياني المجرد (ADT) باستخدام (class) الذي يتضمن توصيف النوع البياني (Data Specifications) والعمليات التي ستنفذ عليه (Functions).

كل اخنا سيارة ما ، فإن أهم السمات التي لا يمكن فصلها عنها هي : القيادة ، التسارع ، التوقف . وتصميم هذه السيارة يمكن أن يصوره كدور بياني مجرد (ADT) وفعالياته هي (القيادة ، التسارع ، التوقف) ، وهذا التصميم هو نفسه لجميع أنواع السيارات حتى لو اختلفت الموديلات أو أنواع المحركات مثلاً ، لذا فإن السائق يمكنه استخدام (سيارة) اية سيارة لأنه سيتعامل معها كنموذج لـ (ADT) تتشابه فيها الصفة العالية للسيارة كمركبة وفعاليتها الأساسية هي قيادة - تسارع - توقف.

1-2 المصفوفة Array

هي عبارة عن مجموعة من المواقع الخزنائية في الذاكرة تستخدم وتصف بها بلتي:

1- جميع المواقع تكون من نوع بياني واحد ، حسب صيغة التعريف ، char , int , double , short , float , ... الخ .

2- يمكن الوصول عشوائيا (Randomly accessed) إلى أي موقع من مواقعها دون الاعتماد على أي موقع في المصفوفة فمقدار الوقت المطلوب للوصول إلى أي موقع هو مقدار ثابت

3- مواقع عناصر المصفوفة تبقى ثابتة ولا تتغير أثناء التعامل مع أي من عناصر المصفوفة

4- تمثل المصفوفة في مواقع متعاقبة في الذاكرة

2-2 تمثيل المصفوفة الأحادية في الذاكرة

في لغة C++ تعرف هذه المصفوفة كالآتي :-

int x[10]

وهذا يعني تعريف هيكل بياني يستوعب مجموعة من العناصر البيانية عددها (10) مثلا ، باسم بياني واحد هو (x) ويستخدم الخليل (index) للوصول إلى عنصر بياني محدد في هذا الهيكل ، وتتراوح قيمة الخليل (9 <= i <= 0) .

ويستكون الإشارة إلى كل عنصر فيه كالآتي :

x[0] ، x[1] ، x[2] ، ... ، x[9]

ويوجب هذا التعريف سيجد مخرج اللغة (compiler) المنطقة الخزنائية لاستيعاب مجموعة العناصر البيانية ابتداءً من العنصر الأول للمصفوفة x[0] الذي سيكون في الموقع الخزنائي الأول وهو ما يطلق عليه عنوان البداية (BA) Base Address وليكن افتراضاً هو (500) ، أما العنصر الثاني للمصفوفة x[1] فسيكون عنوانه بعد عنوان البداية مباشرة أي (501) [بافتراض أن طول الكلمة لمنظومة الحاسوب هي خلية واحدة (one cell) أي بايت واحد] ، وهكذا لبقية العناصر بالتتابع ، ويستخدم الخليل (i) بقيمة التي تتراوح بين 9 <= i <= 0 نسبة إلى موقع البداية (500) باستخدام العلاقة التالية:

$$\text{Location (x[i])} = \text{Base Address} + (i)$$

فإذا كان المطلوب تحديد عنوان 'العنصر الرابع في المصفوفة أي (i=3) فسيكون :

2-3-1 طريقة الصفوف Row-wise Method

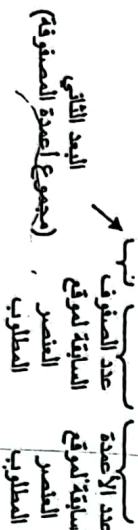
وهذه الطريقة مستخدمة في كل من لغة C++ ، باسكال ، كوبرول الخ حيث تؤخذ جميع عناصر الصف الأول ($i = 0$) للمصفوفة وتخزن في الذاكرة ابتداءً من موقع البداية (Base Address) ولكن 700.

700	أي BA	تألفنصر	$a[0][0]$	يخزن في الموقع
701	أي BA + 1	والعنصر	$a[0][1]$	يخزن في الموقع
702	أي BA + 2	والعنصر	$a[0][2]$	يخزن في الموقع

وهكذا بقية عناصر الصف الأول .

ثم تؤخذ جميع عناصر الصف الثاني ($i = 1$) للمصفوفة وتخزن في الذاكرة ابتداءً من الموقع الذي يلي موقع آخر عنصر من عناصر الصف الأول .
وتخزن جميع عناصر الصف الثالث ($i = 2$) للمصفوفة في الذاكرة ابتداءً من الموقع الذي يلي موقع آخر عنصر من عناصر الصف الثاني ، وهكذا بقية الصفوف .
ولهذا فإن احتساب موقع العنصر $a[i][j]$ سيكون وفق العلاقة التالية :-

$$\text{Location (a[i][j])} = \text{Base Address} + d2 * (i) + (j)$$



وهذه العلاقة الرياضية هي التي يحسب مخرج اللغة بموجبها موقع العنصر المطلوب معالجة بموجب كل إيعاز من إيعازات البرنامج .

مثال :

لدينا تعريف المصفوفة التالية :

المطلوب احتساب موقع العنصر $a[5][3]$ بافتراض أن عنوان البداية في الذاكرة هو (BA=900)

$$\text{Location (x[3])} = 500 + (3) = 503$$

أي أن موقع (عنوان) العنصر الرابع للمصفوفة سيكون في الخلية (503)
لأن العنصر الأول $x[0]$ سيكون في الموقع 500
و العنصر الثاني $x[1]$ سيكون في الموقع 501
و العنصر الثالث $x[2]$ سيكون في الموقع 502

فتعنيما يتبعن البرنامج البرنامج أية إشارة أو تعامل مع عناصر المصفوفة في أي إيعاز مثل
لحل قيمة متغير $cin >> x[i]$
أو اخراج قيمة متغير $cout << x[i]$
أو إيعاز $x[2]=20$
أو غير ذلك فإن المترجم (compiler) سيعتمد العلاقة المشار إليها سابقاً لتحديد الموقع المطلوب أخذ المعلومة منه أو وضعها فيه .

2-3-2 تمثيل المصفوفة الثنائية في الذاكرة

هناك طريقتان لتمثيل المصفوفة الثنائية هما طريقة الصفوف (Row-wise method) ، وطريقة الأعمدة (Column-wise method) .
لنأخذ التعريف التالي للمصفوفة

int d1=10 ; int d2=10;
int a[d1][d2];

وهذا يعني في لغة C++ تعريف لهيكل بياني اسمه (a) يستوعب مجموعة من العناصر البيانية عددها (10 * 10) ويستخدم دليلين للوصول إلى أي عنصر بياني فيه وهما (i, j) حيث أن :
 $d1-1 \leq i \leq 0$ لتحديد الصف الذي فيه العنصر
 $d2-1 \leq j \leq 0$ لتحديد العمود الذي فيه العنصر
فمثلاً العنصر $a[2][4]$ حيث أن $i = 2, j = 4$ ،
سيعني العنصر الذي يقع في الصف الثالث والعمود الخامس (لأن ترتيب / تسلسل كل من الصفوف و الأعمدة يبدأ من 0) .

أن مترجم اللغة (compiler) يعتمد عادة إحدى الطريقتين الآتيتين لتمثيل هذه المصفوفة :-

2-3-2 طريقة الأعمدة Column - Wise Method

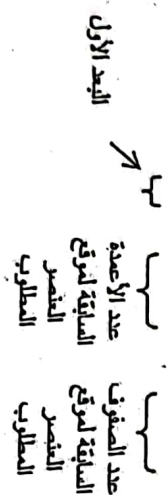
هذه الطريقة مستخدمة في كل من مترجمات لغة فورتران وبيسك وغيرهما ،
إذ تؤخذ جميع عناصر العمود الأول ($j=0$) للمصفوفة وتُخزن في الذاكرة ابتداءً من
موقع البداية (Base Address) وليكن (200)

200	أي	BA	يُخزن في الموقع	a [0][0]	فالعنصر
201	أي	BA+1	يُخزن في الموقع	a [1][0]	والعنصر
202	أي	BA+2	يُخزن في الموقع	a [2][0]	والعنصر

وهكذا تبقى عناصر العمود

ثم تؤخذ جميع عناصر العمود الثاني ($j=1$) للمصفوفة وتُخزن في الذاكرة ابتداءً
من الموقع الذي يلي موقع آخر عنصر من عناصر العمود الأول.
وتُخزن جميع عناصر العمود الثالث ($j=2$) للمصفوفة في الذاكرة ابتداءً من الموقع
الذي يلي موقع آخر عنصر من عناصر العمود الثاني وهكذا يُلَاقى باقي الأعمدة.
وعليه فإن احتساب موقع العنصر $a [i][j]$ يكون وفق العلاقة التالية :-

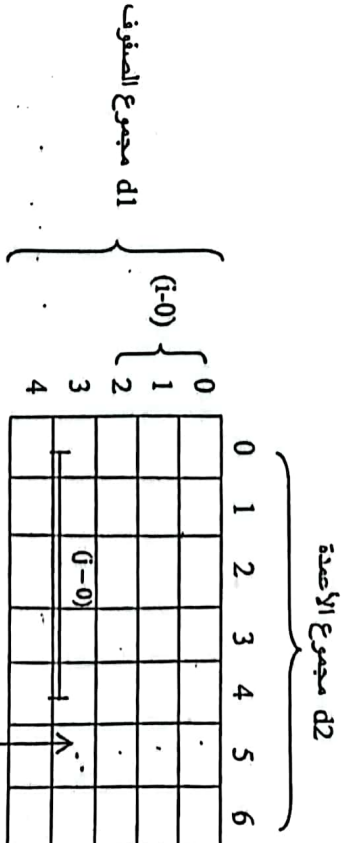
$$\text{Location (a [i][j])} = \text{Base Address} + d1 \cdot (j) + (i)$$



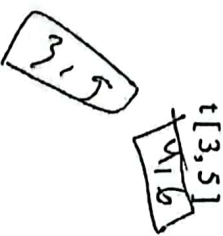
الحل: بما أن المطلوب هو العنصر $t[3][5]$ ، أي أن $i=3$ ، $j=5$ ، فهذا يعني
أن العنصر يقع في الصف الرابع (لأن ترتيب الصفوف في لغة C++ يبدأ من 0) ،
ويقع في العمود السادس (لأن ترتيب الأعمدة في لغة C++ يبدأ من 0) ، وبما أن
مجموع الصفوف (البعد الأول) هو (5) ، ومجموع الأعمدة (البعد الثاني) هو (7)
كما ورد في التعريف أعلاه ، لذا تصبح العلاقة الرياضية بعد التعويض فيها كما يلي :

$$\text{Location (t[3][5])} = \text{BA} + 7 * 3 + 5$$

$$= 900 + 21 + 5 = 926$$



الشكل (2 - 1)



4-2 تمثيل المصفوفات المتعددة الأبعاد

إن المصفوفات متعددة الأبعاد (الأكثر من بعدين) تكون عناصرها هي مصفوفات أيضا. فالمصفوفة الثلاثية الأبعاد يكون كل عنصر من عناصر البعد الأول هو مصفوفة ذو بعدين. والمصفوفة الرباعية الأبعاد يكون كل عنصر من عناصر البعد الأول هو مصفوفة ثلاثية الأبعاد. وهكذا للمصفوفات بأبعاد أخرى.

فيما يلي الصيغة الرياضية العامة لكل من طريقتي التمثيل (طريقة الصفوف) و (طريقة الأعمدة) لاحتساب موقع أي عنصر بياني في مصفوفة متعددة الأبعاد (عدد أبعادها d) ومعرفة باغة C++ بالشكل التالي:

$int\ z[m_1][m_2] \dots [m_d]$

/ التمثيل بطريقة الصفوف :

إن العلاقة الرياضية التي يتم الاحتساب بموجبها هي :

$$Location(z[i_1][i_2] \dots [i_d]) = BA + \sum_{k=1}^d (\prod_{m=k+1}^d N_m) i_k$$

حيث أن (d) هو عدد الأبعاد للمصفوفة (ثلاثية، ثنائية، رباعية، ... الخ) $N_1, N_2, \dots, N_d$ قيمة كل بعد من أبعاد المصفوفة $i_1, i_2, \dots, i_d$ قيمة الدليل (index) لكل بعد

ن/ التمثيل بطريقة الأعمدة :

إن العلاقة الرياضية التي يتم الاحتساب بموجبها هي :

$$Location(z[i_1][i_2] \dots [i_d]) = BA + \sum_{k=1}^d (\prod_{m=1}^{k-1} N_m) i_k$$

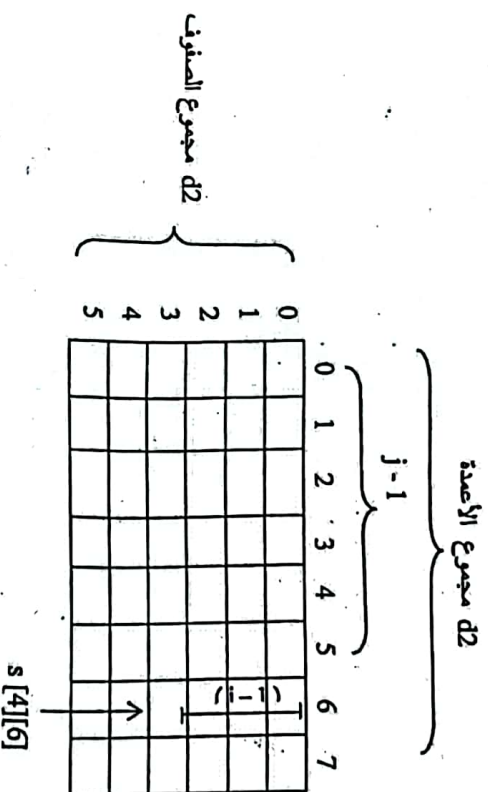
حيث أن (d) هو عدد الأبعاد للمصفوفة (ثلاثية، ثنائية، رباعية، ... الخ) $N_1, N_2, \dots, N_d$ قيمة كل بعد من أبعاد المصفوفة $i_1, i_2, \dots, i_d$ قيمة الدليل (index) لكل بعد

مثال :-
لدينا تعريف المصفوفة التالية

$int\ s[6][8]$

المطلوب احتساب موقع العنصر $s[4][6]$ بالقرائن أن عنوان موقع البداية في الذاكرة هو (BA = 300)

الحل :-



الشكل (2-2)

بما أن المطلوب هو العنصر $s[4][6]$ أي أن $i=4, j=6$ فهذا يعني أن العنصر يقع في الصف الخامس (لأن ترتيب الصفوف في لغة C++ يبدأ من 0)، ويقع في العمود السابع (لأن ترتيب الأعمدة في لغة C++ يبدأ من 0)، وبما أن (البعد الأول) مجموع صفوف المصفوفة هو (6) والبعد الثاني مجموع أعمدة المصفوفة هو (8) كما في التعريف.

لذا نصبح العلاقة الرياضية بعد التعويض فيها كما يأتي :-

$$\begin{aligned} Location(s[4][6]) &= BA + 6 * (6) + (4) \\ &= 300 + 36 + 4 \\ &= 340 \end{aligned}$$

1- لطريقة الصفوف

$$\text{Loc}(z[n_1][n_2] \dots [n_n]) = BA + n_2 n_3 \dots n_n i_1 + n_3 n_4 \dots n_n i_2 + \dots + i_n$$

2- لطريقة الأعمدة

$$\text{Loc}(z[n_1][n_2] \dots [n_n]) = BA + i_1 + n_1 i_2 + n_1 n_2 i_3 + \dots + n_1 n_2 \dots n_{n-1} i_n$$

و / ملاحظة مهمة

في حالات لغات البرمجة التي يكون تعريف دليل (index) المصفوفة فيها في $[i=1 \dots n]$ ، فإنه يمكن استخدام نفس المعادلات التي تقدم ذكرها على أن يستبدل الدليل (i) بـ $(i-1)$ أيضا ورد ذلك.

ن / ولغرض التوصل الى صيغة أكثر شمولية فإنه يمكن استبدال الدليل (i)

بـ $(i-l)$ أيضا ورد ، حيث أن (l) تعني (lower bound) أي القيمة الدنيا لدليل المصفوفة (array index).

فإذا كان تعريف المصفوفة $[0 \dots n]$ تكون قيمة $(l=0)$ كما في لغة ++C.
وإذا كان تعريف المصفوفة $[1 \dots n]$ تكون قيمة $(l=1)$ كما في لغة باسكال.
وإذا كان تعريف المصفوفة $[8 \dots -3]$ تكون قيمة $(l=-3)$ كما في لغة باسكال.

ج / احتساب عنوان موقع عنصر لمصفوفة ثلاثية الأبعاد :

لناخذ تعريف المصفوفة التالية :

int z [n1][n2][n3]

1 - العلاقة الرياضية للاحتساب بطريقة (Row-wise)

$$\text{Location} (z [i_1][i_2][i_3]) = BA + n_2 n_3 i_1 + n_3 i_2 + i_3$$

2 - العلاقة الرياضية للاحتساب بطريقة (Column-wise)

$$\text{Location} (z [i_1][i_2][i_3]) = BA + i_1 + n_1 i_2 + n_1 n_2 i_3$$

د / احتساب عنوان موقع عنصر لمصفوفة رباعية الأبعاد :

لناخذ تعريف المصفوفة التالية :

int z [n1][n2][n3][n4]

1. العلاقة الرياضية للاحتساب بطريقة (Row-wise)

$$\text{Location} (z [i_1][i_2][i_3][i_4])$$

$$= BA + n_2 n_3 n_4 i_1 + n_3 n_4 i_2 + n_4 i_3 + i_4$$

2. العلاقة الرياضية للاحتساب بطريقة (Column-wise)

$$\text{Location} (z [i_1][i_2][i_3][i_4])$$

$$= BA + i_1 + n_1 i_2 + n_1 n_2 i_3 + n_1 n_2 n_3 i_4$$

هـ / مصفوفات بأبعاد أخرى

بإمكان استخدام الصيغ الرياضية التي وردت في (أ، ب) لاكتساب مواقع عناصر لمصفوفات بأبعاد أخرى لتصبح الصيغة العامة كالآتي :

تمرين:

استخدم تعريف لغة ++C للمصفوفة الرباعية التالية :-

int bob[5][8][7][9].

واصحب موقع (عنوان) العنصر bob[3][6][5][7] بموجب كل من طريقة

المصفوف وطريقة الأعمدة إذا كان عنوان البداية (Base Address = 800)

الحل:

The dimensions of bob are :

$$n_1 = 5, n_2 = 8, n_3 = 7, n_4 = 9$$

To compute the location of the element bob [3][6][5][7]

This means the indices are :

$$i_1 = 3, i_2 = 6, i_3 = 5, i_4 = 7$$

Row - Wise

طريقة الصفوف:

$$\text{Location (bob}[i_1][i_2][i_3][i_4]) = BA + n_2 n_3 n_4 i_1 + n_3 n_4 i_2 + n_4 i_3$$

+ i_4

$$\text{Location (bob}[3][6][5][7]) = 800 + 8 * 7 * 9 * 3 + 7 * 9 * 6 * 5 + 7$$

$$= 800 + 1512 + 378 + 45 + 7$$

$$= 2742$$

Column - Wise

طريقة الأعمدة:

$$\text{Location (bob}[i_1][i_2][i_3][i_4]) = BA + i_1 + n_1 i_2 + n_1 n_2 i_3$$

+ $n_1 n_2 n_3 i_4$

$$\text{Location (bob}[3][6][5][7]) = 800 + 3 + 5 * 6 + 5 * 8 * 5 + 5 * 8 * 7 * 7$$

$$= 800 + 3 + 30 + 200 + 1960$$

$$= 2993$$

تمرين:

استخدم تعريف لغة ++C للمصفوفة الثلاثية التالية :-

int box[9][6][8]

واصحب موقع (عنوان) العنصر box[5][3][6] بموجب كل من طريقة

المصفوف وطريقة الأعمدة إذا كان عنوان البداية (Base Address = 1200)

الحل:

The dimensions of box are :

$$n_1 = 9, n_2 = 6, n_3 = 8$$

To compute the location of the element box [5][3][6]

This means the indices are :

$$i_1 = 5, i_2 = 3, i_3 = 6$$

Row - Wise

طريقة الصفوف:

$$\text{Location (box}[i_1][i_2][i_3]) = BA + n_2 n_3 i_1 + n_3 i_2 + i_3$$

$$\text{Location (box}[5, 3, 6]) = 1200 + 6 * 8 * 5 + 8 * 3 + 6$$

$$= 1200 + 240 + 24 + 6$$

$$= 1470$$

Column - Wise

طريقة الأعمدة:

$$\text{Location (box}[i_1][i_2][i_3]) = BA + i_1 + n_1 i_2 + n_1 n_2 i_3$$

$$\text{Location (box}[5, 3, 6]) = 1200 + 5 + 9 * 3 + 9 * 6 * 6$$

$$= 1200 + 5 + 27 + 324$$

$$= 1556$$

الفصل الثالث المكدس والطاير

Linear List	القائمة الخطية	1-3
Types of linear lists	أنواع القوائم الخطية	1-1-3
Operations on linear Lists	العمليات التي يمكن إجراؤها على القوائم الخطية	2-1-3
Stack	المكدس	2-3
Array Representation of Stack	تمثيل المكدس باستخدام المصفوفة	1-2-3
Stack's Algorithms	خوارزميات المكدس	2-2-3
Stack's subprograms	البرامج الفرعية لتنفيذ عمليات المكدس	3-2-3
Structure representation of Stack	تمثيل المكدس باستخدام القيد	4-2-3
Stack's Applications	أهم تطبيقات المكدس	5-2-3
Queue	الطاير	3-3
Array Representation of Queue	تمثيل الطاير باستخدام المصفوفة	1-3-3
Queue's Algorithms	خوارزميات الطاير	2-3-3
Queue's Subprograms	البرامج الفرعية لتنفيذ عمليات الطاير	3-3-3
Structure Representation of queue	تمثيل الطاير باستخدام القيد	4-3-3
Queue's Applications	تطبيقات الطاير	5-3-3
Circular Queue (CQ)	الطاير الدائري	4-3
Double Ended Queue	الطاير المزدوج	5-3

أسئلة الفصل

1. What do we mean by data structures? Explain that in detail
2. What are the classifications of data structures?
3. What are the main factors for selection the required data structure?
4. What are the characteristics of the array?
5. Let X : array [50] of integer
What is the address of the element X[33] if the base address is (BA = 970) ?
6. Let A : array [M][N] of integer
How you can compute the address of the general element A[i][j] using row – wise method?
7. Let X : array [A][B][C] of integer
What is the address of the general element X [i][j][k] using column – wise method ?
8. Let S : array [7][10][8] of char
Compute the location of the element S[5][9][3] using row – wise and column – wise methods when the base address is 1200 .
9. Let A : array [9][5][8][6] of integer
Compute the location of the element A[8][3][6][4] using row – wise and column – wise methods when the base address is 950 .

1-3 القائمة الخطية Linear List

هي مجموعة من العناصر البينائية (elements , nodes , items) المتسلسلة والمرتبطة ، تربط عناصرها علاقة تجاور بحيث يسبق كل عنصر عنصراً عاد العنصر الأول الذي لا يسبقه عنصر آخر والعنصر الأخير الذي لا يليه عنصر فلو مثلاً كل عنصر على شكل عقدة (node) فإن القائمة تصبح مجموعة من العقد (n).

$x[n], x[n-1], \dots, x[k+1], x[k], \dots, x[3], x[2], x[1]$

فالعقدة الأولى هي $x[1]$ والعقدة الأخيرة هي $x[n]$ أما العقدة $x[k]$ عندما 1

$n \leq k < n$ فإن العقدة التي تسبقها هي $x[k-1]$ والتي تليها هي $x[k+1]$ فمثلاً :
إن كل مجموعة من البيانات والمعلومات يمكن تسميتها قائمة (list) فمثلاً :

+ مجموعة أسماء طلبة كلية ما مرتبة حسب الحروف.

+ مجموعة أسماء المشتركين في دليل الهاتف مرتبة وفق نسق معين .

1-1-3 أنواع القوائم الخطية

أ- القوائم غير الموصولة Non-linked lists

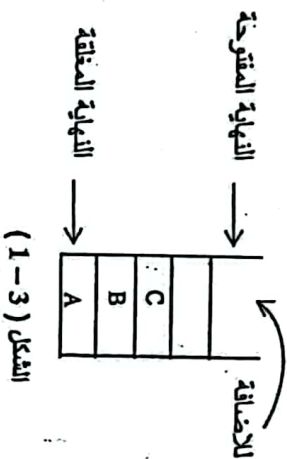
وهي القوائم التي لا تستخدم المؤشرات وتكون على شكل بيانات متتابعة ومتجاورة (sequential) وتستخدم الصفوفات في تمثيلها كما يستخدم هذا النوع عند معالجة البيانات التي لا تعرض للتغير كثيراً لصعوبة عمليات الحذف والإضافة إذ قد تكون المواقع التالية في ذاكرة الحاسوب مشغولة أصلاً مما يعترض استخدامها لأغراض الحذف والإضافة .

ب - القوائم الموصولة Linked lists

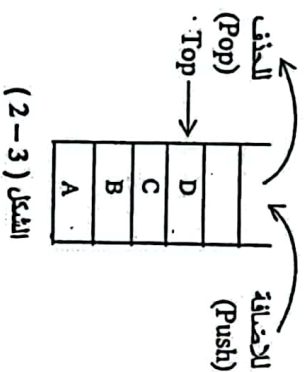
وهي القوائم التي تستخدم المؤشرات (pointers) لتسهيل عمليات الإضافة والحذف والتعديل إذ يكون لكل عنصر مؤشر يحدد موقع العنصر التالي ، ووجود المؤشرات يلغي الحاجة لغزول بيانات القائمة في مواقع خزانة متجاورة .

2-3 المكدس Stack

هو عبارة عن قائمة خطية تتم فيها عمليات الإضافة والحذف من إحدى نهايتي القائمة وتكون النهاية الأخرى مغلقة.



نأخذ المكدس الموضح في الشكل إذ نجده يحتوي على العناصر A, B, C وعلد إضافة عنصر جديد مثل (D) يجب أن تكون الإضافة من الجهة المقترحة ليصبح الشكل كالآتي :



وعند حذف عنصر من المكدس يجب أن نستخدم نفس الجهة المقترحة فقط أي نستطيع أن نأخذ العنصر (D) ثم نأخذ العنصر (C) بالتتابع ولا نستطيع أن نأخذ العنصر (C) قبل أن نأخذ العنصر (D) ، مع ملاحظة أن العنصر (D) دخل أخيراً. ولهذا نستطيع أن نلخص عمل المكدس بالمعادلة الآتية :

(آخر من يدخل أول من يخرج) (Last In First Out (LIFO)

كما أنه لا يمكن أخذ (حذف) عنصر من وسط عناصر المكدس إلا بعد حذف (إخراج) العناصر التي تسبقه من جهة النهاية المقترحة مع التأكيد على أن النهاية الأخرى مغلقة ولا تستخدم أبداً.

2-1-3 العمليات التي يمكن إجراؤها على القوائم الخطية

The Operations on the linear lists

يمكن تنفيذ عدد من العمليات (العمليات) على أي هيكل بياني عند معالجة بياناته وفيما يلي أهم أنواع هذه العمليات التي يمكن تنفيذ بعضها أو كلها حسب التطبيق .

1- البحث Search

هي عملية بحث داخل الهيكل البياني بقصد الوصول إلى عنصر (عقدة) معين فيه بموجب قيمة أحد الحقول يسمى حقل المفتاح (key field) أي أن البحث يتم وفق المحاورات وليس العنوان .

2- إدخال (إضافة) (Insertion) Addition

إضافة عنصر (عقدة) جديد إلى الهيكل البياني مثل تسجيل طالب جديد في المدرسة.

3- حذف Deletion

حذف عنصر (عقدة) من الهيكل البياني ، مثل نقل طالب إلى مدرسة أخرى .

4- دمج Merge

دمج بيانات هيكلين أو أكثر لتكوين هيكل بياني واحد .

5- فصل Split

تجزئة بيانات هيكل بياني إلى هيكلين أو أكثر .

6- احصائي Counting

احصائي عدد العناصر أو العقد في الهيكل البياني .

7- نسخ Copying

نسخ بيانات الهيكل البياني إلى هيكل بياني آخر .

8- ترتيب Sort

ترتيب عناصر (عقد) الهيكل البياني وفق قيمة حقل (field) أو مجموعة حقل.

9- الوصول Access

تتطلب أحياناً الحاجة للوصول إلى عنصر (عقدة) بياني في الهيكل البياني لعدة أغراض لاختياره مثلاً أو تغييره ... الخ.

مثال:
إذا كانت مجموعة مدخلات المكس بترتيب 1,2,3,4,5 بين أيا من المخرجات
المبينة أدناه صحيحة وفق أسلوب عمل المكس ..

- أ- 1 3 5 4 2
ب- 1 3 5 2 4
ج- 3 2 1 5 4
د- 1 2 3 4 5

الحل:
الفرع أ: المخرجات المطلوبة (1,2,3,4,5)

لاخراج العنصر (2) يجب أولا إدخال العنصرين 1,2 أي أن تسلسل تنفيذ
العمليات هو SSU

1

أي أن محتويات المكس تصبح

ولإخراج العنصر (4) بعد العنصر (2) يجب إدخال العنصرين 3,4 أي أن تسلسل
تنفيذ العمليات في هذه الحالة هو SSUSSU وتصبح محتويات المكس :

3
1

ولإخراج العنصر (5) بعد العنصر (4) يجب إدخاله أولا ثم إخراجيه أي أن تسلسل
تنفيذ العمليات يكون SSUSSUSU وتصبح محتويات المكس :

3
1

وفق حالة المكس الحالية ولكن لإخراج العنصرين 3,1 بالتتابع أي أن تسلسل تنفيذ
العمليات هو SSUSSUSU.
إن يمكن الحصول على مثل هذه المخرجات إذا كان تسلسل العمليات بالصيغة
الأخيرة مع الالتزام بترتيب المدخلات.

وتسمى عملية الإضافة إلى المكس (push) أو (Insertion) وعملية الحذف من
المكس (pop) أو (Deletion).

مثال:

نفرض (S) تعني (Stacking) أي ترمز لعملية إضافة عنصر إلى المكس و (U)
تعني (Unstacking) أي ترمز لعملية حذف عنصر من المكس وكانت مجموعة
المدخلات للمكس بالترتيب M, B, Y, N, R. بين ما هي المخرجات بعد تنفيذ كل
سلسلة من العمليات التالية :

1 / SSUSSUSU

ب / SSUSSUSU

الحل:

وقد بترتيب المدخلات أنه عند تنفيذ عملية إدخال عنصر إلى المكس فإن اختيار
العنصر يكون من تلك المدخلات بالتتابع أي تأخذ M أولا ثم B ... وهكذا، ولا
نستطيع اخذ العنصر N قبل العناصر السابقة له.

أ-

→ المدخلات M B Y N R
→ سلسلة العمليات S S U U S U S U
→ المخرجات B M Y N R

ب-

→ المدخلات M B Y N R
→ سلسلة العمليات S S S U S U S U
→ المخرجات Y N B R M

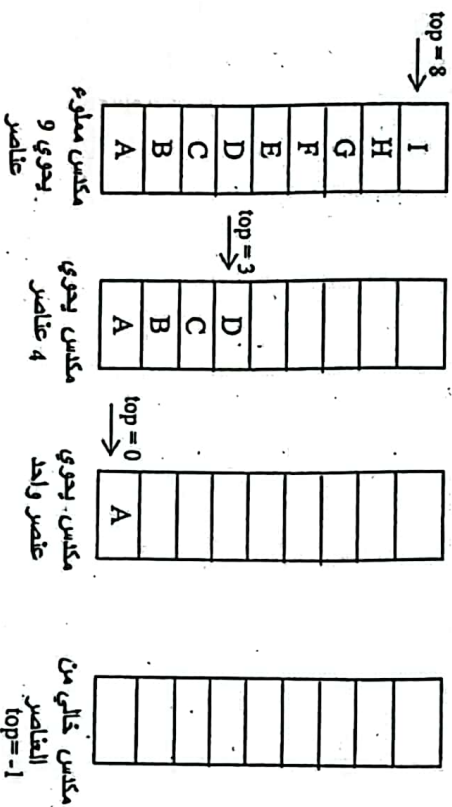
1-2-3. تمثيل المكس باستخدام المصفوفة

Array Representation of Stack

يمكن تمثيل المكس حاسوبياً باستخدام مصفوفة أحادية البسطة المطلوبة (size) وبالترتيب المناسب للبيانات (Data Type) التي سنخزن فيه (int, float, ... الخ) مع استخدام متغير مستقل يسمى (top) يشتمل كموشر يشير إلى موقع أعلى عنصر في المكس (موقع أقرب عنصر إلى النهاية المتوقعة)، وإبتداء تكون قيمة المؤشر $(top = -1)$ عندما يكون المكس خالياً من العناصر، ويُعرف المكس برمجياً بأنه كالآتي:-

```
const int size = 9;    { or any other value }
int stackelement;      { or any other type }
stackelement stack[size];
int top;
```

ملاحظة: مواقع المصفوفة في لغة (C++) تبدأ من (0) وليس من (1)، وبالتالي لهذا المثال ستكون المواقع التسعة من (0-8)



الشكل (3-3)

الفرع ب.: المخرجات المطلوبة (4, 2, 3, 1, 5)

لاخراج العنصر (4) يجب أولاً إدخال العناصر 1, 2, 3, 4 وفق سلسلة العمليات SSSSU وتصيب محركات المكس:

3
2
1

ولإخراج العنصر (2) من المكس بحالته الحالية يجب إخراج العنصر (3) قبله لذا فإن هذا التسلسل من المخرجات (5, 4, 2, 3, 1) لا يمكن تنفيذه.

الفرع ج.: المخرجات المطلوبة (4, 5, 1, 2, 3)
يمكن إخراج العنصرين 5, 4 بعد تنفيذ سلسلة العمليات الآتية:

المدخلات	→	1	2	3	4	5
سلسلة العمليات	→	S	S	S	S	U
المخرجات	→				4	5

وستصبح محركات المكس:

3
2
1

وهنا سيتغير إخراج العنصر (1) قبل العنصرين (2, 3) لذا فإن تسلسل المخرجات (4, 5, 1, 2, 3) غير صحيح.

الفرع د.: المخرجات المطلوبة (4, 3, 5, 2, 1)

يمكن الحصول على هذه المخرجات عند تنفيذ عمليات الإدخال والإخراج بالتسلسل الآتي:

المدخلات	→	1	2	3	4	5
سلسلة العمليات	→	S	S	S	S	U
المخرجات	→				4	3
					5	2
					1	

2- خوارزمية الحذف POP Algorithm.

```

If      Stack is Empty
Then   under flow ← True
Else

```

```

    under flow ← false
    element ← stack [Top]
    Top ← Top - 1

```

3- خوارزمية ملء المكس Stack full

هذه الخوارزمية للتحقق من هل أن المكس مملوء أم لا اعتمادا على قيمة المؤشر (top) قبل عمليات الإضافة، إذ يجب التحقق من هذه الحالة قبل عملية إضافة أي عنصر بياني لأنه سيكون من المتعذر تنفيذ أية عملية إضافة إلى مكس مملوء.

```

If      top = size - 1
Then   stackfull ← true

```

4- خوارزمية خلو المكس Stack Empty

هذه الخوارزمية للتحقق من هل أن المكس خال أم لا اعتمادا على قيمة المؤشر (top) قبل عمليات الحذف، إذ يجب التحقق من هذه الحالة قبل عملية حذف أي عنصر بياني لأنه سيكون من المتعذر تنفيذ أية عملية حذف إلى مكس خال.

```

If      top = - 1
Then   stackempty ← true

```

5- خوارزمية إخلاء المكس ClearStack

هذه الخوارزمية تستخدم لغرض تهيئة المكس وإخلائه من العناصر بجعل قيمة المؤشر (top = - 1)

```
Top ← (- 1)
```

عملية الإضافة للمكس (Push)

لتنفيذ عملية الإضافة بشكل صحيح تتبع الخطوات الآتية :

- 1- التحقق من كون المكس غير مملوء (not full) أي أن المؤشر (top < size) لتجنب حالة الفيض (over flow) ونحذف تنفيذ عملية الإضافة.
- 2- تحديث قيمة المؤشر (top + 1) ليؤشر إلى الموقع التالي.
- 3- إضافة العنصر الجديد Stack [top]

عملية الحذف من المكس (Pop)

أن تنفيذ عملية حذف أي عنصر من المكس يجب أن تكون وفق الخطوات الآتية:

- 1- التحقق بأن المكس غير خال (not empty) أي أن المؤشر (top > -1) لتجنب حالة الفيض (under flow) ونحذف تنفيذ عملية الحذف.
- 2- اخذ العنصر من الموقع الذي يشير إليه (top) وخرزه وقتها في متغير مستقل item = Stack [top]
- 3- تحديث قيمة المؤشر (top - 1) ليؤشر إلى موقع العنصر التالي للعنصر الذي تم حذفه.

ملاحظة :

يقتضى أعلاه أن الخطوتين 2 ، 3 في عملية الحذف معكوبة الترتيب عنها في عملية الإضافة.

2-2-3 خوارزميات المكس Stack's Algorithms

يمكن تصميم (اعداد) مجموعة من الخوارزميات لتغطية فعاليات المكس ومن ثم برمجتها لتنفذها عمليا .

1- خوارزمية الإضافة Push Algorithm

```

If      Stack is full
Then   Overflow ← True
Else
    Overflow ← false
    Top ← Top + 1
    Stack [Top] ← New element

```

2- إضافة عنصر يتالي واحد الى المكس (Push)

```
void push(int item)
{
    if (top == size)
        cout<<"\n The stack is Full ";
    else
        stack[++top] = item;
}
```

هذه الخطوات هي لإضافة عنصر واحد فقط ، ويمكن استخدامها داخل أحد ايعازات التكرار مثل (for ... do) او (while ... do) بنا يسمح بقراءة (الدخال) قيمة العنصر الجديد (item) مع كل دورة تكرار ومن ثم ادخاله (push) الى المكس

4- حذف عنصر يتالي واحد من المكس (Pop)

```
int pop()
{
    if (top == -1)
        cout<<"\n The stack is Empty ";
    else
        item = stack[top--];
    return item;
}
```

هنا أيضا هذه الخطوات هي لحذف عنصر واحد فقط ، ويمكن استخدامها داخل أحد ايعازات التكرار بما يسمح بأخذ العنصر الذي يشير اليه (top) من المكس ونسخه (حفظه) في المتغير (item) لاستخدامه لاحقا و إجراء أية عملية مطلوبة عليه او له، ومن ثم الاستمرار في ذلك مع كل دورة تكرار.

Stack Subprograms

بما ان المكس هو هيكيل بياني خصائصه (attributes) هي : حجمه size ، ونوع البيانات المخزنة فيه data type ، وأهم عملياته (actions) هي الاضافة (Push) و الحذف (Pop) ، لذا فان تعريفه برمجيا بلغة C++ يكون بما يسمى (Class) ، وستكون مكوّناتها كالآتي :

1- نفترض وجود التعريفات البرمجية التالية :

```
const int size = 20; { or any other integer value }
int stackelement; { or any other type }
stackelement stack[size];
int top;
```

2- برنامج فرعي (function) لإخلاء المكس

```
void initialize();
{ top = -1; };
```

لاحظ عدم الحاجة للمرور على جميع مواقع الصفوفة وجعلها مساوية للصفر والاكتفاء فقط بجعل المؤشر (top = -1) وهذا البرنامج الفرعي (function) يُستدعى في بداية العمل لجعل المكس خالياً .
ويمكن أيضا استخدام (constructor) في لغة C++ لتحقيق نفس الهدف باستخدام اسم الـ class بالصيغة التالية :

```
Stack ()
{ top=-1; };
```

3-2-3 البرامج الفرعية لتنفيذ عمليات المكس

```

        cout<<"\n The stack is Empty " ;
        return NULL;
    }
    else
    {
        item = arr[top];
        arr[top]=NULL;
        top--;
        return item;
    }
}
};

```

47

تكوين Class Stack
 لتكوين الكلاس برمجيا بلغة C++ فانه يمكن تصحيح المكونات التي سبق ذكرها
 لتكوين الكلاس برمجيا بلغة Class Stack وبالصفحة البرمجية التالية :

```

const int size=10;
class stack
{
private
    int arr[size];
public
    int top;
    stack() // this is construct
    {
        top=-1;
    }
    void push(int item) // Addition function
    {
        top++;
        if( top < size )
            arr[top]=item;
        else
        {
            cout<<"\n The stack is Full ";
            top--;
        }
    }
    int pop() // Deletion function
    {
        if( top== -1 )
        {

```

46

st:

[illegible]

0 1 2 3 4 5 6 7 8 9

st. elements
st. elements [3]

st. top

॥३॥

```
int POP( )
```

تسرين : اكتب الأبعاد الثلاثة عناصر من الأعداد الصحيحة إلى المكس (set) الذي سمته (20) عنصر فقط .

```
for(int j=1; j<=3; j++)
```

49

في التمثيل السابق للمكس باستخدام المصفوفة ورد تعريف المؤشر (top) كمثبتر مستقل عن المصفوفة التي تمثل المكس ، إلا أننا هنا نستخدم ما نوفره لغة ++C ما يسمى (Structure) في تمثيلها معا كيهكل بياني واحد حيث يتكون هذا التركيب من جزأين الأول يمثل المكس وهو على شكل مصفوفة والجزء الثاني هو حقل يمثل المؤشر (top) ويُعرف في لغة ++C بالطريقة التالية :

structure stack

```
int top ;
```

```
    } st;
```

A	B	C										2
0	1	2	3	4	5	6	7	8	9			

st:

لاضاقة عنصر جديد لهذا المكس تتبع الخطوات التالية :

المؤثر هي (3)

```
st.elements[st.top]=D
```

```
cout<<"\n Enter the new item : " ;
cin>>item ;
table[top]=item ;
}
```

في هذا المقطع لم نضع خطوة التحقق من امتلاء المكس لكونها غير ضرورية في هذا المثال لأن سعة المكس هي (30) ، ويحتوي على (4) عناصر فقط بالإضافة المطلوبة هي (8) فقط لذا فإن المكس لن يصل إلى حالة الامتلاء .
مثال : اكتب الأبعادات المطلوبة لحذف (4) عناصر من نوع الأعداد الحقيقية من المكس (bob) الذي سعة (15) عنصر .

الحل : أن المكس المطلوب يمثل المصفوفة (bob) بسعة (15) عنصر ونوع البيانات (float)

```
float pop()
{
    for (int i=1 ; i<=4 ; i++)
    {
        if (top == -1)
            cout<<"\n The tack is Empty " ;
        else
        {
            item = bob[top] ;
            top -- ;
        }
    }
    return item ;
}
```

```
else
{
    cout<<"\n input the new item : " ;
    cin>>item ;
    ++top ;
    st[top]=item ;
}
```

تضمن هذا المقطع من الأبعادات خطوة التحقق من امتلاء المكس داخل إيجاز التكرار لتفقد عند كل عملية إضافة مع أن سعة المصفوفة (المكس) هي (20) ، والسبب في ذلك هو أننا لا نعرف عدد العناصر الموجودة فعلا في المكس قبل الإضافة .

مثال : المكس (table) بسعة (30) عنصر يحتوي على أربعة عناصر A , B , C , D ، اكتب الأبعادات المطلوبة لإضافة (8) عناصر أخرى .

الحل : أن المكس المطلوب يمثل بالمصفوفة (table) وسعتها (30) ونوع بياناته (char) هو

```
void push8 ( char item )
{
    top=3 ;
    for (int j=1 ; j<=8 ; j++)
    {
        ++top ;
    }
}
```


3- صيغة Postfix Notation

لا تكون إشارة العملية الحسابية تلحق العوامل مثل :

(RPN) Reverse Polish Notation وتسمى X 20 / , A B - , 34 +

لأنها عكس الحالة العادية (Polish Notation).

ملاحظة:

لتنفيذ أي تعبير حسابي مكتوب بصيغة (infix) فإن العمليات تنفذ من اليسار إلى اليمين وحسب أعلى أسبقية للعمليات الحسابية وهي :

الأسبقية	نوع العملية الحسابية
4	^ (power), Unary (-), Unary (+), Not
3	*, /, AND, DIV, MOD
2	+, -, OR
1	=, <, >, <=, >=

وتستخدم الأقواس عند الحاجة إلى تغيير أسبقيات التنفيذ وتضمن الخطوات .

أن البرامج التي تتضمن تعابير حسابية بصيغة (Infix) يقوم المترجم (compiler) بتحويلها إلى صيغة (postfix) باستخدام المكس وبق الخوارزمية الآتية :

Main { main program }

```

100 CALL X
102 —
200 CALL Y
202 —
— 400 CALL P
— 402 —
— 600 CALL R
— 602 —
— 700 CALL S
— 702 —
— 500 CALL Q
— 502 —
300 CALL Z
302 —

```

2- استخدام المكس في معالجة التعابير الحسابية

Arithmetic expressions

من المعروف أن التعابير الحسابية تكتب بثلاث صيغ هي :

1- صيغة Infix notation

تكون إشارة العملية الحسابية تتوسط العوامل مثل :
X/20, A - B, 3 + 4

2- صيغة Prefix Notation

لا تكون إشارة العملية الحسابية تسبق العوامل مثل :
X 20, - A B, + 3 4

مثال: حول العبارة الحسابية التالية من صيغة (fix) إلى صيغة (postfix) باستخدام مكسبين .
 $a - b * (c + d) / (e - f) \wedge g * h$

الحل:

رقم الخطوة الرمز المدخل المكسب الأول ST1 المكسب الثاني ST2

1	A	a	ab	ab
2	-	a	ab	ab
3	B	ab	ab	ab
4	*	ab	ab	ab
5	(	ab	ab	ab
6	C	ab	ab	ab
7	+	ab	ab	ab
8	D	ab	ab	ab
9	)	ab	ab	ab

تلاحظ هنا عند ورود القوس الأيمن يتم إخراج (نقل) جميع العمليات الحسابية لغاية القوس الأيسر من المكسب (ST2) إلى (ST1) مع إخراج القوس الأيسر ليكمل هو والقوس الأيمن .

10	/	ab	ab	ab
11	(	ab	ab	ab
12	E	ab	ab	ab
13	-	ab	ab	ab
14	F	ab	ab	ab
15	)	ab	ab	ab
16	^	ab	ab	ab
17	g	ab	ab	ab
18	*	ab	ab	ab
19	h	ab	ab	ab
20	-	ab	ab	ab

هنا انتهت جميع المدخلات لذا ينقل الباقي في المكسب

(ST2) إلى المكسب (ST1) بالتتابع ليصبح

abed + * ef-g ^ /h *

خوارزمية تحويل صيغة (infix) إلى (postfix) باستخدام مكسبين

- 1- نستخدم مكسبين ، المكسب الأول (ST1) لحزن المتغيرات (العوامل operands) والمكسب الثاني (ST2) يستخدم لحزن إشارات العمليات الحسابية (Operators).
- 2- نفحص التعبير الحسابي رمزا رمزا من اليسار إلى اليمين .
- 3- عند كل رمز نقوم بما يلي :-

ينشف ما يلي:

إذا كان الرمز:

- + يحزن (push) في المكسب (ST1) .
- + يحزن (push) في المكسب (ST2) .
- + قوس ليسر
- + قوس إيمين

+ إخراج (pop) جميع الرموز من المكسب (ST1) وخزنها (push) بالتتابع في المكسب (ST2) لغاية الوصول إلى القوس الأيسر الذي يجب إخراجه وإمهاله مع القوس الأيمن .

+ عملية حسابية (operator) + إخراج (pop) جميع العمليات الحسابية (أن وجدت) في المكسب (ST2) التي أسبقها أعلى أو تساوي أسبقية العملية الحسابية الحالية وخزنها في المكسب (ST1) (التوقف عن ذلك عند عدم تحقق الشرط) ومن ثم خزن العملية الجديدة في المكسب (ST2).

- 4- عند انتهاء كل رموز التعبير الحسابي يتم إخراج (pop) جميع الرموز المتبقية في المكسب (ST2) بالتتابع وخزنها (push) في المكسب (ST1) الذي يحتوي الصيغة النهائية (postfix).

تعريف: حول العبارة الحسابية التالية من صيغة (infix) إلى صيغة (postfix) باستخدام مكس واحد.

الحل:

رقم الخطوة	الرمز المدخل	المكس ST	Output string
1	y	...	y
2	*	*	y *
3	m	*	y * m
4	+	+	y * m +
5	(	(	y * m + (
6	a	(	y * m + (a
7	^	(^	y * m + (a ^
8	3	(^	y * m + (a ^ 3
9	/	(/	y * m + (a ^ 3 /
10	b	(/	y * m + (a ^ 3 / b
11	-	(-	y * m + (a ^ 3 / b -
12	n	(-	y * m + (a ^ 3 / b - n
13	)	)	y * m + (a ^ 3 / b - n +
14	-	-	y * m + (a ^ 3 / b - n + -
15	d	-	y * m + (a ^ 3 / b - n + - d
16	...	...	y * m + (a ^ 3 / b - n + - d -

خوارزمية تحويل صيغة (Infix) إلى (postfix) باستخدام مكس واحد

- 1- نستخدم مكس واحد (ST) لتخزين إشارات العمليات الحسابية (operators).
- 2- نفحص (تقرأ) التعبير الحسابي رمزاً رمزاً من اليسار إلى اليمين
- 3- عند كل رمز نقوم بما يلي :-

ينفذ ما يلي :

إذا كان الرمز :

+ أحد العوامل (operand) + يتقل إلى جملة المخرجات output string

+ قوس إيسر . + يخزن (push) في المكس (ST)

+ عملية حسابية (operator) + إخراج (pop) جميع العمليات الحسابية (أن وجدت) في المكس (ST) التي أسبقها أعلى أو تساوي أسبقية العملية الحسابية الجديدة

وإضافتها إلى جملة المخرجات (التوقف عن ذلك عند عدم تحقق الشرط)

+ بعد ذلك تخزن (push) إشارة العملية الحسابية الجديدة في المكس (ST).

+ إخراج (pop) جميع إشارات العمليات الحسابية من المكس وإضافتها بالتتابع إلى

جملة المخرجات لغاية الوصول إلى القوس الإيسر في المكس الذي يجب إخرجه وإهماله مع القوس الأيمن المقابل له.

مع القوس الأيمن المقابل له.

- 4- عند انتهاء فحص (الممرور على) جميع رموز التعبير الحسابي يتم إخراج (pop) جميع الرموز المتبقية في المكس (ST) بالتتابع وإضافتها إلى جملة المخرجات ليصبح الشكل النهائي لجملة المخرجات هو صيغة ال (postfix) المطلوبة.

7+8-6*3/2

مثال: نأخذ العبارة الحسابية المكثمة بصيغة infix ()

78+63*2/-

عند تحويلها إلى صيغة postfix تصبح

ولا احتساب قيمة هذه العبارة بصيغتها الأخيرة نطبق خطوات الخوارزمية كالآتي:

محتويات المكس ST

المدخلات

رقم الخطوة

7

7

1

7 8

8

2

15

+

3

لاحظ هنا تنفيذ عملية الجمع (+) على العاملين

الموجودين في المكس ومسا (7) ، (8) وخزن

(push) نتيجة الجمع (15) بـلها في المكس

15 6

6

4

15 6 3

3

5

15 18

.

6

لاحظ هنا تنفيذ عملية الضرب (*) على

العاملين (6) ، (3) وخزن النتيجة (18)

بـلها في المكس

15 18 2

2

7

15 9

/

8

لاحظ هنا تنفيذ عملية القسمة (/) على

العاملين (18) ، (2) وخزن النتيجة (9)

بـلها في المكس

6

-

9

لاحظ هنا تنفيذ عملية الطرح (-) على

العاملين (15) ، (9) وخزن النتيجة (6)

بـلها في المكس

أن القيمة المتبقية في المكس وهي (6) تمثل النتيجة النهائية لعملية احتساب قيمة

العبارة الحسابية

احتساب قيمة (تنفيذ) التعبير الحسابي المحول إلى صيغة postfix

بعد أن يحول المترجم العبارة الحسابية من صيغة infix إلى صيغة postfix (postfix)

فإن احتساب قيمتها في المرحلة التالية يكون بموجب الخوارزمية المبينة أدناه باستخدام

مكس واحد .

الخوارزمية

1- يستخدم مكس واحد وليكن (ST).

2- تفحص (تأخذ) التعبير الحسابي رمزا رمزا من اليسار إلى اليمين ويعامل

كالآتي :

ينفذ ما يأتي :

إذا كان الرمز المدخل هو :

+ يخزن (push) في المكس (ST).

+ أحد العوامل (operand)

+ تنفذ هذه العملية على العاملين في

+ عملية حسابية (operator)

أعلى المكس : أي يتم إخراج (pop)

العاملين من المكس (ST) وتنفذ

العملية عليهما وتخزن (push) النتيجة

المتحققة في المكس (ST).

3- عند انتهاء مدخلات التعبير الحسابي فإن القيمة المتبقية في المكس هي النتيجة

النهائية للعبارة الحسابية.

مثال : أوجد قيمة العبارة الحسابية الآتية المكتوبة بصيغة (Infix) باستخدام المكس : $3 + 7 * 2 - 6$

الحل :-

الخطوة	الرمز	ST1	ST2
1	3	3	3
2	+	3	4
3	7	3 7	+
4	*	3 7 7	+
5	2	3 7 2	+
6	-	3 7 2 17	-

لاحظ هنا تنفيذ عملية الضرب (*) على العاملين (7) ، (2) ، والنتيجة هي (14) لأن أسبقيتها < من العملية الجديدة الطرح (-) ثم الاستمرار في تنفيذ عملية الجمع (+) على النتيجة المنقطة (4) والقيمة (3) لنحصل على (17) ولانتهاء العمليات الحسابية التي أسبقيتها < أسبقية العملية الجديدة نخزن إشارة هذه العملية (-) في المكس (ST2) 6 17 عند انتهاء جميع رموز العبارة الحسابية المدخلة نبدأ بتنفيذ العمليات الحسابية المتبقية في المكس (ST2) بالتتابع على محتويات المكس (ST1) وتصبح الخطوة الأخيرة 11 8

فالقيمة (11) المتبقية في المكس (ST1) هي نتيجة الاختساب

خوارزمية احتساب قيمة (تنفيذ) العبارة الحسابية Infix
من التطبيقات الأخرى للمكس استخدامه في المقدرات (Interpreters) لاحتساب قيمة العبارة الحسابية المكتوبة بصيغة (Infix) بدون تحويلها إلى صيغة (postfix).

خطوات الخوارزمية

- 1- يستخدم مكسان هما (ST1) لخزن العوامل الحسابية (operands) و (ST2) لخزن إشارات العمليات الحسابية (operators) :
- 2- تؤخذ رموز العبارة الحسابية بالتتابع واحدا بعد الآخر من اليسار إلى اليمين.
- 3- حسب نوع الرمز نقوم بما يلي :
إذا كان الرمز :
+ يخزن (push) في المكس (ST1) .
+ عملية حسابية (operator) :
+ إخراج (pop) بالتتابع جميع العمليات الحسابية (أن وجدت) في المكس (ST2) التي أسبقيتها < أسبقية العملية الحسابية الجديدة وتنفيذ كل منها على العاملين في قمة المكس (ST1) وتخزن (push) للنتيجة بدلها في (ST1) .
بعد ذلك نخزن (push) إشارة العملية الحسابية الجديدة في المكس (ST2) .
- 4- بعد انتهاء جميع رموز العبارة الحسابية نبدأ بتنفيذ جميع العمليات الحسابية المتبقية في المكس (ST2) بالتتابع على كل عاملين في قمة المكس (ST1) وحل نتيجة تلك العملية محلها في نفس المكس (ST1) ونستمر بتكرار هذه الخطوة لحين خلو المكس (ST2) وتكون آخر قيمة موجودة في المكس (ST1) هي النتيجة النهائية.

مثال :-
حول التعبير الحسابي التالي من صيغة (Infix) إلى صيغة (Postfix) باستخدام مكسبين :

$$(A > B) \text{ AND } ((E - C > A) \text{ OR } (G < F))$$

الحل :-

Step No.	Input char	ST1 For operands	ST2 For operators
1	(		(
2	A	A	(
3	>	A	(>
4	B	AB	(>
5	)	AB>	
6	AND	AB>	AND
7	(	AB>	AND (
8	(	AB>	AND ((
9	E	AB>E	AND ((
10	-	AB>E	AND ((-
11	C	AB>EC	AND ((-
12	>	AB>EC-	AND ((->
13	A	AB>EC-A	AND ((->
14	)	AB>EC-A>	AND (
15	OR	AB>EC-A>	AND (OR
16	(	AB>EC-A>	AND (OR (
17	G	AB>EC-A>G	AND (OR (
18	<	AB>EC-A>G	AND (OR (<
19	F	AB>EC-A>GF	AND (OR (<
20	)	AB>EC-A>GF<	AND (OR (<
21	)	AB>EC-A>GF<OR	AND (OR
22		AB>EC-A>GF<OR AND	

مثال :-
حول التعبير الحسابي التالي من صيغة (Infix) إلى صيغة (Postfix) باستخدام مكسبين
 $M := X / 6 + (a - 2 * (b / 3) ^ 5 + f) ^ 2$

الحل :-

Step No.	Input char	ST1 For operands	ST2 For operators
1	M	M	
2	:=	M	:=
3	X	MX	:=
4	/	MX	:= /
5	6	MX6	:= /
6	+	MX6 /	:= +
7	(	MX6 /	:= + (
8	A	MX6 / a	:= + (
9	-	MX6 / a	:= + (-
10	2	MX6 / a2	:= + (-
11	*	MX6 / a2	:= + (-*
12	(	MX6 / a2	:= + (-*(
13	B	MX6 / a2 b	:= + (-*(
14	/	MX6 / a2 b	:= + (-*(/
15	3	MX6 / a2 b3	:= + (-*(/
16	)	MX6 / a2 b3 /	:= + (-*
17	^	MX6 / a2 b3 /	:= + (-**
18	5	MX6 / a2 b3 /	:= + (-**
19	+	MX6 / a2 b3 / 5 ^ *	:= + (-** +
20	F	MX6 / a2 b3 / 5 ^ *-f	:= + (-** +
21	)	MX6 / a2 b3 / 5 ^ *-f +	:= +
22	^	MX6 / a2 b3 / 5 ^ *-f +	:= + ^
23	2	MX6 / a2 b3 / 5 ^ *-f + 2	:= + ^
24		MX6 / a2 b3 / 5 ^ *-f + 2 ^ + :=	

3- تطبيق آخر

يستخدم المكس كهيكل ل تخزين المعلومات التي تحتاج استرجاعها بصورة معكوسة (يترتيب معكوس) والحالات التي تتطلب العودة إلى موقع الخطوة السابقة (back tracking) وكمثال على ذلك مسائل المتاهة (A mazing problems) فعند المرور بموقع معين وتكون هنالك عدة مسارات وتقرض اختيار أحدها للوصول إلى الهدف فإن الأمر يتطلب تخزين هذا الموقع قبل تركه وتجربة مسار آخر إذ نحتاج إلى العودة لهذا الموقع في حالة خطأ ذلك المسار.

إن استخدام المكس في مثل هذه الحالات يسمح ب تخزين سلسلة المواقع السابقة بحيث يمكن العودة إليها بعكس ترتيب المرور فيها .

تبرين محلول:
اكتب خوارزمية لقراءة جملة (string) تنتهي بالرمز (.) ثم طبعها بترتيب معكوس باستخدام المكس .

الحل:

```
Algorithm
Begin
clear the stack
Do
{ Read a character
  If character <> '.'
  Then push the character onto stack
}
While ( character == '.' )
While stack is not empty
Do
{ Pop the stack
  Print the character
}
End
```

مثال:

حول التعبير الحسابي التالي من صيغة (Infix) إلى صيغة (Postfix) باستخدام مكس:

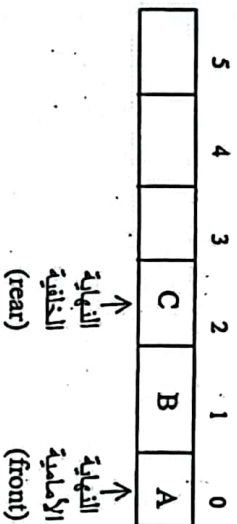
A Not (B OR Z OR Not(G < E))

الحل:

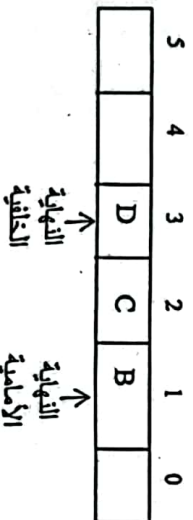
Step No.	Input char	ST1 For operands	ST2 For operators
1	A	A	
2	Not	A	Not
3	(	A	Not (
4	B	AB	Not (
5	OR	AB	Not (OR
6	Z	ABZ	Not (OR
7	OR	ABZ OR	Not (OR
8	Not	ABZ OR	Not (OR Not
9	(	ABZ OR	Not (OR Not (
10	G	ABZ OR G	Not (OR Not (
11	<	ABZ OR G	Not (OR Not (<
12	E	ABZ OR GE	Not (OR Not (<
13	)	ABZ OR GE <	Not (OR Not
14	)	ABZ OR GE < Not OR	Not
15		ABZ OR GE < Not OR Not	

3-3 الطابور Queue

هو هيكل تسلسلي (sequential) تكون فيه عمليات الإضافة في النهاية الخلفية (rear) وعمليات الحذف في النهاية الأخرى (الأمامية front) كما في الشكل التالي :



حيث نلاحظ أن العنصر (A) في مقدمة الطابور يليه العنصر (B) ثم (C). وعند إضافة عنصر جديد يكون موقعه بند (C)، أما عند حذف عنصر من الطابور تكون عملية الحذف من النهاية الأمامية أي حذف العنصر (A) ويصبح الشكل أعلاه بعد إضافة العنصر (D) وحذف العنصر (A) كالآتي :-



نجد أن الطابور يفيد في العمليات التي تتضمن جولة الوصول حسب ترتيب وصولها أو طلبها ويمكن تلخيص هذا بالعبارة الآتية :

First IN First Out [FIFO]

أول من يدخل أول من يخرج

أي أن من يصل أولاً يحصل على الخدمة أولاً وتسمى عملية الإضافة إلى الطابور (ENQueue) أو (Insertion) أما عملية الحذف فتسمى (DEQueue) أو (Deletion).

تمرين : اعتمد خوارزمية التمرين السابق وكتب الأيعازات اللازمة لقراءة آية جملة (string) تنتهي بالرمز (.) ، ثم اطبع تلك الجملة بترتيب معكوس وذلك باستخدام المكس .

Print Reverse

```
{
    const char dot = '.';
    char ch;
    int top = -1;
    do
    {
        cin >> ch;
        if (ch != dot)
            Push(ch);
    } while (ch != dot)

    while (top != -1)
    {
        pop();
        cout << item;
    }
}
```

ويعرف الطابور برمجيا بلغة C++ باستخدام العبارات البرمجية التالية :-

```
const int size = 10 ; { or any other value }
char item ;
{ or any other type }
char queue[size] ;
int rear, front ;
```

عملية الإضافة للطابور

تعتمد الخطوات الآتية لإضافة عنصر واحد إلى الطابور :

- 1- التحقق بأن الطابور غير مملوء (Not Full) أي أن المؤشر (rear < size-1) لتجنب حالة النض (Overflow) ولتفرض تنفيذ عملية الإضافة عند ذلك .
- 2- تحديث قيمة المؤشر (rear = rear + 1) ليشير إلى الموقع التالي
- 3- إضافة العنصر الجديد في الموقع الجديد queue [rear]

عملية الحذف من الطابور

تعتمد الخطوات الآتية لحذف عنصر واحد من الطابور :

- 1- التحقق بأن الطابور غير خال (Not Empty) أي أن المؤشر (front > -1) لتجنب حالة النض (Underflow) وتعذر تنفيذ عملية الحذف .
- 2- اخذ العنصر من الموقع الذي يشير إليه المؤشر (front) وخرزته وقتيا في متغير مستقل وليكن item = queue [front]
- 3- تحديث قيمة المؤشر (front = front + 1) ليشير إلى موقع العنصر الآتي للعنصر الذي حذف .

ملاحظة :

هذا أيضا يتضح أن الخطوات (2 ، 3) في عملية الحذف معكوسة للترتيب عنها في عملية الإضافة .

3-3-1 تمثيل الطابور باستخدام المصفوفة

يمكن تمثيل الطابور حاسوبيا باستخدام مصفوفة أحادية الباعثة المطوية (size) وبالترتيب المناسب لترتيب البيانات (Data Type) التي ستخزن فيه (int, float... الخ) مع استخدام ما يلي :

المتغير (rear) كمؤشر يشير إلى موقع العنصر الأخير في الطابور
المتغير (front) كمؤشر يشير إلى موقع العنصر الأول في الطابور
أن قيمة المؤشرين في الحالة الابتدائية (front = -1, rear = -1) عندما يكون الطابور خاليا (Empty) من العناصر

تنفذ عملية إضافة عنصر إلى الطابور بعد تحديث قيمة المؤشر (rear) ليشير إلى الموقع الجديد بعد موقع آخر عنصر أما تنفيذ عملية حذف عنصر من الطابور فيحدث المؤشر (front) ليشير إلى موقع العنصر التالي بعد حذف العنصر في الموقع الذي كان يشير إليه المؤشر (front) ، ولتفرض لدينا الطابور (Q) سعته (6) عناصر وتنفذ عليه سلسلة العمليات الآتية :-

حالة الطابور

الحالة	المؤشر F	المؤشر R	Q[0]	Q[1]	Q[2]	Q[3]	Q[4]	Q[5]
- الطابور خالي	-1	-1	-	-	-	-	-	-
- إضافة العنصر A	0	0	A	-	-	-	-	-
- إضافة العنصر B	0	1	A	B	-	-	-	-
- إضافة العنصر C	0	2	A	B	C	-	-	-
- حذف عنصر	1	2	-	B	C	-	-	-
- إضافة العنصر D	1	3	-	B	C	D	-	-
- إضافة العنصر E	1	4	-	B	C	D	E	-
- حذف عنصر	2	4	-	-	C	D	E	-
- حذف عنصر	3	4	-	-	-	D	E	-

5- خوارزمية إخلاء (تفريغ) الطابور (Clear Queue)

هذه الخوارزمية تستخدم لتفريغ الطابور وإخلائه من العناصر لجعل قيمة كل من المؤشرين (front = -1, rear = -1).

```
front ← -1
rear ← -1
```

3-3-3 البرامج الفرعية لتنفيذ عمليات الطابور

Queue's Subprograms

أن الطابور هو هيكل بياني، خصائصه (attributes) هي: حجمه (size)، ونوع البيانات المخزنة فيه (data type)، وأهم عملياته (actions) هي: الإضافة (Enqueue)، والحذف (Dequeue)، ويمكن تعريفه ببرمجة بلغة C++ بما يسمى (class)، وستكون مخرجاتها كالآتي:

1- وجود التعريفات البرمجية التالية

```
const int size = 10; { or any other value }
char item; { or any other type }
char queue[size];
int rear, front;
```

2- برنامج فرعي (function) لإخلاء الطابور

```
void initialize()
{
    rear = -1;
    front = -1;
}
```

لاحظ عدم الحاجة للدور على جميع مواقع الصفوف والإكثاء فقط لجعل قيمة كل من المؤشرين مساوية (-1)، وهذا البرنامج الفرعي (function) يُستدعى في بداية البرنامج لجعل الطابور خالياً.

2-3-3 خوارزميات الطابور Queue's Algorithms

1- خوارزمية الإضافة Add Queue

```
If Queue is Full
Then Overflow ← True
Else
    Overflow ← False
    Rear ← Rear + 1
    Queue[Rear] ← New element
```

2- خوارزمية الحذف Delete Queue

```
If Queue is Empty
Then Underflow ← True
Else
    Underflow ← False
    Element ← Queue[Front]
    Front ← Front + 1
```

3- خوارزمية ملء الطابور Full Queue

هذه الخوارزمية للتحقق من الطابور أن كان مملوء أم لا اعتماداً على قيمة المؤشر (Rear) قبل عمليات الإضافة.

```
If Rear = (size - 1)
Then FullQueue ← True
Else FullQueue ← False
```

4- خوارزمية خلو الطابور Empty Queue

هذه الخوارزمية للتحقق من الطابور أن كان خالياً أم لا اعتماداً على قيمة المؤشر (Front) قبل عمليات الحذف.

```
If Front = -1
Then EmptyQueue ← True
Else EmptyQueue ← False
```

4- حذف عنصر واحد من الطابور (Deque)

```
void dequeue (char item) ;
{
    if (front == -1)
        cout << "The queue is Empty" ;
    else
    {
        item = queue[front] ;
        if (front == rear)
        {
            front = -1 ;
            rear = -1 ;
        }
        else
            front = front + 1 ;
    }
}
```

هذه الخطوة لمعالجة حذف آخر عنصر في الطابور عندما يكون كلا المتغيرين front, rear يشيران اليه ولكن نعيد المؤشرين الى البداية

① : إذا كان front = rear = -1
② : إذا كان front = rear = 0
③ : إذا كان front = rear = 1
④ : إذا كان front = rear = 2
⑤ : إذا كان front = rear = 3

ملاحظة : يمكن استدعاء هذا البرنامج الفرعي من قبل البرنامج الرئيسي بأي عدد من المرات بقدر عدد العناصر المطلوب حذفها باستخدام أحد ايمارات التكرار .

3-3-4 تمثيل الطابور باستخدام Structure

يستخدم الـ (structure) في تمثيل الطابور والمؤشرين (front, rear) في هيكل بياني ويتكون من ثلاثة اجزاء ، الجزء الأول يمثل مصفوفة الطابور والجزء الثاني وهو حقل يمثل المؤشر (front) والجزء الثالث هو حقل آخر يمثل المؤشر (rear).
ويكون التعريف حسب لغة C++ كما يلي :-

ويمكن أيضاً "استخدام (constructor) في لغة C++ لتحقيق نفس الهدف باستخدام اسم الـ class بالصيغة التالية :

```
Queue()
{
    rear = -1 ;
    front = -1 ;
}
```

3- إضافة عنصر واحد الى الطابور (Enqueue)

```
void enqueue (char item)
{
    if (rear == size-1)
        cout << "Queue is Full" ;
    else
    {
        rear ++ ;
        Queue[rear] = item ;
    }
}
```

هذه الخطوة لمعالجة أول اجزاء (اول عنصر) الـ الطابور ليصبح المؤشر front يشير اليه

```
if (front == -1)
    front = 0 ;
}
```

ملاحظة : يمكن استدعاء هذا البرنامج الفرعي داخل البرنامج الرئيسي (main) بأي عدد من المرات باستخدام أحد ايمارات التكرار مثل (For - While) بقدر عدد العناصر المطلوب إضافتها .

أما عند حذف عنصر من الطابور فتتبع الخطوات الآتية :

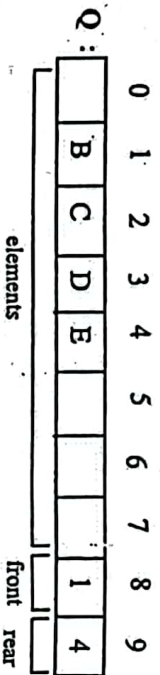
1- نأخذ العنصر من مقدمة الطابور كما يشير إليه المؤشر (front) بالإيعاز :

item = Q.elements[Q.front]

2- تحديث قيمة المؤشر (front) من (0) ليصبح (1) بالإيعاز :

Q.front = Q.front+1

ويصبح الطابور بالصورة الآتية :



تمرين: اعد كتابة البرامج الفرعية لعمليات الطابور باستخدام الـ (structure).

3-5 تطبيقات القائمة للطابور

من التطبيقات الشائعة للطابور في مجال الحاسوب :

* جدولة الأعمال (job scheduling) ، ففي نظام معالجة الدفعات

(batch processing) تنظم الأعمال المطلوب تنفيذها في طابور حسب وقت

وصولها ومن ثم تنفذ بالتتابع واحدا بعد الآخر . . .

* كما تستخدم أنظمة التشغيل الطابور في جدولة استخدام المصادر المختلفة

للحاسوب (Computer Resources) ، فطابور للأعمال التي تحتاج وقت

للإخراج على الطابعة و آخر الاحتمال استخدام القرص، وهكذا باقي الأجهزة.

const int size = 8 ; { or any other value }

char item ;

structure queue

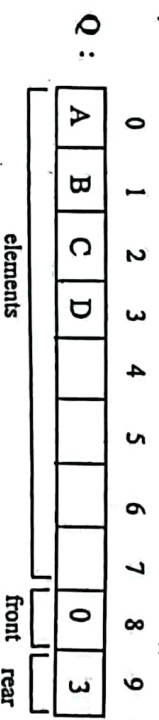
{

char elements[size] ;

int front ;

int rear ;

}



يسمى برمجيا

Q.front

Q.elements

إضافة عنصر جديد لهذا الطابور تتبع الخطوات التالية :-

1- نحدد قيمة المؤشر (rear) الذي هو جزء في الـ (structure) لتصبح

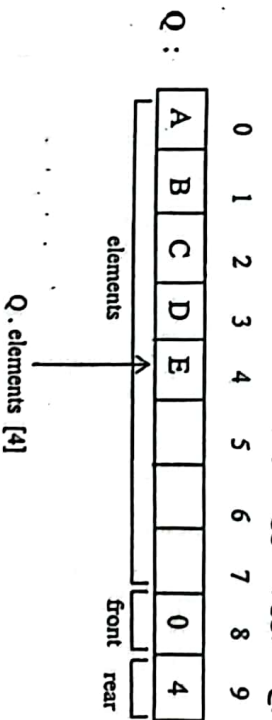
قيمته (4).

Q.rear = Q.rear + 1

2- نضيف العنصر الجديد (E) في الموقع الجديد (4)

Q.elements [Q.rear] = E

وبهذا يصبح الطابور بالصورة التالية :



تمرين: اكتب خوارزمية (Algorithm) لقراءة جملة (string) تتكون من جزئين (two) اكتب بفصلهما الرمز ':' ثم التحقق بكونهما متطابقين أم لا باستخدام الطابور.

Algorithm

Begin

Clear the Queue

Stringmatch $\leftarrow$ True

Repeat

Read a character and put it in the string

If character $\diamond$ ':'

Then EndQ the character

Until character = ':'

While (more elements in the Queue) Do

Dequeue an element

Read the next character and place in string

If Dequeue character $\diamond$ character

Then stringmatch $\leftarrow$ false

End.

تمرين: اكتب الأيعزات البرمجية (برنامج) فرعية (5) لا إضافة عناصر إلى الطابور (Line) الذي سيقته (25) عدد صحيح (integer).

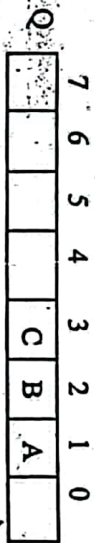
الحل:

```
const int size = 25;
int Line[size];

void Addqueue(int item)
{
    for (int j=1; j<=5; j++)
    {
        if (rear == size)
            cout<<"\n the queue is full ";
        else
        {
            rear = rear + 1;
            cin >> item;
            Line[rear] = item;
        }
        if (front == -1)
            front = 0;
    }
}
```

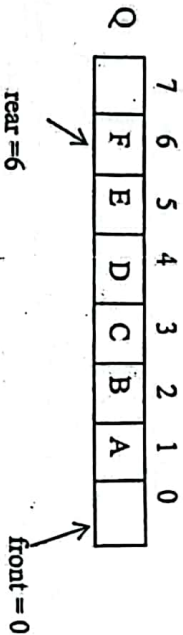
ولتأخذ هذه الأمثلة التوضيحية الآتية لطاير دائري سعته (8) :

الحالة الأولى: يحتوي الطاير ثلاثة عناصر A, B, C.



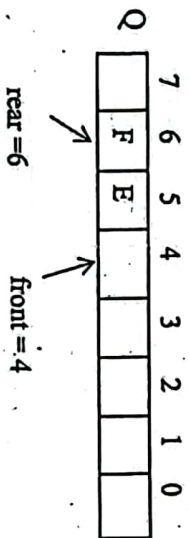
rear = 3, front = 0

الحالة الثانية: بعد إضافة العناصر D, E, F.



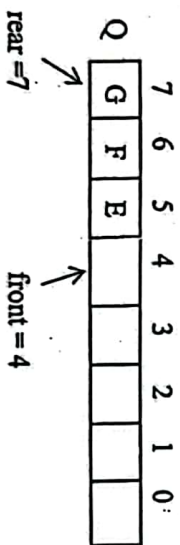
rear = 6, front = 0

الحالة الثالثة: بعد حذف العناصر A, B, C, D.



rear = 6, front = 4

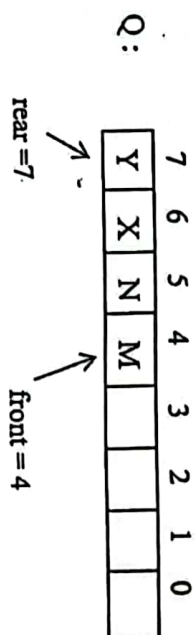
الحالة الرابعة: بعد إضافة العنصر (G)



rear = 7, front = 4

4-3 الطاير الدائري Circular Queue

لاحظنا عند إضافة عنصر إلى الطاير الأصيلي فإنه يتطلب تدقيق (فحص) قيمة المؤشر $(rear = size - 1)$ إذ أن ذلك يعني أن الطاير مملوء حتى وإن كانت هناك مواقع خالية في مقدمة الطاير كما في الشكل الآتي :



rear = 7, front = 4

أي أننا سنخسر المساحة التخزينية في مقدمة الطاير دون استخدام ، ولتجنب هذه الحالة نستطيع استخدام الطاير استخداما دائريا وذلك بأن نسمح للمؤشر (rear) بالدوران إلى النهاية الأمامية للطاير (wrap - around) عندما يكون هناك مواقع خالية فيها.

أن خصائص هذا الطاير هي كالاتي:

- 1- أن المؤشر (front) يشير إلى الموقع الذي أمام أول عنصر في الطاير .
- 2- أن المؤشر (rear) يشير إلى موقع العنصر الأخير في الطاير .
- 3- عندما يصل المؤشر (rear) إلى الموقع الأخير في الطاير أي $(rear = size - 1)$ نجعله يدور إلى البداية أي $(rear = 0)$ وكذلك الحال بالنسبة للمؤشر (front) .
- 4- أن أكبر عدد من العناصر التي يستوعبها هذا الطاير هو $(size - 1)$ لأن المؤشر (front) يشير إلى الموقع الخالي أمام أول عنصر في الطاير .

برنامج فرعي (function) لإضافة عنصر واحد إلى الطابور الدائري

```
void AddCQ(char item) ;
{
    if (rear == size - 1)
        rear = 0;
    else
        rear = rear + 1;

    if (rear == front)
        cout << "\n The CQ is full ";

    else
        CQueue[rear] = item;
}

void DeleteCQ(char item) ;
{
    if (front == rear)
        cout << "\n The CQ is Empty ";
    else
    {
        if (front == size - 1)
            front = 0;
        else
            front = front + 1;
        item = CQueue[front];
    }
}
```

هذا الجزء لتحديث قيمة المؤشر rear

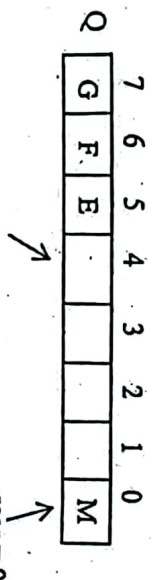
للتحقق بأن rear لن يشير إلى موقع front كما في الحالة السابقة من المثال التوضيحي السابق .

إضافة العنصر item

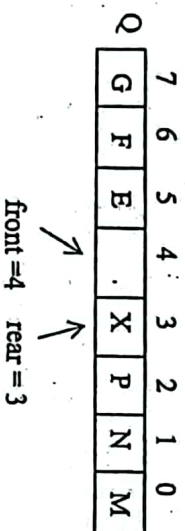
برنامج فرعي (function) لحذف عنصر واحد من الطابور الدائري

لاحظ هنا في الطابور البسيط يعتبر الطابور مملوء لان $(rear = size - 1 = 7)$ مع انه يوجد مواقع خالية في البقعة أما في الطابور الدائري فستطيع إضافة عنصر كما في الحالة التالية .

الحالة الخامسة: لإضافة العنصر (M) لاحظ دوران المؤشر (rear) إلى الجهة الأمامية بعد وصوله إلى الموقع الأخير في الطابور .



الحالة السادسة: بعد إضافة العناصر X, P, N لاحظ هنا أن الطابور أصبح مملوءاً لأننا إذا أردنا إضافة عنصر فوجب تحديث قيمة المؤشر (rear) وستصبح قيمته (4) ، أي أنه سيكون مساوياً لقيمة المؤشر الذي يجب أن يبقى موقفاً خالياً ، ربما أن سعة هذا الطابور هي $(size - 1)$ أي $(8 - 1 = 7)$ وهذا ما يحويه حالياً ، لذا فستعجز لإضافة في هذه الحالة .



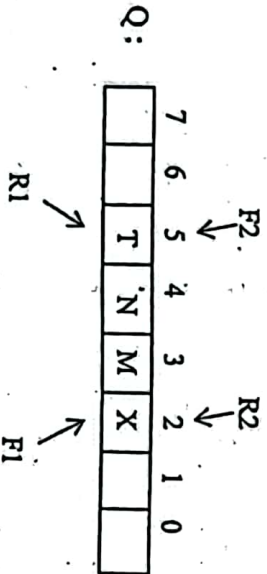
5-3 الطابور الثنائي (المزدوج)

Double Ended Queue (Deque)

هيك تسلسلي (Linear Structure) يمكن إضافة أو حذف العنصر من أي من طرفيه ويمثل بصفوفة أحادية مع أربعة مؤشرات هي:

- F1 يشير إلى موقع أول عنصر في الطابور عند استخدامه من الجهة اليمنى.
- R1 يشير إلى موقع آخر عنصر في الطابور عند استخدامه من الجهة اليمنى.
- F2 يشير إلى موقع أول عنصر في الطابور عند استخدامه من الجهة اليسرى.
- R2 يشير إلى موقع آخر عنصر في الطابور عند استخدامه من الجهة اليسرى.

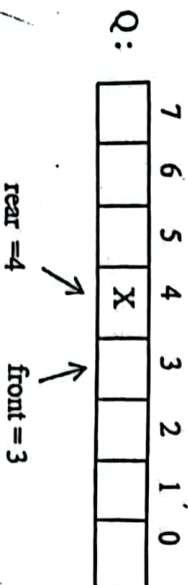
كما في الشكل الآتي :



إذ يمكن حذف العنصر X باستخدام المؤشر (F1) وإضافة عنصر بعد العنصر (T) باستخدام المؤشر (R1)، كما يمكن حذف العنصر (T) باستخدام المؤشر (F2) وإضافة عنصر بعد العنصر (X) باستخدام المؤشر (R2).

توضيح:

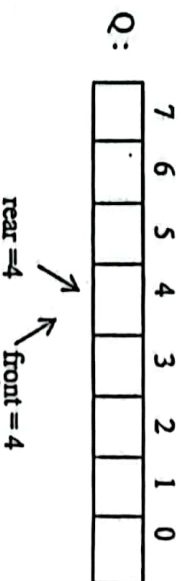
لناخذ طابورا دائريا سعة (8) عناصر يحتوي حاليا على عنصر واحد هو (X) كما في الرسم التوضيحي أدناه حيث قيمة المؤشر (front = 3) وقيمة المؤشر (rear = 4)



عند حذف عنصر واحد ماذا سيحصل :

- نُحدث قيمة المؤشر (front = front + 1) أي يصبح (front = 4).
- نأخذ العنصر من موقعه item = CQ [front]

- يصبح شكل الطابور الدائري كالآتي:



حيث أن (rear=front=4)، والطابور خال من العناصر لذا فإن أي عملية حذف لاحقة كما في البرنامج الفرعي DeleteCQ سيتعذر تنفيذها.

ملاحظة مهمة:

نجد أن الشرط نفسه (rear = front) يستخدم في البرنامج الفرعي (AddCQ) ليعني (أن الطابور مملوء) وفي البرنامج الفرعي (DeleteCQ) ليعني (أن الطابور خال) وهذا لا يعني التناقض لأن تسلسل الخطوات يختلف إذ يورد عند الإضافة بعد تحديث قيمة المؤشر (rear) ويورد عند الحذف في البداية.

```

st.push('A'); cout<<" A is pushed\n";
st.push('B'); cout<<" B is pushed\n";
st.push('C'); cout<<" C is pushed\n";

cout<<st.pop()<<" is popped\n";
cout<<st.pop()<<" is popped\n";
cout<<st.pop()<<" is popped\n";
return 0;
}

```

تمثيل المكس لتنفيذ اي عدد من عمليات الاضافة و الحذف ،
وطبع (عرض) محتويات المكس بعد كل عملية .

```

#include <iostream>
using namespace std;
const int size=10;
class stack
{
private:
char arr[size];
public:
int top; int allowed; // constructor to set the top
{
top=-1;
}
bool stackfull(int top)
{
if(top==size-1)
return 1;
else return 0;
}
bool stackempty(int top)
{
if(top==-1)
return 1;
else return 0;
}
void push(char item)
{
if ( stackfull(top))
cout<<"\n the stack is FULL ";
else
{
arr[++top]=item;
}
}
char pop()
{
if ( stackempty(top))
cout<<" Stack is Empty "<<endl;
else
{
char item2=arr[top];
}
}
}

```

87

برنامج رقم 2 :-

برامج رقم 1 :- تمثيل المكس لتنفيذ عمليات الاضافة (3) عناصر بالتتابع ثم حذفها تلقائياً

```

#include <iostream>
using namespace std;
const int size=10;
class stack
{
private:
char arr[size];
int top;
public:
void clear() // initializer function to set the top
{
top=-1;
}
void push(char item) // Adding an item to the stack
{
top++;
if (top<size)
{
arr[top]=item;
}
else
{
cout<<" Stack is Full " << endl;
top--;
}
}
char pop() // Deleting an item from the stack
{
if (top==1)
{
cout<<" Stack is Empty " << endl;
return 0;
}
else
{
char item2=arr[top];
arr[top]='';
top--;
return item2;
}
}
};

int main()
{
cout<<"This Program will add (Push) A,B,C in the STACK and POP it up \n";
stack s;
s.clear();
}

```

86

```

    void push(char item) // Adding an item to the stack
    {
        st.top++;
        if (st.top < size)
            { st.arr[st.top] = item;
            }
        else
            { cout << "Stack is Full" << endl;
              st.top--;
            }
        char popl // Deleting an item from the stack
        {
            if (st.top == -1)
            {
                cout << "Stack is Empty" << endl;
                return 0;
            }
            else
            {
                char item2 = st.arr[st.top];
                st.arr[st.top] = '\0';
                st.top--;
                return item2;
            }
        }
    }

int main()
{
    stack st;
    st.clear();
    st.push('A'); cout << "A is pushed\n";
    st.push('B'); cout << "B is pushed\n";
    st.push('C'); cout << "C is pushed\n";
    cout << st.pop() << " is popped\n";
    cout << st.pop() << " is popped\n";
    cout << st.pop() << " is popped\n";
    return 0;
}

```

89

```

arr[top] = '\0';
top--;
return item2;
}

void display()
{
    for(int i=top; i>=-1; i--)
    {
        cout << arr[i] << " ";
    }
}

int main()
{
    stack st; char c; int m;
    cout << "\n how many elements ( <= 10 of char type ) you like to push ? "; cin >> m;
    for(int i=1; i<=m; i++)
    {
        cout << "enter the data item : "; cin >> c;
        st.push(c);
    }
    cout << "\n the content of the stack after addition is : ";
    st.display();
    cout << "\n how many elements you like to delete ? "; cin >> m;
    for( int i=1; i<=m; i++)
    {
        st.pop();
        cout << "\n the content of the stack after deletion is : ";
        st.display();
    }
    return 0;
}

```

برنامج رقم -3: تمثيل المكس (باستخدام struct) لتنفيذ عمليات اضافة
(3) عناصر بالتتابع ثم حذفها بالتتابع ، أي انه مثله للبرامج الأول ولكن
يستخدم هنا (struct) .

```

#include <iostream>
using namespace std;
const int size=20;
struct stack
{
    char arr[size];
    int top;
}

class stack
{
private:
    stack st;
public:
    void clear() // initializer to set the top
    {
        st.top=-1;
    }
}

```

88

برنامـج رقم 5: استخدام المـكس لقراءة جملة (string) تنتهي بالرمز (.) ، ومن ثم طباعة الجملة بصورة معكوسة.

طبيع (عرض) محتويات المكس باستخدام قائمة اختيارات (menu) .

برنامج رقم 6: تمثيل الطابور (Queue) وعمليات الأضائة و الحذف وطبع (عرض) محتوياته

```
#include <iostream>
using namespace std;
const int size=10;
int top; char item2;
class stack
{
private:
char arr[size];
public:
stack() // constructor to set the top
{
top=-1;
}
bool stackfull(int top)
{
if(top==size-1)
return 1;
else return 0;
}
bool stackempty(int top)
{
if(top==-1)
return 1;
else return 0;
}
void push(char item)
{
if ( stackfull(top))
cout<<"\n the stacke is FULL ";
else
arr[++top]=item;
}
char pop()
{
if ( stackempty(top))
cout<<" Stack Is Empty " <<endl;
else
{
item2=arr[top];
arr[top]='';
top--;
return item2;
}
}
void display()
{
for(int i=top; i>=-1; i--)
cout<<arr[i];
}
```

93

```
bool stackempty(int top)
{
if(top==-1)
return 1;
else return 0;
}
void push(char item)
{
if ( stackfull(top))
cout<<"\n the stacke is FULL ";
else
arr[++top]=item;
}
char pop()
{
if ( stackempty(top))
cout<<" Stack Is Empty " <<endl;
else
{
item2=arr[top];
arr[top]='';
top--;
return item2;
}
}
void display()
{
for(int i=top; i>=-1; i--)
cout<<arr[i];
}
int main()
{
char ch; const char dot=".";
stack s;
cout<<"Input the string's characters one by one ended by (dot) :\n";
do
{
ch>>ch;
if ( ch!=dot )
s.push(ch);
} while (ch!=dot);
cout<<"\n the reversed string is : ";
while ( top>=-1 )
{
s.pop();
cout<<item2;
}
return 0;
}
```

92

```

    {
        if(front==rear)
            cout<<" The CQueue is Empty ";
        else
        {
            if(front==size-1)
                front=0;
            else
                front=front+1;
            item=Cqul(front);
            Cqul(front)=0;
        }
        return item;
    }

    void displaycq()
    {
        cout<<" CQueue content is : ";
        for(int j=0; j<size-1; j++)
            cout<<Cqul[j]<<" ";
        cout<<"\n front="<<front<<" rear="<<rear;
    }
};

int main()
{
    CQueue cq1;
    CQueue cq2;
    int item2,m,choice; int count=0;
    do
    {
        cout<<"\n Representation of the CQUEUE & its operations ";
        cout<<"\n _____ ";
        cout<<"\n 1 - Add ( ENQUEUE ) elements( Integer values ) to the queue ";
        cout<<"\n 2 - Delete ( DEQUEUE ) elements from the queue ";
        cout<<"\n 3 - Display the content of the queue [ 0 It means empty space ] ";
        cout<<"\n 4 - Exit ";
        cout<<"\n ";
        cout<<endl;
        cout<<"\n select your choice : ";
        cin>>choice;
        switch(choice)
        {
            case 1:cout<<"\n how many elements you like to Add ? ";cin>>m;
                if (m>size-1-count)
                    cout<<"\n the available space(s) are="<<(size-1-count)<<endl;
                else
                {
                    for(int j=1; j<=m; j++)
                    {
                        cout<<" enter the data Item : "; cin>>item2;
                        cq1.Enqueue(item2);

```

95

```

    }
}

int main()
{
    char ch; const char dot=".";
    stack<> stack;
    stack s1;

    cout<<"input the string's characters one by one ended by (dot) :\n";
    do
    {
        cin>>ch;
        if (ch!=dot)
            s1.push(ch);
        while (ch!=dot)
            cout<<"\n the reversed string is : ";
        while (top!=1)
        {
            s1.pop();
            cout<<item2;
        }
        return 0;
    }
}

#include <iostream>
using namespace std;
const int size=10;
class CQueue
{
public:
    int rear; int front; int item;
    int Cqul(size);
    CQueue()
    {
        rear=0; front=0;
        for (int i=0; i<size-1; i++) // To clean up the content of Cqu
            Cqul[i]=0;
    }
    void Enqueue(int item)
    {
        if (rear==size-1)
            rear=0;
        else
            rear=rear+1;
        if (rear==front)
            cout<<" The CQueue is Full ";
        else
            Cqul[rear]=item;
    }
    int Dequeue()

```

تمثيل الطابور (Circular Queue) وعمليات الإضافه
 والحذف وطبع (عرض) محترقته.

برنامج رقم -7:

94


```

        return 1;
    else return 0;
}

bool stackempty(int top)
{
    if(top==1)
        return 1;
    else return 0;
}

void push(char item)
{
    if (stackfull(top))
        cout<<"\n the stack is FULL ";
    else
    {
        arr[++top]=item;
    }
}

char pop()
{
    if (stackempty(top))
        cout<<" Stack is Empty "<<endl;
    else
    {
        item2=arr[top];
        arr[top]='';
        top--;
        return item2;
    }
}

void display()
{
    for(int i=top; i>1; i--)
    {
        cout<<arr[i]<<endl;
    }
}

}

int main()
{
    char ch1, ch2; const char dot=".";
    int count=0; bool palindrome=true;
    int top1=1; int top2=1; int top=1;
    stack();
    stack s1,s2;
    cout<<"Input the string characters one by one ended by (dot) : ";
    do
    {
        ch1>>ch1;

```

97

```

        ++count;
    }
}

break;
case 2: cout<<"\n how many elements you like to delete ?>>nm;
    if (nm>count)
        cout<<"\n No. of item(s) in the queue is "<<(count)<<endl;
    else
    {
        for( int i=1; i<=nm; i++)
        {
            cq1.Dequeue();
            --count;
        }
    }
}

break;
case 3: if (count==0)
    cout<<"\n The Queue is Empty ";
else
    cq1.display();
}

break;
// end of switch
} // end of DO
while(choice!=4);
return 0;
}

```

برنامج رقم - 8 : لقراءة جملة (string) وفحصها ان كانت مقروءة من
اليمين الى اليسار ، اي ان نوحها (Palindrom) وذلك باستخدام مكسبين .

```

#include <iostream>
using namespace std;
const int size=10;
char item2;
char arr1[size], arr2[size]; int top=1;
class stack
{
private:
    char arr1[size];
public:
    int top; char ch;
    stack() // constructor to set the top
    {
        top=1;
    }
    bool stackfull(int top)
    {
        if(top==size-1)

```

96

أمثلة الفصل

1. Let S be a stack of size (15) characters, write a C++ segment of code to add (push) one element.
2. Write a procedure to (pop) three elements from the stack (TABLE) of size (20) integers.
3. Let S for stacking an element in the stack, and U for unstacking. If the order of the stack input stream is 1 2 3 4 5.
 - a - what is the output if we execute the following operations SSUUSSUUU
 - b - which of the following permutations can be obtained as output stream (explain the reason for each case).
 - i- 5 1 3 2 4
 - ii- 2 3 5 1 4
 - iii- 3 2 1 5 4
4. If the stack input stream is A B C D E F what is the sequence of operations to get the output C B D E F A ?
5. Write a function to check if the stack (S) is empty.
6. Explain how we can use the structure for stack representation ?
7. State the main applications of the stack?
8. Explain how we can use stack to achieve the processing of Subprograms calls in computer programming ?

```

s1.push(ch1);
count=count+1;
} while (ch1!=dot);
count=count-1; s1.top--;
for (int i=1; i<=(count/2); i++)
{
    s1.pop();
    s2.push(i*len/2);
}
if ((count%2)==1)
    s1.pop();
while (!s1.empty() && s1.top() != 1 && palindrome==1)
{
    if (s1.pop() != s2.pop())
        palindrome=false;
}
if (palindrome == true)
    cout<<"\n the string is PALINDROM ";
else
    cout<<"\n the string is NOT PALINDROM ";
}
    
```

Explain in details how we can use the stack in processing of such program.

10. Convert the following infix expressions into postfix notations using two stacks:

- $a + b \wedge 2/4 - (c * 5/8 - d) \wedge 3$
- $m \wedge 3 \text{ OR } n - b/2 \text{ AND } (m + n)$
- $a + b + c * (-d/3 \wedge 4) \text{ AND Not } F$
- $a \wedge (b/2) \text{ OR } (x + y/3 - w) \text{ AND } (c \wedge 2) \wedge 3$
- $x \wedge N + (M - P * 4) \wedge F \text{ AND } (-b) \wedge 2 / C + 6$
- $a + b - c \text{ OR } (x * 2 \wedge n) - m/p \text{ AND } F \wedge 2$
- $a \wedge (b/2) \text{ AND } (x - y/3 + w) \text{ OR } (c \wedge 3) \wedge 2$
- $N \wedge x - (p * 4 + M)/F \text{ OR } -c / b \wedge 2$
- $B - a + c \text{ AND } n \wedge x - (p/M \text{ OR } F \wedge 2)$

11. Execute the following postfix notation using stack

$ab*cde\wedge+$, when $a=5, b=6, c=8, d=2, e=2$.

12. Execute the following infix notation using stack

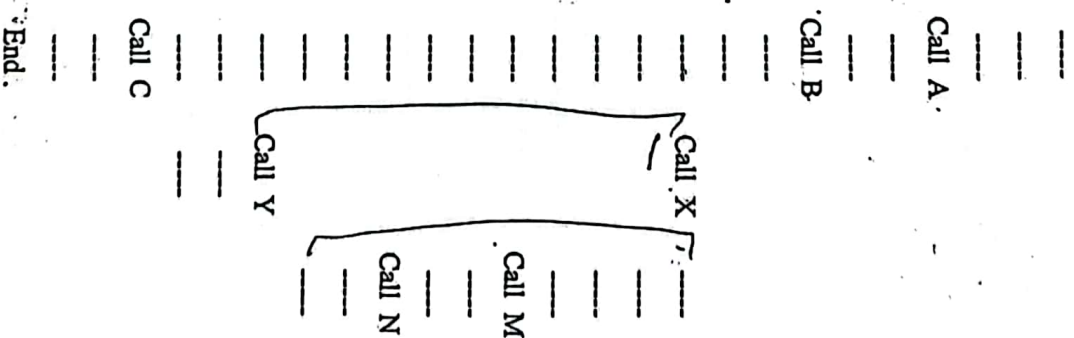
$a / (b * c/2) \wedge 4 + m$ If $a=10, b=8, c=4, m=20$.

13. Write an algorithm that reads in a string and prints its characters in reverse order [Note: the string terminator is a '.' which should not be printed as a part of the reversed string]

14. Write a C++ segment of code that reads in a string and prints its characters in reverse order [Note: the string terminator is a blank, which should not be printed as a part of the reversed string] .

15. Write a C++ segment of code to add an element to the queue called (LINE) of size (30) characters.

9. Consider the following program structure
Begin { main program }



الفصل الرابع الهياكل الموصولة Linked Structures

Storage Allocation	التخصيص الخزني	1-4
Sequential Storage Allocation	الخزن التسلسلي	1-1-4
Dynamic Storage Allocation	الخزن الديناميكي	2-1-4
	المقارنة بين الخزن التسلسلي والخزن الديناميكي	3-1-4
Pointers	المؤشرات	2-4
Linked List	القائمة الموصولة	3-4
Linked Stack	المكس الموصول	4-4
Linked queue	الطابور الموصول	5-4
Circular Linked list	القائمة الموصولة الدائرية	6-4
Double linked list	القائمة الموصولة المزدوجة	7-4

16. Write a C++ segment of code to delete an element from the circular queue called (BOB) of size (15) integers .
17. What are the main differences between simple queue and circular queue ? Explain that in detail.
18. What are the main applications of the queue in the computer field?

1-4 التخصيص الخرنى Storage Allocation

1-1-4 التخصيص الخرنى التسلسلى

Sequential Allocation of Storage

ان ابسط الطرق لخرن القائمة الخطية هو استخدام الخرن التسلسلى في ذاكرة الحاسوب أي يتم الخرن في مواقع متتالية (متسلسلة) ويمكن ان نعرف موقع أي عنصر اذا عرفنا موقع العنصر الاول الذي هو عنوان البداية (Base Address) ، ومواقع العناصر التالية ستكون مشيرة له . فالعنصر (k) سيكون موقعه تاليا لموقع العنصر (k-1) وهكذا بقية العناصر .

المزايا : Advantages

- 1- اكثر سهولة في التمثيل والتطبيق .
- 2- يكون اقتصاديا اكثر لأنه يستخدم مساحة خرنية اقل .
- 3- اكثر كفاءة في الوصول العشوائي .
- 4- مناسب جدا عند التعامل مع المكس .

المساوئ : Disadvantages

- 1- صعوبة تنفيذ عمليات الإضافة والحذف .
 - 2- يتطلب التعريف المسبق وتحديد عدد العناصر المطلوب خرنها .
- ### 2-1-4 التخصيص الخرنى الديناميكي

Dynamic Allocation of Storage

ان الطريقة الاخرى للخرن هي استخدام رابط (link) او مؤشر (pointer) مع كل عنصر يحتوي عنوان موقع العنصر التالي لذلك لا توجد ضرورة لخرن البيانات في مواقع متتالية (متسلسلة) بل يمكن خرن أي عنصر بياني في أي موقع ، ولهذا فكل عنصر (عقدة) يكون من جزئين هما :

الجزء الاول : يحتوي البيانات (Data)
الجزء الثاني : حقل يحتوي على عنوان موقع العنصر التالي (link)



2-4 المؤشرات Pointers

التي نستطيع فهم تمثيل وصل الهياكل البيانية باستخدام الخزن الديناميكي لا بد من شرح كيف يمكن تعريف المؤشرات وطريقة استخدامها برمجيا.

عند التعامل مع المصفوفات (arrays) سبق ان استخدمنا الدليل (index) للوصول الى موقع احد عناصرها ، أي ان الدليل عبارة عن متغير استخدمه البرنامج في الوصول الى الموقع المطلوب ويستخدمه المترجم (compiler) كعنوان لموقع معين في الذاكرة بنفس الاسم (اسم الدليل) .

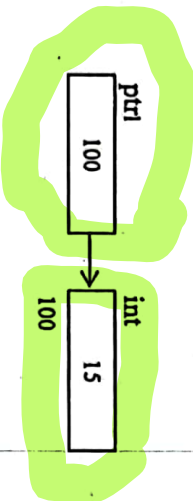
ان هذا المؤشر (الدليل) يعتبر مؤشرا نسبيا (relative) أي يدلنا على موقع العنصر بالنسبة الى موقع بداية (Base Address) خزن المصفوفة في الذاكرة .

هناك حالات تستوجب بناء هياكل بيانية مختلفة السعة الخزنية خلال مرحلة تنفيذ البرنامج (program execution stage) وفي الهياكل البيانية الموصولة (linked structures) (data structures) اذ يكون لكل عنصر بياني فيها حقل (جزء) اضافي (link) يستعمل كمؤشر يشير الى موقع العنصر التالي ، أي ان هذه الهياكل تكون (تتم) وتضيف عناصر جديدة لها بصورة ديناميكية خلال مرحلة تنفيذ البرنامج عند الحاجة لمواقع جديدة .

وبرمجيا بلغة C++ يكون تعريف المؤشرات كالآتي :

```
int *ptr1 ;
```

ويعني تعريف المتغير (ptr1) كمؤشر ، وهو يشير الى عنوان موقع (address) يحتوي بيانات نوعها (int) ويقرأ هكذا (ptr1 مؤشر لموقع من نوع int) .



وبهذه الطريقة يمكن تعريف مؤشرات أخرى لكل نوع متغير ، فمثلا " الايمان التالي :

```
char *ptr2 ;
```

سيعني تعريف المؤشر (ptr2) ليشير الى عنوان موقع (address) يحتوي بيانات نوعها (char) ، وهكذا لكل نوع بياني اخر float ، double ، ... الخ .

الخلاصة:

- سهولة تنفيذ عمليات الاضافة والحذف لأي عنصر اذ لا يتطلب اكثر من كحيت قيمة محل المؤشر الذي يعطي عنوان الموقع التالي ..

المصادر:

- يحتاج الى مساحة خزنية اكبر لتمثيل محل المؤشر اضافة الى البيانات الاسمية.

4-3 المقارنة بين الخزن التسلسلي والخزن الديناميكي

يمكن ان تتركز المقارنة في النقاط التالية :

أ- المساحة الخزنية Amount of Storage

ان اسلوب الخزن الديناميكي يحتاج الى مساحة خزنية اكبر لأن كل عنصر في الهيكل البياني يحتاج الى مؤشر (أي موقع خزن اضافي) يحتوي عنوان موقع العنصر التالي ، وهكذا لجميع العناصر .

ب - عمليات الاضافة والحذف

ان هذه العمليات اسهل تنفيذا في الخزن الديناميكي لأنها لا تتطلب غير تغيير قيمة المؤشر ليحتوي عنوان موقع العنصر بعد الاضافة او الحذف . اما في الخزن التسلسلي فإنه يتطلب عمليات تحريف (shifting) للعناصر .

ج- الوصول العشوائي للعناصر Random Access

ان اسلوب الخزن التسلسلي يعتبر افضل في الوصول عشوائيا لأي عنصر من عناصر الهيكل البياني مباشرة (k-th element from the start) أما في الخزن الديناميكي فإنه يتطلب البدء من اول عنصر ثم العناصر التالية بالتتابع لحين الوصول الى موقع خزن العنصر المطلوب .

د- الدمج والافصال Merge & Split

في الخزن الديناميكي تكون هذه العمليات اسهل تنفيذا وذلك بتغيير قيمة المؤشر للعناصر (المقد) في مواقع الدمج او الفصل اما في الخزن التسلسلي فالعمل اكثر تعقيدا اذ قد يتطلب تحريف العناصر واصادة تنظيم الخزن (Reorganization of Storage) .

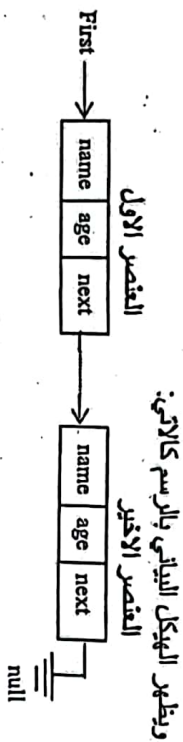
مثال:
لتعريف هيكل بياني موصول (Linked Structure) تتكون عناصره البيانية من جزأين هما (اسم الطالب - عمره).
سيكون تعريفه وفق الآتي :

```

structure student
{
    char name ;
    int age ;
    student *next ;
};

```

تعريف قيد الطالب ومكوناته هي :
الاسم : نوع (string)
العمر : نوع عدد صحيح
المؤشر : نوعه (int) (المعرف اعلاه)



لاحظ ان مؤشر العنصر الاخير قيمته (null) ويعني لا شيء بعده .

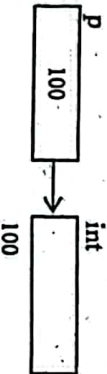
1- البرنامج الفرعي (new)
توفر لغة ++C استخدام (new) للحصول على مواقع خزنه في الذاكرة ، وتحديد المؤشر (pointer) الذي يشير الى تلك المواقع ، ولناخذ المثال التالي للعرض .
تعريف المؤشرين p , q لموقعين ذي صفة int

```

int *p , *q ;
int new int ;

```

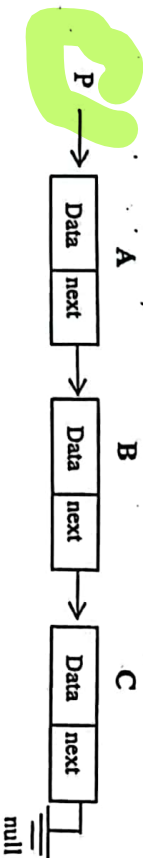
+ استخدم new لحجز موقع في الذاكرة لمؤشر
نوعه int ويكون عنوان الموقع في المؤشر p



ومن الصنيع البرمجية الأخرى للتعامل مع المؤشرات ما يلي :

1 / إذا كان لدينا متغير مثل `int (var)` :
فلمعرفة عنوان الموقع يمكن باستخدام الأيعاز
`ptr = &var`
أذا ان إشارة (&) تعني عنوان موقع (address) المتغير (var) .

ب / لطبع محتويات الموقع الذي يشير اليه المؤشر (ptr) يستخدم الأيعاز
التالي :
`cout << *ptr ;`
أذا ان (*ptr) تعني محتوى الموقع الذي يشير اليه (ptr) وهو (15) كما
في الرسم السابق .
لناخذ الهيكل البياني الموصول التالي:-



لان كل عنصر يتكون من جزأين فيعرف كـ (Structure) وفق الآتي :

```

struct node
{
    int data ;
    link *next ;
};

```

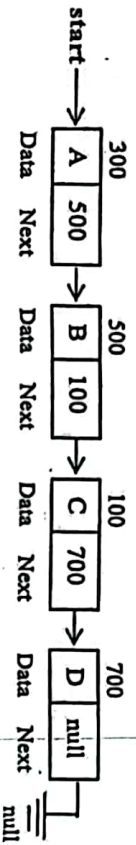
هذا التعريف يعني ان العنصر البياني اسمه (node) لهذا الهيكل (structure)
يتكون من جزأين :
الجزء الاول : Data ونوعه (int) ، ويمكن تعريفه بأي نوع اخر حسب
نوع البيانات .
الجزء الثاني : next وهو نوع مؤشر سيحتوي عنوان موقع (address) العنصر
التالي .

3-4 القائمة الموصولة Linked List

هي مجموعة من العناصر (العقد) التي كل منها يحتوي البيانات (Data) والمؤشر (link) الذي يشير الى العنصر (العقدة) التالي في القائمة. فالعنصر (x) يتكون من الجزئين Data , Next

x :
Data (x) | Next (x)

مثال :
لناخذ قائمة موصولة تتكون من اربعة عناصر كالآتي :-

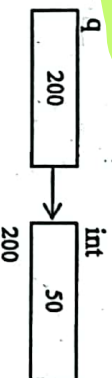


نلاحظ ما يلي :

start : هو المؤشر الرئيسي الذي يشير الى بداية اول عنصر في القائمة وسنكون قيمته في هذا المثال (300).

- next (A) = 500 : محتويات حقل المؤشر للعنصر الاول هو (500) ويدل على موقع العنصر الثاني.
- next (B) = 100 : محتويات حقل المؤشر للعنصر الثاني هو (100) ويدل على موقع العنصر الثالث.
- next (C) = 700 : محتويات حقل المؤشر للعنصر الثالث هو (700) ويدل على موقع العنصر الرابع.
- next (D) = null : محتويات حقل المؤشر للعنصر الرابع هو (null) ويدل على انه لا يوجد عنصر بعد العنصر الرابع.

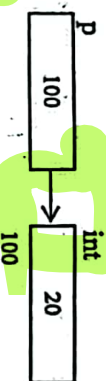
q = new int(50);



+ استخدم new لحجز موقع في الذاكرة لمؤشر نوعه int وقيمته (50) ويكون عنوان الموقع في المؤشر (q)

*p = 20;

+ وضع القيمة (20) في الموقع الذي يشير اليه p



cout<<*p<<;

+ اطبع محتويات الموقع الذي يشير اليه المؤشر p وسنكون (20) كما في الرسم السابق

cout<<*q<<;

+ اطبع محتويات الموقع الذي يشير اليه المؤشر q وسنكون (50) كما في الرسم السابق

ك-البرنامج الفرعي (delete)

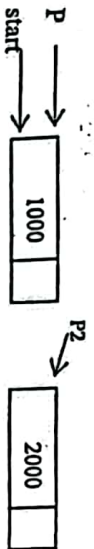
يستخدم هذا الايمان في لغة C++ ، ووظيفته هو تحرير الموقع الذي حجز باستخدام (new) ويكتب بالصيغة التالية :

delete p ;

ليني تحرير موقع الذاكرة الذي يشير اليه المؤشر (p) اي ان البرنامج لا يحتاج الى استخدامه وهذا يحقق الخاصية المهمة للخرن الديناميكي الذي يقتن استخدام المساحة الخزنية بقر الحاجة الفعلية فيسمع بخلق (حجز) الموقع عند الحاجة اليه وحذفه او تحريره عند انتهاء استخدامه. لذا فعمل تنفيذ عملية حذف أي عنصر من الهيكل الموصول يستخدم (delete) بعده مباشرة .

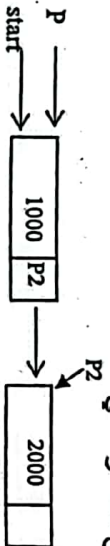
```
cin<<(p2->data);
```

ادخال بيانات العنصر الثاني ولكن (2000)



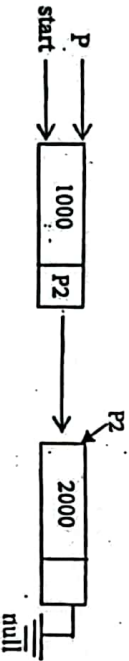
```
p->next = p2;
```

تكن قيمة حقل المؤشر في العنصر الأول هي (p2) أي موقع العنصر الثاني

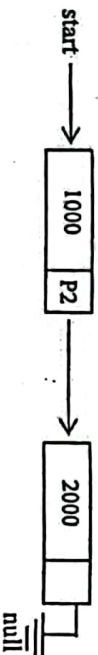


```
p2->next = null;
```

اجعل مؤشر العنصر الثاني null



وبعض النظر عن كل من p, p2 التي استخدمت بشكل وقفي فإن الشكل النهائي للقائمة هو:



Create a Linked list (of N node).

3. تكوين قائمة موصولة تحتوي (N) من العناصر

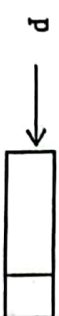
```
p = new node;
start = p;
cout<<" How many elements you like to create : ";
cin>n;
for (int i = 1; i <= n; i++)
{
    cin>>(p->data);
```

113

فهيبدأ بإنشائي بعض الصرغ البرمجية (مقطع من برنامج) (segment of code of C++ program) لتوضيح كيفية تكوين (خلق) هيكل موصول (Linked Structure) أو تنفيذ بعض العمليات كالإضافة والحذف مع استخدام التعريف الوارد في صفحة (108).

1. تكوين قائمة موصولة من عنصر واحد (Create a Linked list (of one node).

```
p = new node;
```



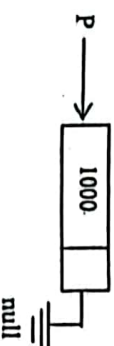
```
start = p;
```



```
cin>>p->data;
```

ادخال بيانات العنصر ولكن (1000)

```
p->next = null;
```

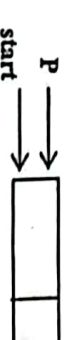


2. تكوين قائمة موصولة من عنصرين (Create a linked list (of two nodes).

```
p = new node;
```



```
start = p;
```

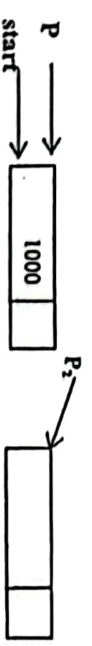


```
cin<<(p->data);
```

ادخال بيانات العنصر الأول ولكن (1000)

تكوين موقع لعنصر جديد يشير اليه p2

```
p2 = new node;
```



112

5. طبع (عرض) جميع عناصر القائمة الموصولة

```
void display()
{
    node *p ;
    p = first ;
    while ( p != null )
    {
        cout<<(p->data) ;
        p = p->link ;
    }
}
```

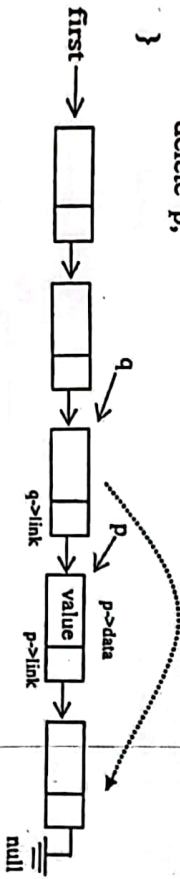
تحريك المؤشر p الى الموقع التالي

6. حذف عنصر (ذو قيمة محددة) لكن (value) من القائمة الموصولة

To delete an element of certain value

```
void deleteval( char value )
```

```
{
    node *p, *q ;
    p = first ;
    while ( p->data != value )
    {
        q = p ;
        p = p->link ;
    }
    q->link = p->link ;
    delete p ;
}
```



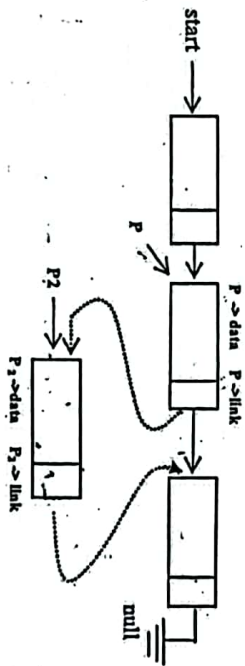
115

```
if ( i != n )
    p2 = new node ;
else p2 = null ;
p->link = p2 ;
p = p2 ;
}
```

4. إضافة عنصر واحد الى القائمة الموصولة بعد الموقع الذي يشير اليه المؤشر (p)

```
void addafterp();
```

```
{
    p2 = new node ;
    cin>>(p2->data) ;
    p2->link = p->link ;
    p->link = p2 ;
}
```



[*] موقع هذه الخطوة مهم جداً ، إذ انها تجعل قيمة المؤشر للعنصر الجديد هي نفس قيمة مؤشر العنصر السابق (p->link) قبل تغييره في الخطوة التالية .

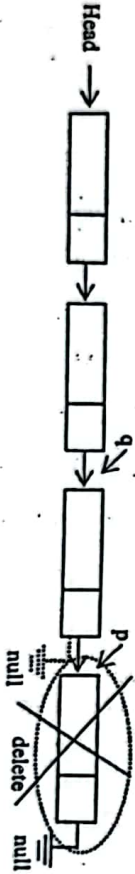
114

8. حذف العنصر الأخير في القائمة الموصولة

```

node *p, *q;
p = head;
if (p->link == Null)
{
    delete p;
    head = null;
}
else
{
    while (p->link != Null)
    {
        q = p;
        p = p->link;
    }
    q->link = Null; delete p;
}

```



117

لاحظ هنا ما يلي :

+ استخدام المؤشر (p) في موقع ما ، ثم نقيمه المؤشر (q) في الموقع الذي يسبقه ، ونحذف المؤشر إن معناه لحين الوصول الى الموقع المطلوب حذفه وفق الشرط المحدد بالمعادلة الشرطية (while) .
ويمكن استخدام هذه الصيغة في معظم عمليات الحذف بعد صياغة الشرط المناسب.
+ نختار قيمة مؤشر للعنصر في الموقع السابق الذي يشير اليه المؤشر (q) لنشير الى موقع للعنصر التالي بعد الموقع الذي يشير اليه (p) ، لأن العنصر الذي يشير اليه (p) سيحذف .
+ من المهم استخدام (delete p) لتحرير الموقع الذي يشير اليه (p) وإعلانه للذاكرة لاستخدام لاحق .

7. حذف العنصر الأول في القائمة الموصولة

```

node *p;
p = start;
start = start->link;
delete p;

```



116

وإضافة عنصر واحد الى نهاية القائمة الموصولة
Segment of code of C++ program to add one element to the end of the linked list pointed by (start).

```
p = start ;
while ( p->link != NULL )
    p = p->link ;
q = new node ;
cin >> (q->data) ;
q->link = NULL ;
p->link = q ;
```

إضافة عنصر بعد الموقع الذي ترتيبه (n) في القائمة الموصولة
عندما يكون عدد عناصر القائمة أكبر أو يساوي 10

```
cout << "\n Enter the position (n) : ";
cin >> n ;
p = head ;
for ( int i = 1 ; i <= n-1 ; i++ )
    p = p->link ;
q = new node ;
cin >> (q->data) ;
q->link = p->link ;
p->link = q ;
```

تصريف : لكتب برنامج فرعي (function) لتلب (Invert) ترتيب عناصر القائمة الموصولة ، فمثلا إذا كان (X) يشير الى بداية عناصر القائمة بحيث :
 $X = (a_1, a_2, a_3, \dots, a_n)$ ، فالمطلوب أن تصبح عناصر القائمة كالآتي :
 $X = (a_n, a_{n-1}, \dots, a_2, a_1)$

الحل

```
void Invertll(char *X)
{
    node *p, *q, *r ;
    p = X ;
    q = Null ;
    while ( p != NULL )
    {
        r = q ;
        q = p ;
        p = p->link ;
        q->link = r ;
    }
    X = q ;
}
```

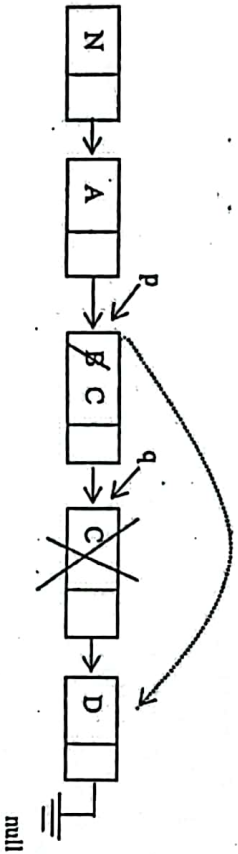

12. حذف عنصر في الموقع (p) من القائمة الموصولة

هذه الحالة تقترض أن كل من المؤشر الرئيسي للقائمة و بيانات العناصر غير

معرّفين، لذا فإن فكرة الحل هنا هي كالآتي :

- + نسخ قيمة العنصر في الموقع اللاحق إلى الموقع (p)
- + حذف العنصر في الموقع اللاحق الذي نسخنا قيمته في الخطوة السابقة.

```
q = p->link ;
p->data = q->data ;
p->link = q->link ;
delete q ;
```



■ لاحظ أن عناصر القائمة أصبحت NACD بعد أن كانت NABCD

تبرير : إضافة عنصر واحد إلى قائمة موصولة، عناصرها مرتبة (ordered linked list) بصورة تصاعديه (Ascending) على أن تبقى عناصر القائمة مرتبة بعد الإضافة.

الحل :

```
cin >> value ;
p = start ;
while ( p->data < value )
{
    q = p ;
    p = p->link ;
}
```

11. إضافة عنصر قبل العنصر في الموقع (p) للقائمة الموصولة

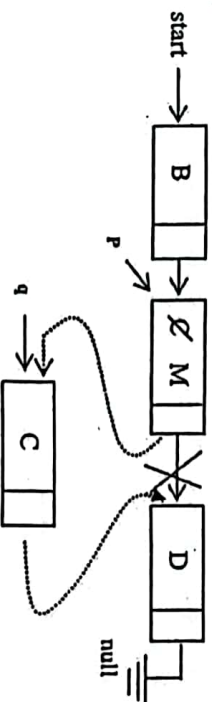
أن هذه الحالة تقترض أن المؤشر الرئيسي للقائمة مجهول وكذلك بيانات العناصر

مجهولة، لذا فإن فكرة الحل هنا هي كالآتي :

- + خلق العنصر الجديد
- + إضافة العنصر الجديد بعد العنصر في الموقع (p)
- + استبدال قيمة العنصر الجديد مع قيمة العنصر في الموقع (p)

```
q = new node ;
q->data = p->data ;
cin >> ( p->data ) ;
q->link = p->link ;
p->link = q ;
```

قراءة العنصر الجديد وليكن (M) بدلاً من قيمته السابقة (C)



■ لاحظ أن عناصر القائمة أصبحت BMCD بعد أن كانت BCD

```

p->link = y ;
}
} return z ;
}

```

تبرين: اكتب برنامج فرعي (function) لتجزئة (split) القائمة الموصولة التي مؤشرها الرئيسي (start) الى قائمتين موصولتين أحدهما (first) تحتوي على جميع العناصر في المواقع الفردية للقائمة الأصلية ، والقائمة الثانية (second) تحتوي على جميع العناصر في المواقع الزوجية للقائمة الأصلية .

الحل:

```

void split (char *start)
{
    node *first, *second, *p, *n, *m; int L ;
    first = NULL ; second = NULL ;
    p = start ; L = 0 ;
    while (p != NULL)
    {
        L = L + 1 ;
        if (L%2 != 0)
        {
            if (L == 1)
            {
                first = p ;
                n = first ;
            }
            else
            {
                n->link = p ;
                n = p ;
            }
        }
    }
}

```

123

```

t = new node ;
t->data = value ;
t->link = p ;
q->link = t ;

```

تبرين: استبدل (Exchange) قيمة عنصر في موقع معين (i) للقائمة الموصولة (start) مع قيمة العنصر في موقع آخر (j) في نفس القائمة على ان يكون (i < j) .

الحل:

```

p = start ;
for (int n = 1 ; n <= i-1 ; n++)
    p = p->link ;
q = p->link ;
for (int n = i+1 ; n <= j-1 ; n++)
    q = q->link ;
x = (p->data) ;
p->data = q->data ;
q->data = x ;

```

تبرين: اكتب برنامج فرعي (function) لتنفيذ عملية دمج (merge) القائمة الموصولة التي مؤشرها الرئيسي (y) في نهاية عناصر القائمة الموصولة التي مؤشرها الرئيسي (x) .

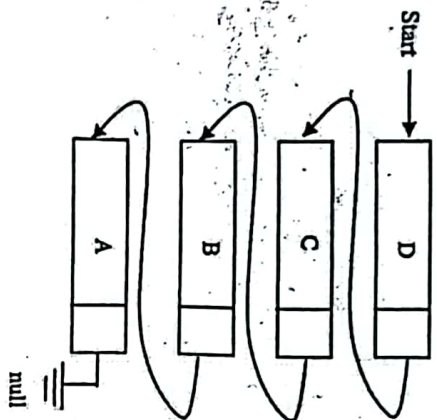
الحل:

```

void merge(char *x, *y)
{
    node *z, *p ;
    if (x == NULL)
        z = y ;
    else
    {
        z = x ;
        if (y != NULL)
        {
            p = x ;
            while (p->link != NULL)
                p = p->link ;
        }
    }
}

```

122



- المؤشر (start) يشير الى قمة المكس حيث العنصر (D) ، وهو يمثل النهاية المقترحة حيث تنفذ عمليات الاضافة و الحذف .
- العنصر (A) في قعر المكس ، وهو يمثل النهاية المغلقة ، مع ملاحظة ان حقل المؤشر (link) لهذا العنصر هو (NULL) اذ لم يسبقه شيء .

برنامج فرعي (function) لإضافة عنصر الى المكس الموصول

```

void pushls(int item)
{
    node *p ;
    p = new node ;
    p->data = item ;
    if (start == NULL)
        p->link = NULL ;
    else
        p->link = start ;
    start = p ;
}
  
```

```

    }
}
else
{
    if (L==2)
    {
        second = p ;
        m = second ;
    }
    else
    {
        m->link = p ;
        m = p ;
    }
}
}
p = p->link ;
}
n->link = NULL ; m->link = NULL ;
return first, second ;
}
  
```

4-4 المكس الموصول Linked Stack

يمكن الاستفادة من خصائص الخزن الديناميكي لتمثيل المكس باعتباره حالة خاصة من القائمة الخطية التي تكون عمليات الاضافة و الحذف من نهاية واحدة (هي النهاية المفتوحة) .
 ان مبدأ عمليات الاضافة و الحذف هي نفسها التي سبق ذكرها ، الا ان الفرق يكون في طريقة التمثيل في الذاكرة .
 والشكل التالي يبين مكس موصول ذو أربعة عناصر

البرنامج الفرعي (function) لضيق عنصر من المكس المتصل

```

int pops()
{
    int item ;
    if (start == NULL)
        cout<<"the linked stack is empty " ;
    else
    {
        q = start ;
        item = q->data ;
        start = q->link ;
        delete q ;
    }
    return item ;
}

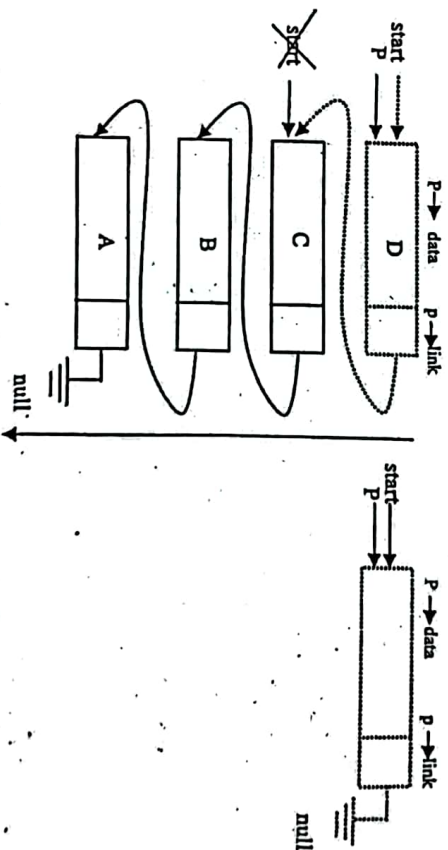
```

- ان خطوات هذا البرنامج الفرعي هي :
- 1- عندما يكون المؤشر الرئيسي (start = NULL) فان المكس خالي وعملية الحذف لن تعد ممكنة.
 - 2- استخدم المؤشر (q) ليشير الى بداية المكس (اول عنصر في المكس).
 - 3- اخذ (سحب) قيمة العنصر الأول الموجودة في الحقل (data) وخذتها وكتبها في المتغير (value).
 - 4- تحديث قيمة المؤشر (start) ليشير الى موقع العنصر التالي المحددة بالحقل (q->link).
 - 5- تحرير الموقع الذي كان يشغله العنصر المحذوف والذي يشير اليه المؤشر (q) باستخدام (delete q).
 - 6- الرسم التوضيحي يبين كيفية تنفيذ الخطوات أعلاه.

ان هذا البرنامج الفرعي يعتمد التوزيع الوارد في الصفحة (108) فيما يتعلق

بمعاصر المكس مع ملاحظة ما يلي :

- 1- ان المؤشر (start) هو المؤشر الرئيس لبداية عناصر المكس ، اي المؤشر الذي ينظر المؤشر (top).
- 2- لا نحتاج الى خطوة للتحقق من امتلاء المكس (stack full) لاننا نستطيع خلق العنصر عند الحاجة اليه ومن ثم ربطه بالعنصر السابق له.
- 3- بموجب أول إيجازين سنخلق العنصر المطلوب أضاقته (المؤشر p يشير اليه) وندخل بياناته.
- 4- أن العبارة التي تلي الجملة الشرطية (if) هي لمعالجة الحالة عند خلق أول عنصر في المكس.
- 5- عبارة (else) هي لجعل قيمة حقل المؤشر للعنصر الجديد (المطلوب أضاقته) تشير الى موقع العنصر السابق والذي يشير اليه (start).
- 6- الخطوة الأخيرة هي لتحديث المؤشر (start) ليشير الى العنصر الجديد بعد ان اصبح في مقدمة المكس.
- 7- الرسم التوضيحي يبين كيفية تنفيذ الخطوات أعلاه.



حالة اضائة اعتيادية لعنصر إلى مكس موجود اصلا ويتكون من ثلاثة عناصر

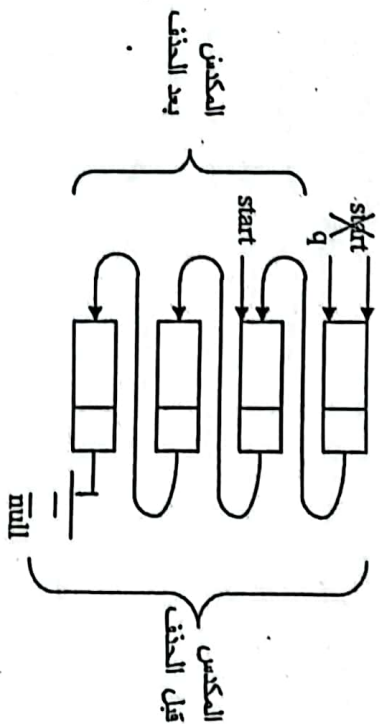
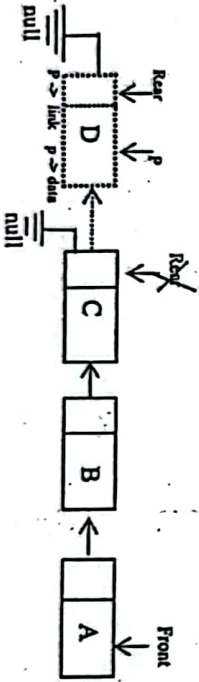
حالة خلق أول عنصر في المكس (بداية تكوين المكس)

البرنامج الفرعي (function) لإضافة عنصر الى الطابور الموصل

```
void add(q)
{
    p = new node;
    cin >> (p->data);
    p->link = NULL;
    if (rear == NULL)
        front = p;
    else
        rear->link = p;
    rear = p;
}
```

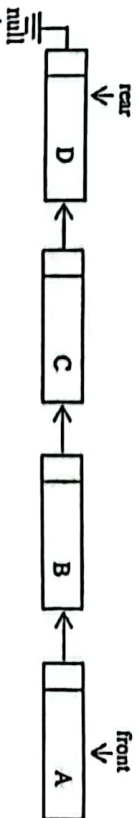
ان خطوات هذا البرنامج الفرعي هي كالآتي :

- 1- أن الخطوات الثلاث الأولى هي لغرض خلق (تكوين) العنصر وإدخال قيمته وجعل قيمة المؤشر (NULL).
- 2- العبارة التي تلي الجملة الشرطية (if ...) هي لمعالجة الحالة عند خلق أول عنصر في الطابور ، ويشير اليه المؤشر (front).
- 3- في عبارة (else) يتم تحديث قيمة حقل المؤشر العنصر الأخير في الطابور جعله يشير الى العنصر الجديد الذي يشير اليه (p).
- 4- أن الخطوة الأخيرة هي تحديث قيمة المؤشر (rear) ليشير الى العنصر الجديد (المضاف) بعد أن أصبح هو الأخير.
- 5- الرسم التوضيحي التالي يبين إضافة العنصر (D).



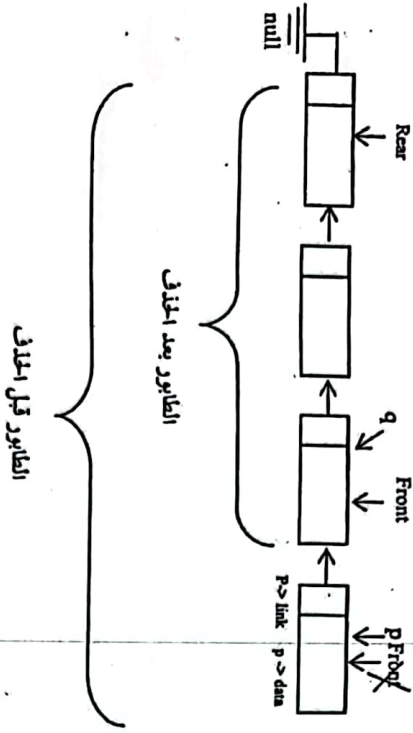
5-4 الطابور الموصل Linked Queue

كما سبق ان متنا المكس باستخدام الخزن الديناميكي يمكن تمثيل الطابور بنفس الطريقة مع وجود المؤشرين (front) ، (rear) وسيظهر الطابور كالآتي :

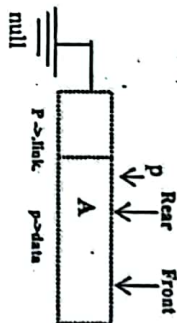


- المؤشر (front) يشير الى أول عنصر في الطابور.
- المؤشر (rear) يشير الى آخر عنصر في الطابور.
- كل عنصر من عناصر الطابور الأربعة (D,C,B,A) فيه حقل يحتوي قيمة المؤشر الى العنصر التالي.
- مؤشر العنصر الأخير قيمته (null) ، إذ لا يوجد بعده عناصر.

- في المتغير (item) .
- 3 - الخطوة الرابعة في عبارة (else) هي لتحديث قيمة المؤشر (front) ليشير الى موقع العنصر الثاني حيث يشير (q).
 - 4 - الخطوات الأخيرة هي لمعالجة حذف اخر عنصر في الطابور ، بما يتطلب جعل قيمة كل من المؤشرين (front) ، (rear) هي (NULL).
 - 5 - الرسم التوضيحي التالي يبين كيفية تنفيذ الخطوات أعلاه :



- 6 - والرسم التوضيحي التالي يبين تكوين أول عنصر في الطابور وهو (A).



البرنامج الفرعي (function) لحذف عنصر من الطابور الموصلة

```
void deleteq()
{
    int item ;
    p = front ;
    if( p == NULL )
        cout<<"n the linked queue is empty " ;
    else
    {
        q = p->link ;
        item = p->data ;
        delete p ;
        front = q ;
        if ( front == NULL )
            rear = NULL ;
    }
    return item ;
}
```

- ان خطوات هذا البرنامج الفرعي هي :
- 1 - استخدام مؤشر وقي (p) ليشير الى اول عنصر في الطابور ، حيث يشير المؤشر (front) ، وعندما تكون قيمته (NULL) فهذا يعني ان الطابور خالي من العناصر ولا يمكن تنفيذ عملية الحذف .
 - 2 - في مقامة عبارة (else) نستخدم مؤشر ثاني هو (q) يشير الى العنصر التالي في الطابور لكي نستطيع حذف العنصر الاول بعد تخزين قيمته ولفي

تعريف: اكتب برنامج فرعي لنسخ جميع عناصر الطابور الموصول المتسلسل (linked queue) الى مكس متسلسل (sequential stack) خلال من العناصر بحيث اول عنصر في الطابور يصبح اعلى عنصر في المكس الحل: نستخدم التعريفات التالية

```
struct node
{
    int data;
    node *link;
} *front, *rear;
const int size=10;
int stack[size], s[size];
```

وفيما يلي البرنامج الفرعي:

```
void copy2(node *front, *rear;)
{
    int t=0; int top=0;
    node *f;
    f=front;
    while (f!=NULL)
    {
        p=f;
        ++t;
        s[t]=(p->data);
        f=(f->link);
    }
    while (t!=0)
    {
        ++top;
        stack[top]=s[t];
        --t;
    }
} Return top;
```

• لاحظ هنا استخدام المكسين (s) ، (stack) . ماهو السبب ؟

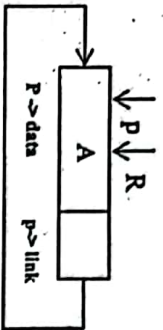
تعريف: اكتب برنامج فرعي (function) لنسخ جميع عناصر المكس المتسلسل (sequential stack) الى طابور موصول (linked queue) خلال من العناصر بحيث اعلى عنصر في المكس يصبح اول عنصر في الطابور. الحل: نستخدم التعريفات التالية:

```
struct node
{
    int data;
    node *link;
} *front, *rear;
const int size = 10;
```

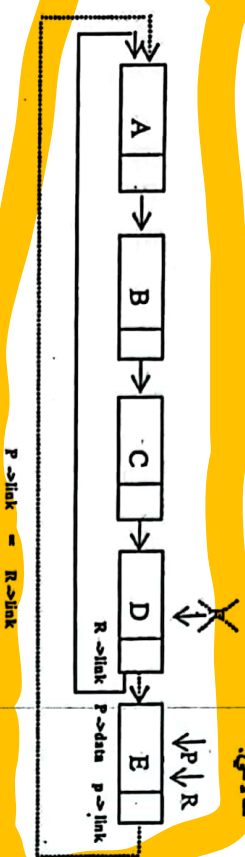
وفيما يلي البرنامج الفرعي:

```
void copy1(int stack[size], int top)
{
    int item;
    t=top; front=NULL; rear=NULL;
    while (t!=-1)
    {
        item = stack[t];
        --t;
        p = new node;
        p->data = item;
        p->link = NULL;
        if (rear == NULL)
            front = p;
        else
            rear->link = p;
        rear = p;
    }
    return *front, *rear;
```

}
في حالة إضافة أول عنصر (تكوين القائمة لأول مرة) فالرسم التالي يوضح ذلك :



أما في حالة إضافة عنصر مثل (E) الى قائمة موجودة أصلاً فالشكل يكون كالآتي:



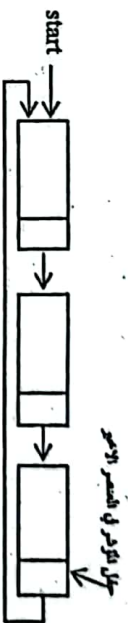
إلى تابع الفرعي (function) لإضافة عنصر من القائمة الموصولة الحاليه

```
void deletecll()
{
    int item ;
    if (r == NULL)
        cout << "the circular linked list is empty " ;
    else
    {
        f = r->link ;
        item = (f->data) ;
        if (r == f)
            r = NULL ;
        else
            r->link = f->link ;
        delete f ;
    }
}
```

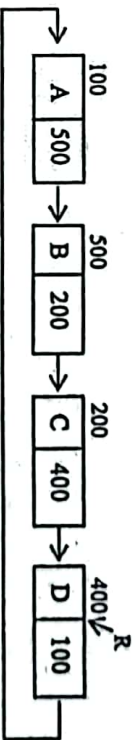
135

6-4 القائمة الموصولة الدائرية (Circular Linked List (Ring Structure))

(رسم الشكل الدائري) القائمة الموصولة الدائرية تستخدم مؤشر رئيسي يشير الى موقع أول عنصر، وحل المؤشر في العنصر الأخير تكون قيمته (NULL) إذ لا يتبعه عنصر آخر. في هذا الشكل الدائري (circular linked list) فإن حل المؤشر في العنصر الأخير سيؤشر الى العنصر الأول في القائمة كما في الشكل التالي :



في هذه الحالة يمكن استخدام مؤشر واحد فقط يشير الى العنصر الأخير ولكن (R) ويندلائه نستطيع الوصول الى العنصر الأول كما في الشكل التالي :



أن المؤشر (R) ستكون قيمته (400) يشير الى العنصر الأخير. وحل المؤشر للعنصر الأخير هو (R->link) نجد ان قيمته هي (100) وهذا يمثل عنوان موقع العنصر الأول .

إلى تابع الفرعي (function) لإضافة عنصر الى القائمة الموصولة الحاليه

```
void addcll()
{
    p = new node ;
    cin >> (p->data) ;
    if (r == NULL)
        p->link = p ;
    else
        p->link = r->link ; r->link = p ;
    r = p ;
}
```

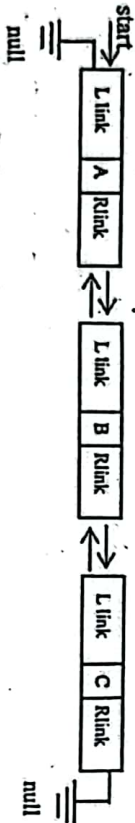
134

7-4 القائمة الموصولة المزدوجة Double Linked List

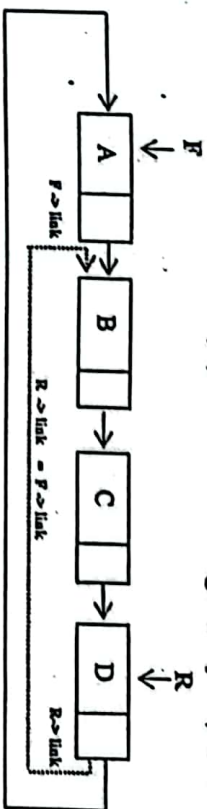
في القائمة الموصولة الاعتيادية (linked list) هلاك صعوبة في حذف العنصر الذي يشير اليه المؤشر (p) لأنه يتطلب العودة الى العنصر السابق له لتغيير حقل المؤشر فيه لينشئ الى العنصر اللاحق ، أي أن التحرك في هذه القائمة يكون باتجاه واحد أما القائمة الموصولة المزدوجة (double linked list) فإن كل عنصر فيها يحتوي على مؤشرين أحدهما يشير الى موقع العنصر اللاحق والآخر يشير الى موقع العنصر السابق ، أي أن كل عنصر في القائمة يتكون من ثلاثة أجزاء ، ويعرف برمجيا "بغلة C++ كالآتي :

```
struct node
{
    int data ;
    node *Llink ;
    node *Rlink ;
};
```

فالقائمة التالية تتكون من ثلاثة عناصر هي A ، B ، C



والرسم التالي لتوضيح حالة حذف العنصر (A) :



يرد اسم فرعي (function) المعروف (طغ) بمحتويات القائمة الموصولة الدائرية

```
void displayall()
{
    node *q ;
    if ( r == NULL )
        cout<<"\n the circular linked list is empty " ;
    else
    {
        q = r ;
        do
        {
            cout<<"\n <<((q->link) ->data) ;
            q = q->link ;
        }
        while( q!=r ) ;
    }
}
```

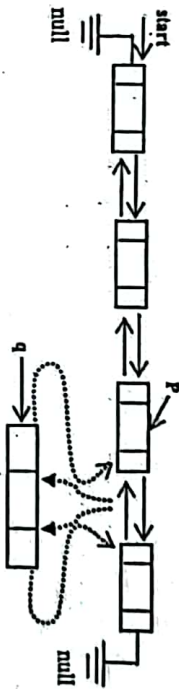
لاحظ الصيغة المركبة لإيمان الكتابة الثاني ، إذ أن (p->link) تعني موقع العنصر الأول.

أما العبارة بإكمالها فتعني كتابة حقل البيانات في موقع العنصر الأول ، وهكذا بالتتابع لبقية العناصر بعد تحريك المؤشر (p) .

أن خطوات حذف العنصر (p) ستكون كالآتي :

لتنفيذ الربط (<--- 5 --- >) : $(p \rightarrow Llink) \rightarrow Rlink = p \rightarrow Rlink$;
 لتنفيذ الربط (<--- 6 --- >) : $(p \rightarrow Rlink) \rightarrow Llink = p \rightarrow Llink$;
 delete p ;

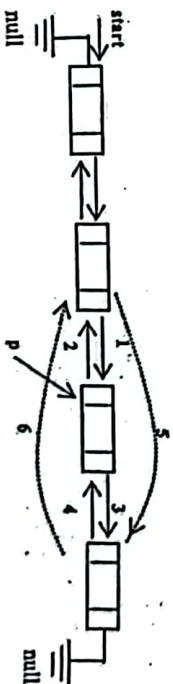
عملية إضافة عنصر بعد الموقع (p)
 لتأخذ القائمة الموصولة بالمرتبة التالية :



فإن سلسلة الأيعزات التالية تمثل خطوات تنفيذ إضافة العنصر (q) الى القائمة.

تكوين العنصر $q = \text{new node}$
 قراءة (إدخال) البيانات $\text{cin} \gg (q \rightarrow \text{data})$
 ربط العنصر الى يساره $q \rightarrow Llink = p$
 ربط العنصر الى يمينه $q \rightarrow Rlink = p \rightarrow Rlink$
 ربط العنصر اللاحق الى العنصر الجديد $(p \rightarrow Rlink) \rightarrow Llink = q$
 ربط العنصر p الى العنصر الجديد $p \rightarrow Rlink = q$

عملية حذف العنصر في الموقع (p)
 لدينا القائمة الموصولة بالمرتبة التالية :



```

void addbefore()
{ int k;
  cout<<"\n give the element where to insert the new item before it: ";
  cin>>item1;
  p=start;
  while (p->data!=item1)
  {
    p=p->link;
    cout<<"\n how many elements you like to add ? "; cin>>k;
    for(int i=1; i<=k; i++)
    {
      cout<<"\n enter the new element : "; cin>>item2;
      q=new node;
      q->data=p->data;
      p->data=item2;
      q->link=p->link;
      p->link=q;
      p=q;
    }
  }
  void deleteall(char value)
  {
    p=start;
    while(p->data!=value)
    {
      q=p;
      p=p->link;
    }
    q->link=p->link;
    delete p;
  }

  void deletepos()
  { int k;
    cout<<"\n give the position (sequence) of the element ";
    cin>>k;
    p=start;
    for( int i=1; i<=k; i++)
    {
      q=p;
      p=p->link;
    }
    cout<<"\n how many elements you like to delete : "; cin>>k;
    for(int i=1; i<=k; i++)
    {
      q->data=p->data;
      q->link=p->link;
      delete p;
      p=q->link;
    }
  }

  int main()
  { char item1; int k, choice; node *p, *first;
    choice=0;
    linklist l;
    do
    {
      cout<<"\n Representation of the linked list & its operations ";
      cout<<"\n _____ ";
    }
  }
}

```

141

برامج الفصل
تحميل القائمة للرصد (linked list) وعدد من عمليات الأمانة والملف

برامج رقم - 9 :

```

#include <iostream>
using namespace std;
struct node
{
  char data;
  node *link;
} *start;
class linklist
{
private:
  node *p, *p2, *q, *r;
  char item1, item2;
public:
  linklist()
  { start=NULL; }
  void create()
  { int n;
    p=new node;
    start=p;
    cout<<"\n how many elements you like to enter ? "; cin>>n;
    for(int i=1; i<=n; i++)
    {
      cout<<"\n enter the new element : ";
      cin>>p->data;
      if(i!=n)
      {
        p2=new node;
        else
        {
          p2=NULL;
          p->link=p2;
          p=p2;
        }
      }
    }
  }
  void addafter()
  { int k;
    cout<<"\n give the element value where to insert the new item after it: ";
    cin>>item1;
    p=start;
    while(p->data!=item1)
    {
      p=p->link;
      cout<<"\n how many elements you like to add ? "; cin>>k;
      for(int i=1; i<=k; i++)
      {
        cout<<"\n enter the new element : "; cin>>item2;
        p2=new node;
        p2->data=item2;
        p2->link=p->link;
        p->link=p2;
        p=p2;
      }
    }
  }
}

```

140

تحليل القائمة الموصولة وتقليد عطلي، طباعة عناصرها اليانية بصورة مكررة
(print reverse) ، رتب (invert) ترتيب عناصرها اليانية بشكل

مكرس .

```
#include <iostream>
using namespace std;
struct node
{
    char data;
    node *link;
} *start;

class linkedlist
{
public:
    node *p, *p2, *q, *t;
    char item1, item2;
    linkedlist()
    { start=NULL; }

void create()
{ int n;
  p=new node;
  start=p;
  cout<<" how many elements you like to enter ? "; cin>>n;
  for(int i=1; i<=n; i++)
  {
      cout<<" enter the new element: ";
      cin>>p->data;
      if(i==n)
          p2=new node;
      else
          p2=NULL;
      p->link=p2;
      p=p2;
  }
}

void invert(node *p)
{
    p=start;
    q=NULL;
    while(p!=NULL)
    {
        t=q;
        q=p;
        p=p->link;
        q->link=t;
    }
    start=q;
}

void printreverse(node *p)
{

```

143

```
cout<<"\n 1- Creation of a linked list ( char type )
cout<<"\n 2- Insertion after certain position (element)
cout<<"\n 3- Insertion before certain position (element)
cout<<"\n 4- deletion an element of certain value
cout<<"\n 5- deletion of an element(s) at certain (sequence ) position
cout<<"\n 6- display the content of the linked list
cout<<"\n 7- exit
cout<<"\n";
cout<<" select your choice: ";
cin>>choice;
switch(choice)
{
    case 1: create();
        break;
    case 2: addafter();
        break;
    case 3: addbefore();
        break;
    case 4: cout<<" give the value of the element you like to delete: ";
        cin>>item1;
        deleteval(item1);
        break;
    case 5: deleteinpos();
        break;
    case 6: if( start=NULL )
        cout<<" the linked list is empty ";
        else
        {
            cout<<" the elements of the linked list are: ";
            testart;
            while(t!=NULL)
            { cout<<"<data><" ";
              t=t->link;
            }
        }
        break;
    // end of switch
    // end of DO
    while(choice!=7);
    return 0;
} // end of main
```

142

برنامج رقم 11 : أمثلة عنصر ياتي واحد الى قائمة موصولة مرتبة تصاعدياً
 (Ascending ordered linked list) على أن تبقى عناصر القائمة مرتبة بعد الأخطاء.

```
#include <ostream>
using namespace std;
struct node
{
    int data;
    node *link;
} *start;
class linklist
{
private:
    node *p, *p2, *q, *f;
    int item1;
public:
    linklist()
    { start=NULL; }
    void create()
    { int n;
      p=new node;
      start=p;
      cout<<"\n how many elements you like to enter ? "; cin>>n;
      cout<<"\n You should enter the elements in ascending order ";
      for(int i=1; i<=n; i++)
      {
          cout<<"\n enter the new element ( Integer type ) : ";
          cin>>p->data;
          if(i!=n)
          {
              p2=new node;
              p2=NULL;
              p->link=p2;
              p=p2;
          }
      }
      void addtosorted()
      {
          cout<<"\n give the element value of the new item you like to Add : ";
          p=start;
          p2=new node;
          cin>>(p2->data);
          if ( p->data >= p2->data )
          { p2->link=p; start=p2; }
          else
          {
              while((p->data < p2->data) && p->link !=NULL)
              {
                  q=p;

```

145

```
if(p!=NULL)
{
    printreverse(p->link);
    cout<<p->data<<" ";
}
}
int main()
{ char item1; int k, choice; node *c, *first;
  choice=0;
  linklist l;
  do
  {
      cout<<"\n Representation of the linked list & its operations ";
      cout<<"\n _____ ";
      cout<<"\n 1 - Creation of a linked list ( char type ) ";
      cout<<"\n 2 - reverse the linked list ";
      cout<<"\n 3 - print the linked list in reverse order ";
      cout<<"\n 4 - display the content of the linked list ";
      cout<<"\n 5 - exit ";
      cout<<"\n select your choice : ";
      cin>>choice;
      switch(choice)
      {
          case 1:l.create();
            break;
          case 2:l.invert();
            cout<<"\n the linked list is inverted ";
            break;
          case 3:cout<<"\n the element(s) of the linked list in reverse order are : ";
            l.printreverse(start);
            break;
          case 4:cout<<"\n the elements of the linked list are : ";
            l.start;
            while(l!=NULL)
            { cout<<t->data<<" ";
              t=t->link;
            }
            break;
          // end of switch
          // end of DO
          while(choice!=5);
          return 0;
      } // end of main
  }
```

144

برنامج رقم 12: لاستبدال (Exchange) قيمة عنصر في موقع معين ولكن (i) القائمة المرسلة (start) مع قيمة العنصر الآخر في موقع ولكن (i) في نفس القائمة على أن تكون (i < j).

```
#include <iostream>
using namespace std;
struct node
{
    char data;
    node *link;
} *start;
class linkedlist
{
private:
    node *p, *p2, *q, *i;
    char item;
public:
    linkedlist()
    { start=NULL; }
    void create()
    { int n;
      p=new node;
      start=p;
      cout<<"\n how many elements you like to enter? "; cin>>n;
      for(int i=1; i<=n; i++)
      {
          cout<<"\n enter the new element ( CHAR type ) : ";
          cin>>p->data;
          if(i!=n)
              p2=new node;
          else
              p2=NULL;
          p->link=p2;
          p=p2;
      }
    }
    void exchange()
    { int i,j,m;
      cout<<"\n give the position (sequence) of the first element required to exchange : ";
      cin>>i();
      cout<<"\n Give the position (sequence) of the Second element required to exchange : ";
      cin>>j();
      p=start;
      for( n=1; n<=i; n++)
          p=p->link;
      q=p->link;
      for( m=i+1; m<=j; m++)
          item=p->data;
      p->data=q->data;
      q->data=item;
    }
};
```

147

```
p=p->link;
}
if (p->link !=NULL)
{ p2->link=p; q->link=p2; }
else
{ p->link=p2; p2->link=NULL; }
}
}
int main()
{ char item; int k,choice; node *q,*first;
  choice=0;
  linkedlist l;
  do
  {
      cout<<"\n Representation of the linked list & its operations ";
      cout<<"\n 1- Creation of a sorted (ascending) linked list(Integer type)";
      cout<<"\n 2 - Insertion of an element to the sorted list ";
      cout<<"\n 3 - display the content of the linked list ";
      cout<<"\n 4 - exit ";
      cout<<"\n select your choice : ";
      cin>>choice;
      switch(choice)
      {
          case 1: create();
            break;
          case 2: if( start==NULL )
              cout<<"\n the linked list is empty ";
            else
              l.addtosorted();
            break;
          case 3: if( start==NULL )
              cout<<"\n the linked list is empty ";
            else
            {
                cout<<"\n the elements of the sorted linked list after addition is : ";
                p=start;
                while(p!=NULL)
                { cout<<p->data<<" ";
                  p=p->link;
                }
            }
            break;
          // end of switch
          // end of DO
          while(choice!=4);
          return 0;
      } // end of main
  }
```

146

```

public:
    list() // constructor
    {
        start=NULL;
    }
    void push(int item)
    {
        node* p;
        p=new node;
        p->data=item;
        if(start==NULL)
            p->link=NULL;
        else
            p->link=start;
        start=p;
    }
    int pop()
    {
        if start==NULL)
            cout<<"Stack is Empty ";
        else
        {
            q=start;
            item=q->data;
            start=q->link;
            delete q;
        }
        return item;
    }
    void display()
    {
        q=start;
        while (q !=NULL)
        {
            cout<<" "<<(q->data)<<"end ";
            q=q->link;
        }
    }
}

int main()
{
    int choice,m,k; int count=0;
    list l;
    do
    {
        cout<<"\n Representation of the LINKED STACK & its operations ";
        cout<<"\n _____ ";
        cout<<"\n 1 - Add ( PUSH ) elements to the stack ( Integer type ) ";
        cout<<"\n 2 - Delete ( POP ) elements from the linked stack ";
        cout<<"\n 3 - Display the content of the linked stack ";
        cout<<"\n 4 - Exit ";
        cout<<"\n ";
        cout<<"\n ";
        cout<<"\n select your choice : ";
        cin>>choice;
        switch(choice)
    }
}

```

149

```

}
int main()
{
    char item; int k,choice; node *first;
    choice=0;
    list l;
    do
    {
        cout<<"\n Representation of the linked list & Exchange the values of any two elements ";
        cout<<"\n _____ ";
        cout<<"\n 1 - Creation of a of the linked list ( C++ type ) ";
        cout<<"\n 2 - Exchange any two elements in this linked list ";
        cout<<"\n 3 - display the content of the linked list ";
        cout<<"\n 4 - exit ";
        cout<<"\n ";
        cout<<"\n select your choice : ";
        cin>>choice;
        switch(choice)
        {
            case 1: create();
                break;
            case 2: exchange();
                cout<<"\n The Two elements are exchanged *** ";
                break;
            case 3: if( start==NULL )
                cout<<"\n the linked list is empty ";
                else
                {
                    cout<<"\n the elements of the linked list are : ";
                    print;
                    while(l!=NULL)
                    {
                        cout<<"<data>"<";
                        l=l->link;
                    }
                    break;
                }
            // end of switch
        }
        // end of DO
        while(choice!=4);
        return 0;
    } // end of main
}

```

برنامج رقم - 13 : تمثيل للكنس للبرول (Linked Stack) وتمثيل عدد من البيانات على

```

#include <iostream>
using namespace std;
struct node
{
    int data;
    node * link;
} *start;
class listack
{
private:
    node *p, *q;
}

```

148


```

node *p;
p=new node;
p->data=m;
p->link=NULL;
if (rear==NULL)
    front=p;
else
    rear->link=p;
rear=p;
}

int deleteq()
{
    int item;
    p=front;
    if (p==NULL)
        cout<<"\n the linked Queue is Empty "<<endl;
    else
    {
        q=p->link;
        item=p->data;
        delete p;
        front=q;
        if (front==NULL)
            rear=NULL;
    }
    return item;
}

void displayq()
{
    if (front==NULL)
        cout<<"\n the linked queue is empty ";
    else
    {
        cout<<"\n the content of the linked queue is "<<endl;
        q=front;
        while (q !=NULL)
        {
            cout<<q->data<<" ";
            q=q->link;
        }
    }
}

}

int main()
{
    int choice,m,k; int count=0;
    lqueue lq;
    do
    {
        cout<<"\n Representation of the LINKED QUEUE & its operations ";
        cout<<"\n ----- ";
        cout<<"\n 1 - Add ( Enqueue ) elements (int type) to the linked Queue ";
        cout<<"\n 2 - Delete ( Dequeue ) elements from the linked queue ";
        cout<<"\n 3 - Display the content of the linked queue ";
        cout<<"\n 4 - Exit ";
        cout<<"\n ";
        cout<<endl;
    }
}

```

151

```

{
    case 1:cout<<"\n howmany elements you like to Add ? ";cin>>m;
    for(int i=1; i<=m; i++)
    {
        cout<<" enter the data item "; cin>>k;
        lq.push(k);
        ++count;
    }
    break;
    case 2:cout<<"\n howmany elements you like to delete ? ";cin>>m;
    if (m>count)
        cout<<"\n the number of element(s) in the linked stack is only "<<count;
    else
    {
        for (int i=1; i<=m; i++)
        {
            lq.pop();
            --count;
        }
        break;
    }
    case 3:if (count==0)
        cout<<"\n the linked stack is empty ";
    else
    {
        cout<<"\n the content of the linked stack stack is : "<<endl;
        lq.display();
    }
    break;
} // end of switch
} // end of DO
while(choice!=4);
return 0;
}

```

برسج رقم 14: تميل الطاقور الوصول (Linked Queue) وتقبل عدد من العمليات

عليه .

```

#include <iostream>
using namespace std;
struct node
{
    int data;
    node *link;
} *front,*rear;
class lqueue
{
private:
    node *p,*q;
public:
    lqueue() // constructor
    {
        front=NULL; rear=NULL;
    }
    void addq(int n)
    {

```

150

```

linklist()
{ start=NULL; first=NULL; second=NULL;
  r=NULL; y=NULL; z=NULL;
}

void create()
{ int n;
  p=new node;
  start=p;
  cout<<"\n how many elements you like to enter ? "; cin>>n;
  for(int i=1; i<=n; i++)
  {
    cout<<"\n enter the new element : ";
    cin>>p->data;
    if(i!=n)
    {
      p2=new node;
      p2=NULL;
      p->link=p2;
      p=p2;
    }
  }

  void merge()
  {
    node *p;
    if(x=NULL)
    {
      z=x;
      if(y=NULL)
      {
        p=x;
        while(p->link!=NULL)
        {
          p=p->link;
          p->link=y;
        }
      }
      start=z;
    }
    void split()
    { node *p,*n,*r; int i;
      first=NULL; second=NULL;
      p=start; i=0;
      while(p!=NULL)
      {
        ++i;
        if(i%2 != 0)
        {
          if(i==1)
          {
            first=p; p=first;
          }
          else
          {
            n->link=p; n=p;
          }
        }
        else
        {
          if(i==2)

```

153

```

cout<<"\n select your choice : ";
cin>>choice;
switch(choice)
{
  case 1: cout<<"\n how many elements you like to Add ? "; cin>>m;
    for(int i=1; i<=m; i++)
    {
      cout<<" enter the data item : "; cin>>data;
      lq1->add(lq1);
      ++count;
    }
  case 2: cout<<"\n how many elements you like to delete ? "; cin>>m;
    if ( m>count )
    {
      cout<<"\n the number of element(s) in the linked queue is only "<<count;
    }
    else
    {
      for(int i=1; i<=m; i++)
      {
        lq1->delete(lq1);
        --count;
      }
    }
  case 3: if ( count==0 )
    cout<<"\n the linked queue is empty ";
    else
    {
      cout<<"\n the content of the linked queue is : "<<endl;
      lq1->display(lq1);
    }
  break;
}
// end of switch
// end of DO
while(choice!=4)
return 0;
}

#include <iostream>
using namespace std;
struct node
{
  int data;
  node *link;
} *start, *p, *q, *r, *first, *second;
class linklist
{
private:
  node *p, *p2, *q, *r;
  int item1, item2;
public:

```

152

برنامج رقم 15 : تنفيذ كل من عملية دمج (Merge) قائمتين موزعتين لتصبح إحداهما في نهاية الأخرى ، وعملية فصل (Split) قائمة موزعة إلى قائمتين ، تقسم الأولى عناصر القائمة الفردية من القائمة الأصلية ، وتقسم الثانية عناصر القائمة الزوجية من القائمة الأصلية .

```

break;
case 5: cout<<endl<<"the content of the linked list (X) is : ";
        ldisplay(first);
        break;
case 6: cout<<endl<<"the content of the linked list (Y) is : ";
        ldisplay(second);
        break;
        // end of switch
        // end of DO
        while(choice!=7);
        return 0;
    } // end of main

```

برنامـج رقم -16 : تميل القائمة المـرسـولة المـلـمـة (circular linked list) و عدد من عمليات الأضافة والحذف .

```

#include <iostream>
using namespace std;
class clist
{
private:
    struct node
    {
        int data;
        node *link;
    };
    struct node *r,*p;
public:
    clist();
    void addcl(int);
    void deleted();
    void display();
    ~clist():{}
    clist::clist()
    {
        r=NULL;
    }
    void clist::addcl(int item)
    {
        node *q;
        q=new node;
        q->data=item;
        if(r==NULL)
            q->link=q;
        else
            { q->link=r->link;
              r->link=q;
            }
        r=q;
    }
}

```

155

```

        { second=p; m=second; }
        else
        { m->link=p; m=p; }
    }
    p=p->link;
    m->link=NULL; m->link=NULL;
    void display(node *p)
    { node *t;
      cout<<endl;
      t=p;
      if(t==NULL)
          cout<<endl<<"the list is empty ";
      else
      {
          while(t!=NULL)
          { cout<<t->data<<" ";
            t=t->link;
          }
      }
    }

    int main()
    { int item; int k,choice; node *q;
      choice=0;
      linklist l;
      do
      {
          cout<<endl<<"Representation of the linked list & its operations ";
          cout<<endl;
          cout<<endl<<" 1 - Create the linked list: Z ( of int type ) ";
          cout<<endl<<" 2 - split the linked list (Z) into two linked lists (X)&(Y) ";
          cout<<endl<<" 3 - merge two linked lists (X)&(Y) into one list (Z) ";
          cout<<endl<<" 4 - display the content of the linked list (Z) ";
          cout<<endl<<" 5 - display the content of the linked list (X) ";
          cout<<endl<<" 6 - display the content of the linked list (Y) ";
          cout<<endl<<" 7 - exit ";
          cout<<endl;
          cout<<endl<<" select your choice : ";
          cin>>choice;
          switch(choice)
          {
              case 1: l.create();
                break;
              case 2: l.split();
                start=NULL;
                cout<<endl<<" the linked list is splitted ";
                break;
              case 3: x=first; y=second;
                l.merge();
                cout<<endl<<" the two linked lists are emerged ";
                break;
              case 4: cout<<endl<<" the content of the linked list (Z) is : ";
                l.display(start);
          }
      }
    }

```

154


```

        break;
    case 2: cout<<" howmany elements you like to delete ?"<<endl;
        if (m>count)
            cout<<" the number of element(s) in the circular linked list is only "<<count;
        else
        {
            for (int j=1; j<=m; j++)
            {
                cl.delete(j);
                --count;
            }
            break;
        }
    case 3: if (count==0)
        cout<<" the circular linked list is empty ";
        else
        {
            cout<<" the content of the circular linked list is : "<<endl;
            cl.displaycl();
        }
        break;
    } // end of switch
    // end of DO
    while(choice!=4);
    return 0;
}

```

برنامج رقم - 17 : تنفيذ عملية نسخ عناصر الكس المتسلسل (Sequential Stack)
الطابور الوصول (Linked Queue).

```

#include <iostream>
using namespace std;
const int size=10; char stack[size];
struct node
{
    char data;
    node *link;
} *front, *rear;
int top=-1;
class CopySQ
{
public:
    char stack[size];
    node *p, *p2, *q, *q2;
    char item;
    void push(char item) // Adding an item to the stack
    {
        top++;
        if (top>size)
            stack[top]=item;
        else
        {
            cout<<" stack is full "<<endl;
            top--;
        }
    }
}

```

157

```

void clst::displaycl()
{
    node *q;
    if (r==NULL)
        cout<<" Display: the CL List is empty \n";
    else
    {
        q=r;
        do
        {
            cout<<((q->link->data)<<" ";
            q=q->link;
        } while(q!=r);
    }
}

void clst::deletecl()
{
    int item;
    if (r==NULL)
        cout<<" deletion: the CL List is empty ";
    else
    {
        node *f;
        f=r->link;
        item=f->data;
        if (f==r)
            r=NULL;
        else
            r->link=f->link;
        delete f;
    }
}

int main()
{
    int choice,m,j; int count=0;
    clst cl;
    do
    {
        cout<<" Representation of the CIRCULAR LINKED LIST & its operations ";
        cout<<"\n";
        cout<<" 1 - Add new elements ( int type ) to the circular linked list ";
        cout<<"\n";
        cout<<" 2 - Delete elements from the circular linked list ";
        cout<<"\n";
        cout<<" 3 - Display the content of the circular linked list ";
        cout<<"\n";
        cout<<" 4 - Exit ";
        cout<<"\n";
        cout<<" select your choice : ";
        cin>>choice;
        switch(choice)
        {
            case 1: cout<<" howmany elements you like to Add ? "<<endl;
                for(int j=1; j<=m; j++)
                {
                    cout<<" enter the data item : "; cin>>k;
                    cl.addcl(k);
                    ++count;
                }
            }
        } while(choice!=4);
    }
}

```

156

```

        cout<<"\n *** The stack's elements are copied to linked queue ***";
        break;
    case 4: if (front==NULL)
        cout<<"\n the linked Queue is empty ";
        else
        {
            cout<<"\n the elements of the linked queue are : ";
            t=front;
            while(t!=NULL)
            {
                cout<< "<data>" ;
                t=t->link;
            }
        }
        break;
    } // end of switch
    } // end of DO
    while(choice!=5);
    return 0;
} // end of main

```

159

```

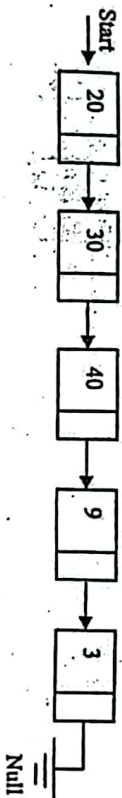
void copySTQL()
{
    char item; int i;
    ttop ; front=NULL ; rear=NULL ;
    while (t !=-1)
    {
        item=stack[i];
        --i;
        p=new node;
        p->data=item;
        p->link=NULL;
        if (rear==NULL)
            front=p ;
        else
            rear->link=p ;
        rear=p ;
    }
}

int main()
{
    char item1; int k,n,choice ; node *t,*first,*start;
    choice=0 ; char stack[size];
    front=NULL; rear=NULL ; ttop=-1 ;
    copySQ SQ;
    do
    {
        cout<<"\n Copying The Stack Elements To Linked Queue ";
        cout<<"\n ";
        cout<<"\n 1 - Creation of a of the STACK ( CHAR type ) ";
        cout<<"\n 2 - display the content of the sequential STACK ";
        cout<<"\n 3 - Copying stack elements to the linked queue ";
        cout<<"\n 4 - display the content of the linked queue ";
        cout<<"\n 5 - exit ";
        cout<<endl;
        cout<<"\n select your choice : ";
        cin>>choice;
        switch(choice)
        {
            case 1: cout<<"\n howmany items you like to push in the stack : ";
                    cin>>n;
                    for (int i=1; i<=n; i++)
                    {
                        cout<<"\n Enter the item : "; cin>>item1;
                        SQ.push(item1);
                    }
                    break;
            case 2: if (top==1)
                    cout<<"\n The stack is empty ";
                    else
                    {
                        cout<<"\n the content of the STACK is ";
                        for (int i=top; i>=0; i--)
                            cout<<"\n "<<SQ.stack[i] ;
                    }
                    break;
            case 3: SQ.copySTQL();
        }
    }
}

```

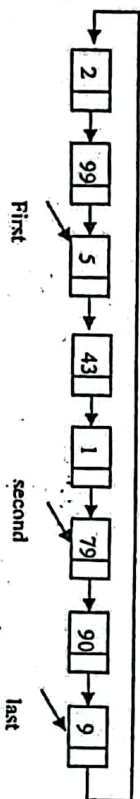
158

8. Write a segment of code of C++ program to delete all elements of the linked list except the first and last elements.
9. Given the following linked list:



Write a segment of Code of C++ program to :

- Increment the value of the fourth an fifth element by (50)
 - add an element of value (60) before the last element in the list
10. Give the output of the following code segments using the given linked list



- while (second != first)


```

      {
          cout<<(second->next);
          second= second->next;
      }
      
```
- temp=last->next;


```

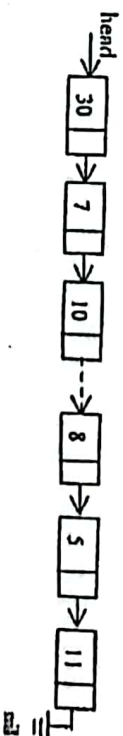
      while (temp !=second-> next)
      {
          cout<<(temp->next)
          temp=temp->next
      }
      
```
- while (first!= second)


```

      {
          cout<<(first->next);
          first= first->next;
      }
      
```

مسئلة الفصل

- what do we mean by Dynamic allocation of storage? Explain that in detail . what are its advantages and disadvantages?
- Compare between sequential and Dynamic allocation of storage .
- What do we mean by a linked list ? How we can declare it? Explain that in detail .
- Write a segment of code of C++ program for each of the following .
 - create a linked list of two nodes.
 - delete the first element in the linked list
 - insert one element after position p in a linked list .
 - insert one the elements before position p in a linked list.
 - delete all the elements where its data field is odd in any linked list .
 - delete the last element of any linked list
 - delete all the elements of even position in the linked list.
 - delete any zero value elements in the linked list .
 - count the number of elements in the linked list.
- Given the following linked list:

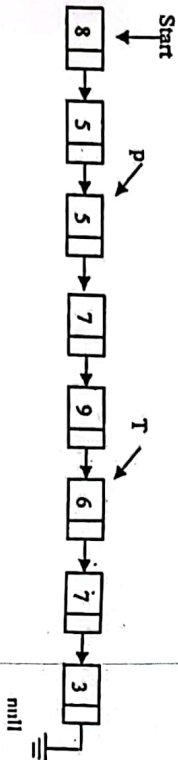


Write a segment of code of C++ program to perform the followings..

- Delete all the even numbers.
 - Add the elements 70,80,90,100 before the element in the list.
6. Let A=(A1, A2, ...A10) and B=(B1,B2,...B8) are two linked lists. Write a segment of code of C++ program to merge the first five elements of the list (B) at the end of the list (A).
7. Write a segment of C++ program to display all elements of any linked list except the element of data value (44).

{Note : Assume the linked list is pointed by X and its elements a1, a2,..., an should be inverted to become X=a_n, a_{n-1},..., a2, a1 }.

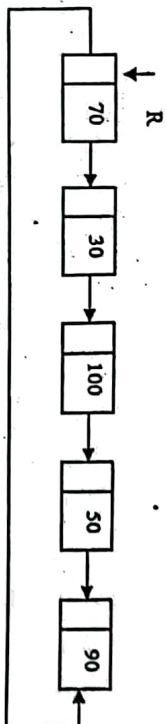
18. Write a segment of code of C++ to:
 - a. delete an element a nth position from a linked list pointed by r.
 - b. delete the element at nth position the last element of the linked list pointed by r.
 - c. delete the element after the node has information HH from the linked list pointed by r.
 - d. insert one node (element) before the last node in a linked list pointed by r.
19. Write a recursive function to printout the elements of the linked list in reverse order.
20. Given the following linked list.



Write a segment of code of C++ program for each of the following to be perform on the original . list.

- a. Insert the element (66) before the element (6)
 - b. Delete the element (8).
 - c. Insert the element (77) after position p.
 - d. Delete the element (3).
21. Give the declaration required and write a segment of code of C++ program to create a linked list of (N) students names.
 22. Let A be a linked list consist of (15) positive integers, write a segment of code of C++ program to split it into two lists, the first one consist of all the odd numbers and the second one consist of all the even numbers.

11. given the following list



Write a segment of code of C++ program to :-

- a. delete one element from it.
 - b. delete two elements from it.
 - c. add one element of data value(40).
 - d. display all the elements.
12. Write a segment of code of C++ program too split any given linked list such that the first will consist of all elements with data field < VALUE and the second will consist of all elements with data field >= VALUE.
 13. Show what is the output of the following segment of code:


```

list=new int ; p=new int ;
list->data= 20 ;
p->data=50 ;
list = p ;
p->data = 70 ;
cout <<(p->data, list->data) ;
      
```
 14. Write a function to display all the elements of the linked list
 15. Write a segment of code of C++ program to:
 - a. merge any two linked lists(X, Y)
 - b. merge any three linked lists (A,B,C)
 16. write a function to find the largest element in the linked list pointed by R.
 17. Write a function to invert any given linked list

الفصل الخامس

الاستدعاء الذاتي

Recursion

Introduction	مقدمة	1-5
Definition	تعريف الاستدعاء الذاتي	2-5
	معالجة برنامج الاستدعاء الذاتي	3-5
Processing of Recursive Subprograms		
When we use Recursion	متى نلجأ للاستدعاء الذاتي	4-5

23. Given a linked list consist of three elements A, B and C, let F is a pointer to the list head, what is the effect of each of the following segment of code,

a) $q = F \rightarrow \text{Next};$

$q \rightarrow \text{data} = M;$

b) $p = F \rightarrow \text{Next};$

$q = p \rightarrow \text{Next};$

$p \rightarrow \text{Next} = q \rightarrow \text{Next};$

Delete (q);

c) $q = F;$

$F = \text{new char};$

$F \rightarrow \text{data} = N;$

$F \rightarrow \text{Next} = q;$

24. Write a function to count the number of odd value elements in any linked list,

25. Write a function (search) to check the existence of a certain value in the linked list.

$$\begin{aligned}
5! &= 5*(5-1)! \\
&= 5*4! \\
&= 5*4*(4-1)! \\
&= 5*4*3! \\
&= 5*4*3*(3-1)! \\
&= 5*4*3*2! \\
&= 5*4*3*2*(2-1)! \\
&= 5*4*3*2*1! \\
&= 5*4*3*2*(1-1)! \\
&= 5*4*3*2*1*0! \\
&= 5*4*3*2*1*1
\end{aligned}$$

ملاحظة: $(0!=1)$ حسب التعريف الرياضي.

يلاحظ في هذا المثال أن التعريف يتكون من جزئين هما ،
الجزء الأول: ويسمى الأساس (Base Part) وهو يعطي القيمة الابتدائية.

$$n! = 1 \quad \text{if } n=0$$

الجزء الثاني: ويسمى الاستقرائي (Inductive Part) وهو يعطي القيم اللاحقة
 $n! = n * (n-1)!$ if $n > 0$

وهنا من خلال التطبيق المتكرر لهذا الجزء من التعريف منصل الى الحالة الابتدائية (الأساس) ويتوقف تنفيذ التعريف عندما تصبح $(n=0)$ لأن نتائج مضروب العدد $(0! = 1)$.

مثال 2:

لاحتساب دالة (X^m) power ، أي لاحتساب قيمة العدد (X) مرفوعا إلى القوة (m) .

هذه الدالة يمكن تعريفها باستخدام صيغة التكرار الذاتي:

$$(x^m) = \begin{cases} 1 & \text{if } m = 0 \\ x * (x^{m-1}) & \text{if } m > 0 \end{cases}$$

الجزء الأساسي الجزء الاستقرائي

1-5 مقدمة

عادة يتم تعريف المصطلحات أو المفاهيم (خصوصا مفاهيم الرياضيات) باستخدام مفاهيم أبسط منها أو اشياء معروفة أصلا ، إلا أنه لحتاج أحيانا الى تعريف بعض المفاهيم باستخدام المفهوم نفسه (define themselves) ، وهذه الصيغة في التعريف تسمى التعريف بالتكرار الذاتي (Recursive Definition).

فمثلا في تعريف المجموعات اللانهائية (Infinite Sets) يعتمد فكر جميع عناصر هذه المجموعة ، فمجموعة العناصر الطبيعية تعرف كالآتي:

$$N = \begin{cases} 0 & \text{if } n=0 \\ n+1 & \text{if } n>0 \end{cases}$$

2-5 تعريف الاستدعاء الذاتي

هو قابلية البرنامج للرجعي (function أو procedure) لاستدعاء نفسه، أنه مفهوم رياضي وطريقة برمجة قيمة ، وأساليب برمجي فعال إذ يمكن استخدامه بدلا من استعمال التكرار (iteration).

هناك العديد من الصيغ الرياضية يمكن التعبير عنها باستخدام الاستدعاء الذاتي.

مثال 1:

لاحتساب دالة مضروب العدد $(n!)$ [factorial of n]

فإنه يعرف رياضيا كالآتي:

$$n! = \begin{cases} 1 & \text{if } n=0 \\ n*(n-1)! & \text{if } n>0 \end{cases}$$

الجزء الأساس الجزء الاستقرائي

نجد أن $(n!)$ تعرف بواسطة $(n-1)!$ أي تعريف نفسها بصورة متكررة ذاتيا وتعني: مضروب العدد = العدد * مضروب العدد الذي يسبقه
أي أن إيجاد مضروب العدد (n) يتطلب تكرار الدالة (n) من المرات باعتماد تعريف الدالة لنفسه في كل مرة ، كما في حساب مضروب العدد $(5!)$.

ب. الأيعازات البرمجية لإدالة مضروب العدد (n!) باستخدام الاستدعاء الذاتي :

```
int fact( int n)
{
    if (n==0)
        return 1;
    else return n * fact(n-1);
}
```

ستتم هذا الاستدعاء بتلقيص متدرج لقيمة العدد إلى 1 أن يصل إلى 1

fact(1)

(n-1)*fact(n-2)
(n-2)*fact(n-3)
(n-3)*fact(n-4)

ان استدعاء الدالة fact (n) ستتم على العدد (n) وعند تنفيذ الجملة الأخيرة (else) فان الدالة ستدعي نفسها على العدد (n-1) ايضا .

وفي هذه المرة ستتم تنفيذ نفس الجملة باستخدام الدالة نفسها ولكن على العدد (n-2) ، ثم لاحقاً على العدد (n-3) وهكذا بالتتابع التراجعي . ويتوقف تنفيذ هذه الجملة حين يصل الاستدعاء إلى fact(1) إذ تحسب النتيجة تراكمياً ويمكن تراجعي ابتداءً من هذه الخطوة ثم الخطوات التي تسبقها ولغاية الحصول على النتيجة النهائية .

فلو نفذنا هذه الدالة على العدد (n = 4) فان الجملة الأخيرة (else) ستتم بصيغة تكرارية كما يلي :

```
fact = 4 * fact (4-1)
      = 4*3*fact (3-1)
      = 4*3*2*fact (2-1)
      = 4*3*2*1 = 1! = 1
```

النتيجة النهائية 24

169

ولو أخذنا أمثال العددي (2⁴) مثلاً :

$$\begin{aligned}
 2^4 &= 2 * 2^{4-1} \\
 &= 2 * 2^3 \\
 &= 2 * 2 * 2^{3-1} \\
 &= 2 * 2 * 2^2 \\
 &= 2 * 2 * 2 * 2^{2-1} \\
 &= 2 * 2 * 2 * 2 * 2^{2-1} \\
 &= 2 * 2 * 2 * 2 * 2^{2-1} \\
 &= 2 * 2 * 2 * 2 * 2^0
 \end{aligned}$$

مثل هذه العمليات يمكن برمجتها حاسوبياً باستخدام ما يعرف بالاستدعاء الذاتي (Recursion) كما يمكن برمجتها بالصيغة الاعتيادية دون استخدام الاستدعاء الذاتي فلو أخذنا المثال الأول عن دالة مضروب العدد (n!) فإتينا يمكن أن نكتبها برمجياً بصيغة التكرار (iteration) وكذلك صيغة الاستدعاء الذاتي (Recursion) .

أ. الأيعازات البرمجية لإدالة مضروب العدد (n!) بدون استخدام الاستدعاء الذاتي (أي باستخدام التكرار)

```
int fact(int n)
{
    int prod=1;
    for(int i=1; i<=n; i++)
        prod:=prod*i;
    return prod;
}
```

لاحظ استخدام صيغة التكرار (looping structure)

168

3. وفي الاستدعاء اللاحق للعدد التالي (n-1) أي fact(2) سيخزن في المكس عنوان الرجوع لهذا الاستدعاء (Rt3) والعدد (2).

← Top = 5	
2	Rt3
3	
Rt2	
4	
Rt1	

4. الاستدعاء اللاحق يكون للعدد التالي (n-1) أي fact(1) وسيخزن في المكس عنوان الرجوع لهذا الاستدعاء (Rt4) والعدد (1).

← Top = 7	
1	Rt4
2	
Rt3	
3	
Rt2	
4	
Rt1	

5. وعند تنفيذ البرنامج الفرعي على العدد (1) مستند بداية الجملة الشرطية (return 1 ;) if (n==0) لان (n<2) لعملي النتيجة (fact=1) وهي حالة توقف التكرار (Base Case).
6. يتم إخراج قيمة المتغير (pop) قيمة المتغير (n=1) وإخراج عنوان الرجوع (Rt4) ثم إدخال نتيجة الاحتساب وهي (1).
7. إخراج (pop) النتيجة الأخيرة وهي (1) وكذلك المتغير (n=2) مع عنوان الرجوع (Rt3) لاحتساب نتيجة الدالة وهي (2) ثم إدخال (push) هذه النتيجة في المكس.
8. إخراج (pop) النتيجة الأخيرة وهي (2) وكذلك المتغير (3) مع عنوان الرجوع (Rt2) لاحتساب نتيجة الدالة وهي 2*3=6 ثم إدخال (push) هذه النتيجة في المكس.

3-5 معالجة برامج الاستدعاء الذاتي

لمعرفة متى يكون الاستدعاء الذاتي متوقفا علينا أن نفهم أولا كيفية معالجة لغات البرمجة للبرامج التي تحتوي برامجه فرعية بصيغة الاستدعاء الذاتي . إذ يستخدم لهذا الغرض المكس (stack) وفق الآتي:

فبعد كل استدعاء للبرنامج الفرعي (recursive procedure) أو (recursive function) يتم تخزين عنوان الرجوع (push) عنوان الرجوع (return address) في المكس مع نسخ قيم المعالم (parameters) لذلك الاستدعاء ويكرر هذا عند كل استدعاء للبرنامج الفرعي لحين الوصول إلى حالة الأساس (base case) في التعريف حيث تبدأ العملية المعكوسة وهي إخراج (pop) محتويات المكس بالتتابع والوصول إلى النتيجة النهائية.

لنأخذ البرنامج الفرعي fact(n) ونتابع حالة المكس عند تنفيذ احتساب متسروب العدد (4) أي [fact(4)].

1. عند أول استدعاء للبرنامج الفرعي fact(4) سيخزن في المكس كل من عنوان الرجوع (Rt1) والعدد (4).

← Top = 1	
4	Rt1

2. في جملة else مستندى الدالة على العدد التالي (n-1) أي fact(3) وعليه يخزن في المكس عنوان الرجوع لهذا الاستدعاء (Rt2) والعدد (3).

← Top = 3	
3	
Rt2	
4	
Rt1	

١. تعريف الدالة يكون استخدام الاستدعاء الذاتي :

double power (double x, unsigned m)

```
{
    double p=1;
    if ( m>0 )
```

```
{
    for( int i=1; i< m; i++)
        p = p*x;
```

```
}
    return p;
}
```

ب. تعريف الدالة بإستخدام الاستدعاء الذاتي :

double power (double x, unsigned m)

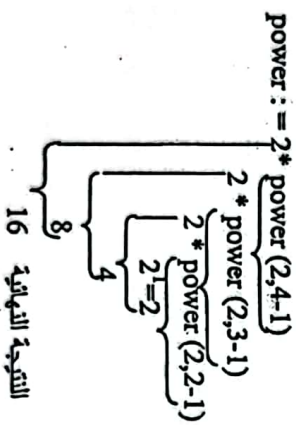
```
{
    if ( m == 1 )
```

```
    p=x;
```

```
    else p=x * power(x,m-1);
```

```
    return p;
}
```

وعند احساب قيمة الدالة (2^4) فإن الجلة (Else) ستند الصيغة التالية:



9. إخراج (pop) النتيجة الأخيرة وهي (6) وكذلك كل من المتغير (4) مع عنوان الرجوع (Ret) لاحتساب الدالة وهي $6 * 4 = 24$ ثم إدخال (push) هذه النتيجة في المكس.

10. إن القيمة الوحيدة المتبقية في المكس هي النتيجة النهائية لعملية احتساب مضروب العدد وهي 24 لأن $1 * 2 * 3 * 4 = 24$

بسبب أسلوب المعالجة اعلاه نجد ان الاستدعاء الذاتي يتطلب مساحة خازنية اكبر (لايه) يكرر خزن نفسه من قيم المتغيرات وعنوان الرجوع بعد كل استدعاء (كما ان هذه المعالجة تستغرق وقتا اطول ، الا اننا نحصل على زيادة وضوح خازنية الحل وبساطته (simplicity & clarity) بما يساعد على سهولة الصيانة والتدقيق (verification & maintainability).

في السنوات السابقة كانت كثرة أجهزة ومعدات الحاسوب عالية بما فيها الذاكرة، كما ان سرعة المعالجة اقل، وعليه فإن المبرمج يعمل على كتابة خوارزميات تستخدم مساحة خازنية اقل وسرعة تنفيذ اكبر ويكون هذا على حساب درجة الوضوح والبساطة اما في الوقت الحاضر قد انخفضت بشكل حاد كثرة الأجهزة والمعدات بما فيها الذاكرة وازدادت سرعة المعالجة ببرجيات كبيرة جدا . بحيث اصبح هذا العامل غير مؤثر إلا انه في الوقت نفسه ازددت كثرة الأيدي العاملة المستخدمة لهذه الموارد وخصوصا إحداد البرامج وصيولتها ومتابعيتها إلى حدود تجارزيت فرق انخفاض كثرة الأجهزة ، ولينما يمكن الاستنتاج إن اللجوء إلى الاستدعاء الذاتي لازال ذا أهمية وله مبرراته لكونه يستغرق وقت إحداد اقصر من أسلوب التكرار (iteration).

وعند إحداد برنامج فرعي (procedure أو function) بصيغة الاستدعاء الذاتي يتطلب مراعاة ما يلي:

- أن يحتوي البرنامج الفرعي حالة الأساس (base case) وهي حالة توقف التكرار، أي انتهاء عمل البرنامج ، كما في برنامج حساب مضروب العدد . { if (n == 0) return }
- إن تنفيذ خطوات البرنامج الفرعي يؤدي إلى اقتراب الحل من الوصول إلى حالة (base case).
- ولو عشنا إلى المثال التالي فيمكن كتابة البرنامج الفرعي الدالة (power) كما يلي:

تبرين: سلسلة اعداد (fibonacci) اي عدد فيها يكون مساويا لمجموع العددين

الذين يسبقه جدا العددين الاول = 0 والثاني = 1 ،

لذا فان سلسلة الاعداد هي :-

....., 5, 34, 21, 13, 8, 5, 3, 2, 1, 1, 0

اكتب دالة استدعاء ذاتي لاستخراج العدد الترتيبي (n-th) في سلسلة اعداد

(fibonacci) (علما ان الحد الاول 0 في المربع 0 (n = 0

الحل:

```
unsigned int fib ( unsigned int n ) ;
```

```
{
```

```
if ( n < 2 ) return n ;
```

```
return fib ( n - 2 ) + fib ( n - 1 ) ;
```

```
}
```

تبرين: اكتب برنامجا فرعيا بصيغة الاستدعاء الذاتي لقراءة الرمز (space)

وتجوزة (امثله)

الحل:

```
void skipspaces() ;
```

```
{
```

```
char ch ;
```

```
cin >> ch ;
```

```
if ( ch == ' ' )
```

```
skipspaces() ;
```

```
}
```

4-5 متى نلجا للاستدعاء الذاتي

When we use Recursion

1- استخدام الاستدعاء الذاتي ونفضل في العمليات التي يمكن تعريفها بصيغة

التكرار الذاتي (تعرف فيها نفسها)

2- استخدام الاستدعاء الذاتي يوفر الوقت والجهد للمبرمج عند الاعداد

3- إن تنفيذ الحل بدون الاستدعاء الذاتي يعد اكثر كفاءة من حيث المساحة الخازنية

ورقت التنفيذ ، اما الاستدعاء الذاتي فيطلب وقت تنفيذ أطول ومساحة خازنية

اكثر بسبب الاستدعاء المتكرر والتعامل مع قيم متعددة للمتغيرات

4- بصورة عامة ونفضل الحل بدون الاستدعاء الذاتي إذا كان الحل قصيرا وبسيطا

5- من التطبيقات المهمة التي يستخدم فيها الاستدعاء الذاتي هي خوارزميات

(searching), (sorting), (Tree Traversal)

6- في الاستدعاء الذاتي نستخدم صيغ وعبارات التفرع

و (If-then-Else) بدلا من صيغ التكرار (For-Do)

7- مع أهمية الاختيار الجيد للبرنامج وتطبيق قيم المتغيرات

فيه قبل تنفيذه على البيانات الحقيقية .

تبرين: اكتب دالة استدعاء ذاتي لحساب القاسم المشترك الأعظم

Greatest Common Divisor لأي عددين صحيحين موجبين (m, n) .

الحل الثاني:

```
int gcd (int m, int n)
```

```
{ int r, g ;
```

```
r = (m % n) ;
```

```
if ( r == 0 )
```

```
g = n ;
```

```
else g = gcd ( n, r ) ;
```

```
return g ;
```

```
}
```

الحل الأول:

```
int gcd(int m, int n)
```

```
{ if ( n == 0 ) return m ;
```

```
return gcd ( n, m % n ) ;
```

```
}
```

برامج الفصل

برئاسه رقم - 18 : احتسابی دالہ مضروب العدد (π) باستخدام كل من صیغة

التكرار (iteration) وصيغة الاستدعاء الذاتي (recursion)

```
#include <iostream>
using namespace std;
```

```
// This program using the iteration & recursion in computing factorial of any number //
int factiteration(int n)
```

```
( int prod=1;
for(int i=1; i<=n; i++)
    prod=prod*i;
```

```

prod = prod * x;
return prod;
}

```

```
int factrecursion(int n)
```

```

- If ( $n=0$ )
  return 1;

```

```

else
    return n*factrecursion(n-1);

```

int main()

IntXY:
 counts the times the iteration to compute the Exponential of an Interval X is

```
cout<<"\n enter the number: ";
cin>>x;
```

```
cout<<"\n the result is : "<<factorial(x);
cout<<"\n *****"
```

```
cout<<"\n Using the Recursion to compute the Factorial of any Integer" ;
cout<<"\n enter the number:" ;
```

```
cout<<"\n the result is : "<<factrecursion(y);
```

```
return 0;
```

احتساب دالة (power) لعدد ما مثل (X^n) باستخدام برنامج رقم - 19:

صيغة الاستدعاء الذاتي (recursion).

```
#include <iostream>
using namespace std;
```

```
int power(int x, int y)
{
    if (y == 0)
    {
        return 1;
    }
    return x * power(x, y - 1);
}
```

```

    if (n==0)
        return 1;
    return x*power(x,n-1);
}

```

)

Int y,m;

```
cout<<"\n Compute the value of ( Y power n ) using Recursion\n";
cout<<"\n enter the base number :";
```

```
cin>>y;
cout<<"Enter the exponent: ";
```

تعرين :- اكتب دالة بصفة الاستعاء الذاتي (Recursive Function)
 لاحساب مجموع مربعات عناصر القائمة الموصولة (Start).

```
int sum( node *p )
```

{ 1111 }

```
return 0 ;
```

```
return ((p->data) * (p->data) + sum(p->link));
```

—

تعرين: اكتب برنامج فرعي (function) لقراءة جملة (string) معينة

وعرضها (أي طباعتها) بصورة معكوسة، أي تقراً (ABC) وتعرض

॥२॥

```
void reverse()
```

{

```
char ch ;
```

```

    If( chi='n' )
      cil.gel(cil) ,

```

```
{ reverse() ; cout.put(ch) ; }
```

**برنامج رقم 21 : لاستخراج العدد الثنائي (n-th) في سلسلة اعداد
فبوناتشي (fibonacci) بصيغة التكرار (iteration) وصيغة الاستدعاء
الذاتي (recursion).**

```
#include <iostream>
using namespace std;
class testif
{
public:
void fib1(int n)
{ int fnum1,fnum2,fni;
if (n<=1)
cout<<n;
else
{
fnum2=0; fnum1=1;
for(i=2; i<=n; i++)
{
fn=fnum1+fnum2;
fnum2=fnum1;
fnum1=fn;
}
cout<<fn;
}
}
int fib2(int n)
{
if (n<=2)
return n;
else
return fib2(n-2)+fib2(n-1);
}
int main()
{
int x;
testif t;
cout<<endl<<"Find out the value of the fibonaci number using Iteration & recursion";
cout<<endl<<"enter the position of the fibonaci number starting from position (0) : ";
cin>>x;
cout<<endl<<"The value of the Fibonacci number using iteration is : ";
t.fib1(x);
cout<<endl<<".....";
cout<<endl<<"The value of the fibonaci number using recursion is : ";
cout<<endl<<t.fib2(x);
return 0;
}
```

179

**برنامج رقم 20 : لأحسب قيمة القاسم المشترك الأعظم (greatest
common divisor) لأي عددين صحيحين موجبين مثل (m, n) باستخدام
صيغة الاستدعاء الذاتي (recursion).**

```
#include <iostream>
using namespace std;
int gcd(int m,int n)
{ int r;
r=(m%n);
if (r==0)
return n;
else
return gcd(n,r);
}
int main()
{
int x,y;
cout<<endl<<"compute the greatest common divisor ( gcd ) of any two positive integers";
cout<<endl<<"enter the first number : ";
cin>>x;
cout<<endl<<"enter the second number : ";
cin>>y;
cout<<endl<<"the greatest common divisor is : "<<gcd(x,y);
return 0;
}
```

178


```

cout<<"\n 4-exit\n";
cout<<endl;
cout<<"\n select your choice : ";
cin>>choice;
switch(choice)
{
case 1: create();
break;
case 2: if (start==NULL)
cout<<"\n The linked list is Empty\n";
else
{
cout<<"\n the elements of the linked list are : ";
test();
while(it!=NULL)
{
cout<<"<data><<" ;
it->link;
}
}
break;
case 3: cout<<"\n the sum of squares of the list elements is : ";
cout<<sum(start);
break;
}
} // end of switch
} // end of DO
while(choice!=4);
return 0;
} // end of main

```

برنامج رقم - 23: لتحديد ألية جملة (string) إن كانت من نوع (palindrome) ، أي يمكن قراءتها من الأتجاهين من اليمين ومن اليسار
 أيضا "] ، يتم اتباع الخطوات التالية :
 + مقارنة الحرف الأول من اليسار مع الحرف الأخير في اليمين .
 + إذا كان الحرفان متشابهين فيتم تركهما و التحرك بموقع واحد من كل جهة الى الداخل .
 + تكرر هذه الخطوات لغاية بقاء حرف واحد أو لا يبقى شيئاً "

مثال : الجملتان (xyzazyx) ، (xyzazyx) هما من نوع (palindrome) (recursion) ، والبرنامج التالي ينفذ ذلك باستخدام صيغة الاستدعاء الذاتي

```

#include <iostream>
using namespace std;
bool ispalindrome(string s)
{
int strlen=s.size();
if(strlen==1)
return true;
else

```

برنامج رقم - 22: لأختساب مجموع مربعات عناصر القائمة الموصولة (linked list) باستخدام صيغة الاستدعاء الذاتي (recursion) .

```

#include <iostream>
using namespace std;
struct node
{
int data;
node *link;
}*start;
class linkedlist
{
private:
node *p,*p2,*q,*r;
int item1,item2;
public:
linkedlist()
{
start=NULL;
}
void create()
{
int n;
p=new node;
start=p;
cout<<"\n how many elements you like to enter ? "; cin>>n;
for(int i=1; i<=n; i++)
{
cout<<"\n enter the new element : ";
cin>>p->data;
if(i!=n)
{
p2=new node;
p->link=p2;
p=p2;
}
}
int sum(node *p)
{
if (p==NULL)
return 0;
return ((p->data)*(p->data)+sum(p->link));
}
}
int main()
{
linkedlist l;
l.create();
l.test();
}

```

Creation of a linked list & compute the sum of the squares of its elements :-
 1- Creation of a linked list
 2- display the content of the linked list
 3- compute the sum of squares of the list elements "

```

#include <iostream>
using namespace std;
void reverseNum1(int n) // For counting-up any number
{
    if (n<0)
        return;
    reverseNum1(n-1);
    cout<<n<<" ";
    return;
}
void reverseNum2(int n) // For counting-down any number
{
    if (n<0)
        return;
    cout<<n<<" ";
    reverseNum2(n-1);
    return;
}
int main()
{
    int x;
    cout<<"\n enter the number you like to count-up : ";
    cin>> x;
    cout<<"\n the count up of this number is : ";
    reverseNum1(x);
    cout<<"\n *****";
    cout<<"\n enter the number you like to count-down : ";
    cin>> x;
    cout<<"\n the count down of this number is : ";
    reverseNum2(x);
    return 0;
}

#include <iostream>
using namespace std;
void reverseStr()
{
    char ch;
    cin.get(ch);
    if (ch!='\n')
        reverseStr();
    cout.put(ch);
}
int main()
{
    cout<<"\n enter the string : ";
    reverseStr();
    return 0;
}

```

برنامج رقم -26: لقراءة أية جملة (String) ثم يتم قلب (Reverse) ترتيب حروفها بشكل م عكوس وذلك باستخدام صيغة الاستدعاء الذاتي (recursion).

183

```

if(strlen<1)
    return true;
else
    return !palindrom(x.substr(1,strlen-2));
}
int main()
{
    string str;
    cout<<"\n enter any string : ";
    cin>> str;
    if(!palindrom(str))
        cout<<"\n YES, the string is palindrom ";
    else
        cout<<"\n NO, the string is NOT palindrom ";
    return 0;
}

#include <iostream>
using namespace std;
int isPrime(int p,int i=2)
{
    if (i==p)
        return 1;
    if (p%i==0)
        return 0;
    return isPrime(p,i+1);
}
int main()
{
    int y;
    cout<<"\n enter the number you like to test it if its PRIME : ";
    cin>> y;
    if(isPrime(y,2))
        cout<<"\n the number is prime ";
    else
        cout<<"\n the number is NOT prime ";
    return 0;
}

```

برنامج رقم -24: لا يقبل أي عدد صحيح موجب إن كان عدداً أولياً (prime) أم لا ، باستخدام صيغة الاستدعاء الذاتي (recursion).

182

برنامج رقم -25: لقراءة عدد صحيح موجب مثل (n) ، وتنفذ عملية الحد التصاعدي (counting-up) لأي أبعاد الأعداد من 0 ، 1 ، 2 ، ... ، n ، وكذلك الحد التنازلي (counting-down) لأي أبعاد الأعداد من n ، n-1 ، ... ، 0 ، باستخدام صيغة الاستدعاء الذاتي (recursion).

الفصل السادس

Tree structure هيكل الشجرة

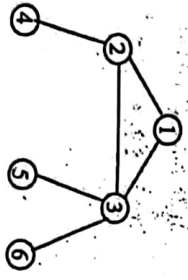
Graph	المخطط	1-6
Tree Structure	هيكل الشجرة	2-6
Binary Tree	الشجرة الثنائية	3-6
Other Types of Trees	أنواع أخرى من الأشجار	4-6
Tree Traversing	استعراض عقد الشجرة	5-6
Tree Representation	تمثيل الأشجار	6-6
- General Tree	- الشجرة الاعتيادية	
- Binary Tree	- الشجرة الثنائية	
7-6	تمثيل التعبيرات الحسابية باستخدام الشجرة الثنائية	
Representation of Arithmetic Expressions Using Binary Tree		
8-6	تحويل الشجرة الاعتيادية الى شجرة ثنائية	
Tree Transformation		
Binary Search Tree	شجرة البحث الثنائية	9-6

اسئلة الفصل

- 1- Write a recursive function that is given a string S and returns the reversal of S . For example , given S = A B C D , the function should return the string "D C B A " .
- 2 - Write a program to check if string (S) is the inverse of string (R) using stack
- 3 - What are the fundamental differences between recursion and iteration techniques .
- 4 - Write a recursive function to search a list of nodes (names) and find the string " Target "

1-6 المخطط Graph

هو عبارة عن مجموعة من العناصر (V) تمثل بنقاط (رؤوس) تسمى (Vertices) ومفردها (Vertex) وهذه العناصر تربطها علاقات (E) تمثل بخطوط تسمى حواف (edges) ومفردها (edge) أي أن المخطط $G = (V, E)$ هو مجموعة من العناصر والعلاقات وفق الشكل التالي :-



الشكل (1 - 6)

$$\begin{aligned} \text{العناصر} & \quad \{ 6, 5, 4, 3, 2, 1 \} = V(G) \\ \text{العلاقات} & \quad \{ (2,3), (1,3), (1,2), (3,6), (3,5), (2,4) \} = E(G) \end{aligned}$$

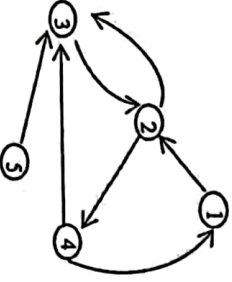
والمخططات على نوعين هما :

أ / المخطط غير المتجه undirected graph

هو المخطط الذي تكون العلاقة بين عناصره (رؤوسه) غير مرتبة (unordered) أي أن الاتجاه غير مهم في تلك العلاقة فمثلا الحالة (1,2) هي نفسها (2,1).

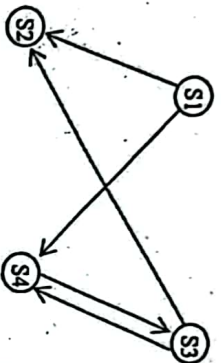
ب / المخطط المتجه directed graph

هو المخطط الذي تكون العلاقة بين عناصره (رؤوسه) مرتبة بنمط معين (ordered) أي أن الاتجاه مهم في تحديد تلك العلاقة فمثلا الحالة (1,2) تختلف عن (2,1) وتمثل هذه العلاقة بوضع سهم في مقدمة الخط ليوضح الاتجاه فالشكل (2-6) يبين أن هناك علاقة بين (3,2) ممثلة بمستقيم أي أن اتجاه العلاقة هي من (3 ← 2) وهناك علاقة أخرى تختلف عنها هي (2,3) ممثلة بمستقيم آخر ويعني أن العلاقة من (2 ← 3).



الشكل (2 - 6)

نأخذ المخطط التالي :-



الشكل (6 - 3)

هذا المخطط هو مخطط منجه يتكون من (4) تقاطع (رؤوس Vertices) (S_1, S_2, S_3, S_4) وخمسة خطوط (حافات - edges) ويمثل في مصفوفة مربعة درجتها (4) وعناصرها $(S_{i,j})$ حيث (i) يمثل نقطة البداية و (j) نقطة النهاية ففي حالة وجود خط (حافة) بين التقاطعين يمثل الموقع بالقيمة (1) وبعكسه يمثل بالقيمة (0).

فالمورد العامة للمصفوفة ستكون كالآتي :-

S:

	1	2	3	4
1	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	$S_{1,4}$
2	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$	$S_{2,4}$
3	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$	$S_{3,4}$
4	$S_{4,1}$	$S_{4,2}$	$S_{4,3}$	$S_{4,4}$

وعند تمثيل القيم لكل مسار ستصبح بالشكل التالي :

S:

	1	2	3	4
1	0	1	0	1
2	0	0	0	0
3	0	1	0	1
4	0	0	1	0

تقاطع البداية

فمثلا لو كان المخطط اعلاه يمثل طرق المواصلات بين مجموعة المدن

1, 2, 3, 4, 5 فيمكن ان نقول ان هناك طريق من المدينة (1) $\leftarrow$ (2) باتجاه واحد ولا يسمح باستخدامه من المدينة (2) $\leftarrow$ الى المدينة (1) ، ولكن هناك طريق من المدينة (2) الى المدينة (3) $\leftarrow$ (2) ويسمح باستخدامه باتجاه مكموس من المدينة (3) $\leftarrow$ (2).

المسار Path

هو مجموعة المستقيمات (الخطوط) التي توصل بين أي نقطتين في المخطط فين النقطتين 1, 5 في الشكل (6 - 1) يكون المسار هو (1, 3), (3, 5).

طول المسار Path Length

يقصد به عدد المستقيمات (الخطوط) التي تربط او تفصل بين أي نقطتين في المخطط فمثلا :

بين النقطتين 2, 6 طول المسار = 2 وهما (3,2), (6,3).

بين النقطتين 1, 6 يوجد مساران هما (2,1), (3,2), (6,3) وطوله 3 ومسار اخر هو (3,1), (6,3) وطوله 2.

المخطط المتصل Connected Graph

هو المخطط الذي توجد فيه مسارات بين أي نقطتين من تقاطع المخطط.

المخطط غير المتصل Unconnected Graph

هو المخطط الذي تكون بعض تقاطعه غير متصلة بمسار بينها.

1-6 تمثيل المخطط Graph Representation

ان اختيار طريقة تمثيل المخطط يعتمد على نوع التطبيق المطلوب الجاه وطبيعة وظائفه وسنوضح هنا طريقتين منها هما :-

أ- استخدام مصفوفة المجاورات Adjacency Matrix

يمثل المخطط بمصفوفة مربعة درجتها مساوية لعدد رؤوس (تقاطع) المخطط (No. of Vertices) ، فلذا كان عدد الرؤوس (3) فان المصفوفة تكون بالبعد $(3 * 3)$ ، اما اذا كان عدد الرؤوس (7) فان المصفوفة يجب ان تكون بالبعد $(7 * 7)$ وهكذا بالنسبة للمخططات الاخرى.

يتكون من (5) نقاط (Vertices) هي T_1, T_2, T_3, T_4, T_5 وسنة خطوط (حافات Edges) يمثل في مصفوفة مربعة لدرجتها (5) وتكون قيمة الموقع (1) في حالة وجود خط بين النقطتين بغض النظر عن الاتجاه .

تكون المصفوفة كما في الشكل ويوضح فيها :

T :

	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	0	0
3	1	1	0	0	1
4	0	0	0	0	1
5	1	0	1	1	0

ان الخط الموجود من (T_1) الى (T_2) ممثل بالموقع $T(1, 2)$ وقيمته (1) وهو نفس الخط الموجود من (T_2) الى (T_1) وممثل بالموقع $T(2, 1)$ وقيمته (1) ايضا وهكذا بالنسبة للخطوط الاخرى بين اية نقطتين .

ان هذه المصفوفة تصف بما يأتي :-

- * متناظرة حول المحور (المثلث الاعلى يناظر المثلث الاسفل) ولهذا يمكن اختصار نصف المساحة الخلفية وذلك بمثل احد المثلثين فقط .
- * ان مجموع القيم في كل صف (Row) تعطي (تمثل) عدد الخطوط الخارجة من كل نقطة (out degree) من كل نقطة فالصف الرابع ($i = 4$) مثلا يكون مجموع القيم فيه هو (1) لان النقطة الرابعة (T_4) يخرج منها خط واحد الى النقطة (T_5) .

ب - استخدام القوائم المجاورة Adjacency lists

تستخدم القائمة المتصلة (Linked list) في تمثيل المخطط اذ ان كل عقدة من عقد المخطط تمثل بقائمة متصلة تحتوي اسماء العقد التي تتصل بها فعناصر (عقد) القائمة الموصولة (i) هي الرؤوس (Vertices) المجاورة للعقدة (i) ، علما ان العقدة الواحدة تتألف من جزئين (اسوة بعناصر الهياكل الموصولة) جزء يحتوي دليل الرأس (index of the vertex) والجزء الاخر هو (Link) مؤشر الى موقع العقدة التالية .

لناخذ المخطط غير المتجه في الشكل الاتي :-

وهذه المصفوفة تكون حالة المخطط اذ منها يوضح :-

- وجود خط (حالة) من S_1 الى S_2 ($S_2 \leftarrow S_1$)
- وجود خط (حالة) من S_1 الى S_4 ($S_4 \leftarrow S_1$)
- وجود خط (حالة) من S_3 الى S_2 ($S_2 \leftarrow S_3$)
- وجود خط (حالة) من S_3 الى S_4 ($S_4 \leftarrow S_3$)
- وجود خط (حالة) من S_4 الى S_3 ($S_3 \leftarrow S_4$)

- لا يوجد خط من S_1 الى S_3

- لا يوجد خط من S_2 الى اية نقطة اخرى

- لا يوجد خط من S_3 الى S_1

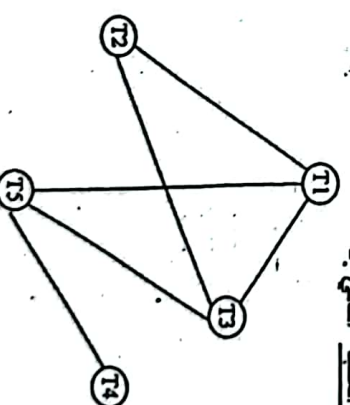
- لا يوجد خط من S_4 الى S_1 او S_2 .

ان المصفوفة التي تمثل المخطط المتجه تصف بما يأتي :

* غير متناظرة

- * مجموع القيم في كل صف (Row) تعطي (تمثل) عدد الخطوط الخارجة من كل نقطة (out degree) من كل نقطة ، فالصف الثالث ($i = 3$) مثلا يكون مجموع القيم فيه هو (2) لان النقطة الثالثة (S_3) يخرج منها خطان الى كل من (S_2, S_4) .
- * مجموع القيم في كل عمود (column) تعطي (تمثل) عدد الخطوط الداخلة الى كل نقطة ، فالعمود الرابع ($j = 4$) مثلا يكون مجموع القيم فيه هو (2) لان النقطة الرابعة (S_4) يدخل اليها خطان من (S_1, S_3) .

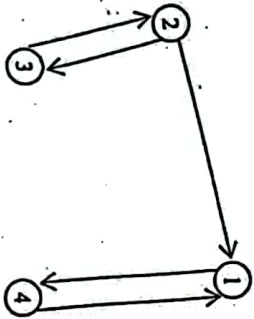
اما المخطط غير المتجه التالي :-



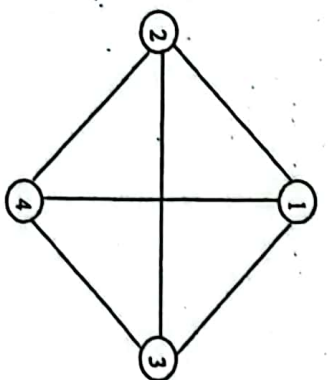
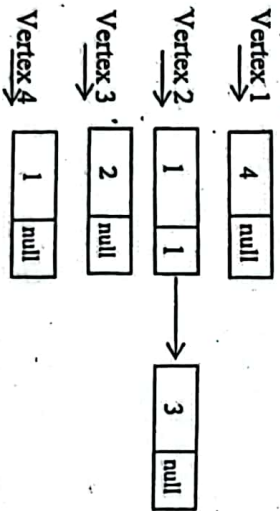
الشكل (4 - 6)

نلاحظ هنا بالنسبة للمخطط غير المتجه المكون من (n) من الرؤوس (vertices) و (e) من الحافات (edges) فإن تمثيله يتطلب $(2 * e)$ من العقد ومصفوفة سعتها (n) لحزن المؤشرات الرئيسية التي تشير إلى بداية كل قائمة ، وفي المثال اعلاه نجد اننا نحتاج إلى (12) عقدة لان عدد الحافات هو $(6 = e)$ بالإضافة إلى مصفوفة سعتها (4) بقدر عدد الرؤوس .

اما تمثيل المخطط المتجه في الشكل (6 - 7) فنلاحظ ان مجموع العقد في التوائم هو بقدر عدد الحافات في المخطط وهي (5) .

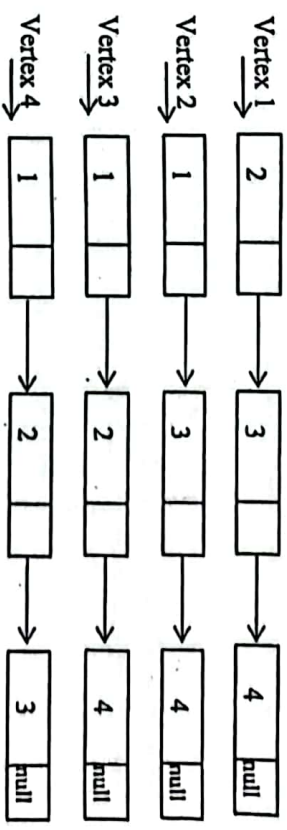


الشكل (6 - 7)



الشكل (5 - 6)

يكون تمثيل هذا المخطط بتوائم متجاورة كل منها لها مؤشر رئيسي يشير إلى بدايتها مثل vertex 1 ، vertex 2 ، الخ وكما في الشكل الآتي :-



الشكل (6 - 6)

ولاستكمال عملية التمثيل هذه فإن المؤشرات التي تشير إلى بداية كل قائمة نخزن في مصفوفة احادية سعتها بقدر التوائم او الرؤوس (n) ونعرف برمجيا بقلة C++ كالآتي :-

```
struct node;
{
    int vertex;
    node link;
};
node head[n];
```

4- ننتقل الى موقع العقدة (f) في العمود الأول للبحث عن العقد المتصلة بها في العمود الثاني ونجد فيه العقدين (e) ، (f) . فناخذ العقدة (f) لأن العقدة (e) تم أخذها كما في الخطوة (3) أعلاه ، ويصبح تسلسل المرور $a < < e < < f < < a$.

5- ننتقل الى موقع العقدة (f) في العمود الأول للبحث عن العقد المتصلة بها في العمود الثاني ونجد فيه العقد (a) ، (e) ، (f) ، وهي عقد تم المرور عليها ووصلنا الى نهاية جدول الحقائق .

6- عند الوصول الى نهاية الجدول بعد المرور على العقد (a) ، (e) ، (f) ، (f) ، ولم يتم المرور على جميع عقد (عناصر) المخطط فوجب الاستمرار بالبحث عن العقد المتبقية وهي العقد (g) ، (b) ، (c) ، (d) ، (h) لم يتم المرور عليها ، فننتج طريقة (Back Tracking) اي الرجوع عبر المواقع السابقة بطريقة معكوسة من موقع العقدة (f) الى (f) ثم (e) ثم (a) .

7- في موقع العقدة (f) ، لا يوجد عقد لم نأخذها فننتقل الى موقع العقدة التي تسبقها في تسلسل المرور وهي (f) ، وهذه بدورها لا يوجد فيها عقدة لم نأخذها فننتقل الى العقدة التي تسبقها في تسلسل المرور وهي (e) وهي الأخرى كذلك فننتقل الى العقدة (a) فنجد في العمود الثاني لها العقدة (g) لم يسبق أخذها فناخذها ، ويصبح تسلسل المرور

$g < < i < < f < < e < < a$

8- ننتقل الى موقع العقدة (g) في العمود الأول للبحث عن العقد المتصلة بها في العمود الثاني ونجد فيه العقد (c) ، (b) ، (a) فناخذ العقدة (b) لأنه لم يسبق المرور عليها ويصبح تسلسل المرور:

$b < < g < < i < < f < < e < < a$

9- ننتقل الى موقع العقدة (b) في العمود الأول للبحث عن العقد المتصلة بها في العمود الثاني ونجد فيه العقدة (g) فقط وهي عقدة سبق المرور عليها . وهنا انتهى (Back Tracking) وبقيت في الجدول عقد لم يتم المرور عليها فناخذها بالتتابع حسب ورودها في الجدول أي (c) ، (d) ، (h) ، ونبدأ بالعقدة (c) ويصبح تسلسل المرور كالآتي :

$c < < b < < g < < i < < f < < e < < a$

10- ننتقل الى موقع العقدة (c) في العمود الأول للبحث عن العقد المتصلة بها في العمود الثاني ونجد فيه العقدين (h) ، (d) فناخذ العقدة (d) لأنه لم يسبق المرور عليها ويصبح تسلسل المرور:

$d < < c < < b < < g < < i < < f < < e < < a$

استعراض (مسح/مرور) لعناصر المخطط Graph Traversal
لغرض الوصول (المرور) الى كل عنصر من عناصر المخطط نحتاج الى معرفة الطريقة المناسبة لذلك وهي المرور لمرة واحدة فقط على كل عقدة في المخطط (العقدة تمثل موقع العنصر البياني Data Item)

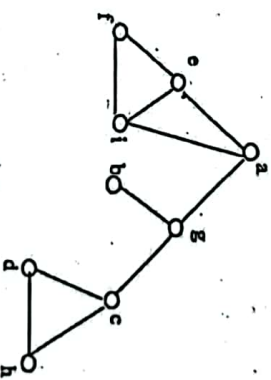
وبالامكان البدء بأية عقدة في المخطط . هنالك طريقتان لاستعراض عناصر المخطط هما :

1 / Depth First Traversal (DFT)

تتلخص هذه الطريقة بالخطوات المبينة أدناه وتتخذها على المخطط المرسوم في الشكل

1 - اعداد جدول الحقائق (Edges Table) والذي يحتوي في عمومه الأول جميع عقد المخطط (التي تمثل العناصر البيانية) ، ولوضع في العمود الثاني بصورة اقلية أسماء العقد التي تتصل (تربط) بها كل عقدة من عقد العمود الأول .

العمود الثاني	العمود الأول
عناصر	عناصر المخطط
$a, b, c, d, e, f, g, h, i$	a
b, c, d, e, f, g, h, i	b
c, d, e, f, g, h, i	c
d, e, f, g, h, i	d
e, f, g, h, i	e
f, g, h, i	f
g, h, i	g
h, i	h
i	i



جدول الحقائق

المخطط

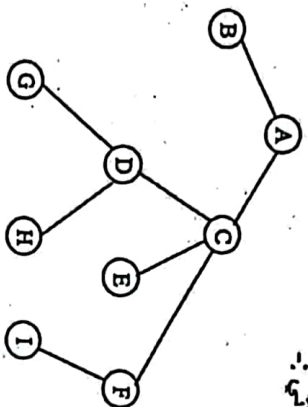
الشكل (6-8)

2- نؤخذ العقدة الأولى من العمود الأول وهي (a) .

[ملاحظة : يمكن البدء بأية عقدة أخرى ، وذلك يمكن أن نحصل على أي ترتيب آخر للمرور على العناصر (المقد)]

3- نؤخذ من العمود الثاني أول عقدة تتصل بها وهي (e) ، ثم ننتقل الى موقع العقدة (e) في العمود الأول والبحث عن العقد المتصلة بها في العمود الثاني ونجد فيه العقد (a) ، (f) ، (f) . وبما أن العقدة (a) سبق ان تم المرور عليها كما في الخطوة (2) أعلاه فنتركها ونأخذ العقدة (f) ليصبح تسلسل المرور :

$f < < e < < a$



لناخذ هيكل الشجرة الاتي :-

الشكل (6 - 11)

وفيما يأتي عدد من التعريفات لتوضيح المفردات المتعلقة بهيكل الشجرة وطريقة استخدامها مع الإشارة الى الشكل اعلاه .

جذر الشجرة Root

هي العقدة التي لا تسبقها عقدة اخرى في الشجرة اي انها ليس لها اب (it has no father) في الشكل السابق.

العقدة المتفرعة branched node

هي العقدة التي لها تفرع مثل A, C, D, F,

العقدة النهائية (الورقة) Terminal (leaf) node

هي العقدة التي ليس لها تفرع مثل B, E, G, H, I,

مستوى العقدة Node level

هو عدد المسارات التي يقعد العقدة عن الجذر

فمستوى عقدة الجذر = صفر

ومستوى العقدة E = 2

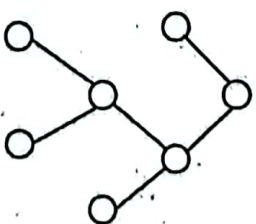
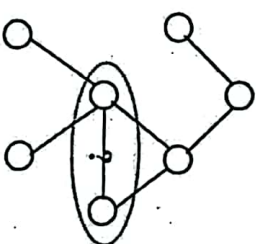
ومستوى العقدة H = 3

2-6 هيكل الشجرة Tree Structure

هناك هيكل بياني ممثلة المخطط المتجه ، اي انها هيكل بياني غير خطية (Non - Linear) مثل تشعب طرق المواصلات في خرائط المدن ، واعتماد مبادئ نظرية المخططات يساعد على تمثيل هذه الهياكل البيانية والتعامل معها من حيث البرمجة والتخزين باستخدام الحاسوب .

الشجرة Tree

هي تركيب من نوع مخطط متجه (Digraph) ولكن بدون تشكيل دائري (Acyclic) اي هناك خط واحد يوصل بين أي نقطتين (عقدتين) .



ليس هيكل شجرة لوجود التشكيل الدائري

هيكل شجرة

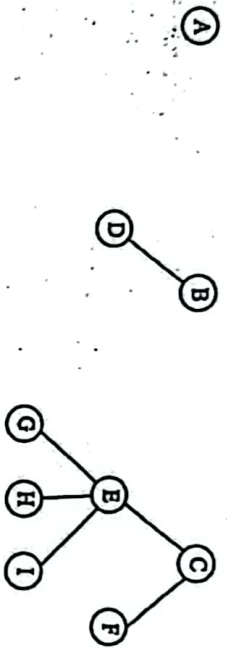
الشكل (6 - 10)

كما يمكن تعريف هيكل الشجرة بأنه مجموعة غير خالية من العقد تتصف بما يأتي:-

+ توجد عقدة واحدة تسمى الجذر (Root) وهي التي لا يسبقها أية عقدة (العقدة التي ليس لها اب (It has no father) .

+ العقد المتبقية remaining nodes مجزأة الى مجموعات منفصلة كل منها هو هيكل شجرة أيضا يسمى شجرة فرعية (subtree) .

هذه الشجرة يمكن تخزينها الى الاشجار النرجية الالية :



(أ)

(ب)

(ج)

ويمكن الاستمرار بالتجزئة فالتجزئة النرجية (ب) يمكن تخزينها ايضا الى شجرتين فرعتين هما :

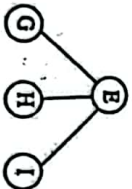
(ب)

(د)

والشجرة النرجية (ج) يمكن تخزينها الى

(ب)

(ف)



(د)

(هـ)

(و)

وكذلك الشجرة النرجية (هـ) يمكن تخزينها الى اربع اشجار فرعية كل منها تتكون من عقدة واحدة

(ب)

(د)

(هـ)

(و)

درجة العقدة Node degree

هي عدد السمات الخارجة منها مباشرة (او عدد الابناء فيها)
No. of children (او عدد التفرعات المباشرة منها) .

فدرجة العقدة 0 = H ودرجة العقدة 1 = F ودرجة العقدة 2 = A ودرجة العقدة 3 = C

درجة الشجرة Tree degree

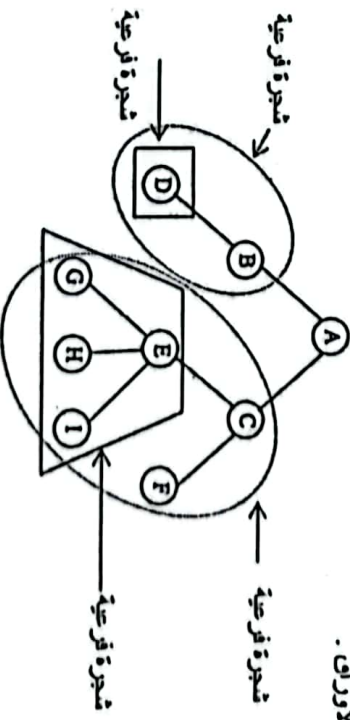
هي اعلى درجة من درجات العقد المكونة للشجرة فدرجة الشجرة المرسومة في الشكل (11-6) هي (3) .

ارتفاع الشجرة Tree height

هو اكبر مستوى (level) لاية عقدة في الشجرة أي هو اطول مسار في الشجرة .

الشجرة الفرعية Subtree

يمكن تجزئة الشجرة الى اجزاء هي اشجار فرعية يكون لكل منها جزءا وقد تكون لها بعض الاوراق .



الشكل (6 - 12)

في الشكل (6-13) نلاحظ ما يلي :-

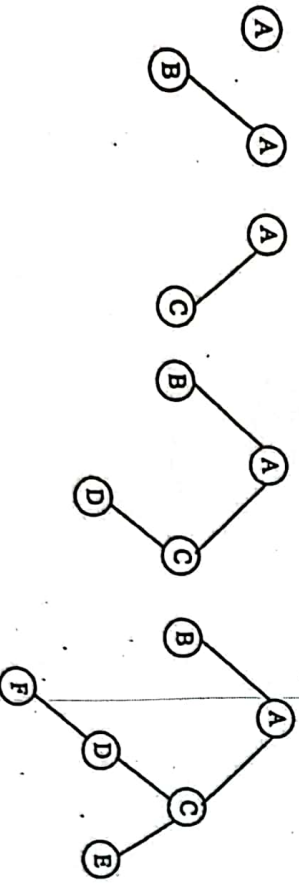
- + جذر الشجرة هو A
- + أوراق الشجرة هي K, J, I, H, D
- + العقد المتفرعة A, B, C, E, F, G
- + العقدة B هي أب للعقد F, E, D
- + العقدة C هي أب للعقد G فقط
- + العقدة E هي أب للعقد H
- + العقدة F هي أب للعقد J, I
- + العقدة G هي أب للعقد K
- + العلاقة بين العقد F, E, D هي علاقة أخ (sibling) أو (brother)
- + العلاقة بين العقد J, I هي علاقة أخ
- + العقدة E لها علاقة عم بالعقدة J
- + العقدة B لها علاقة جد (grand father) بالعقدة I

وهكذا باقي العلاقات

- + ارتفاع الشجرة هو (3) لأن أطول مسار هو (3)
- + درجة الشجرة هي (3) لأن أكبر درجة فيها للعقدة (B) هي (3)

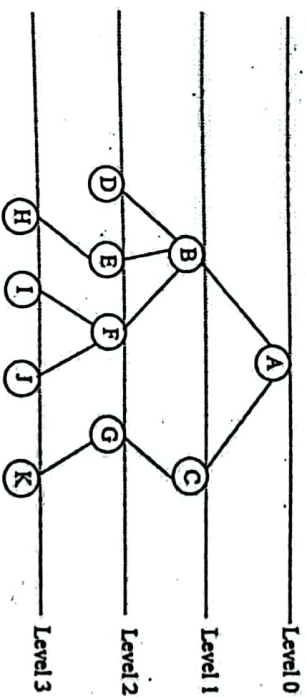
3-6 الشجرة الثنائية Binary Tree

هي الشجرة التي كل عقدة فيها لا تحتوي على أكثر من عقدين فرعيتين (at most two sons) أي أن درجة أية عقدة فيها لا تزيد على 2 (أما 0 أو 1 أو 2) وهذه الشجرة تمثل هيكل بياني مهم ولها تطبيقات كثيرة في علم الحاسبات والاشكال الآتية تمثل اشجار ثنائية مختلفة :

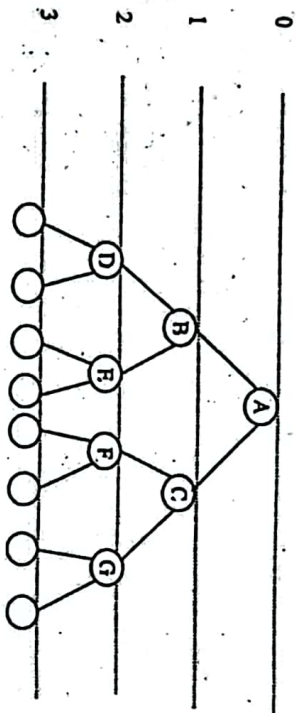


ملاحظات أخرى :

- 1- لا يوجد توصيلات بين العقد في مستوى واحد .
- 2- لا يوجد توصيلات بين أوراق الشجرة .
- 3- لا توجد دورات في الشجرة .
- 4- كل عقدة تكون بمثابة ابن (son) بالنسبة للعقد المتفرعة منها مباشرة وكل من تلك العقد تكون بمثابة ابن (son) بالنسبة للعقدة الاب .
- أي يمكن استخدام المصطلحات العائلية (ابن son) - أخ sibling/brother
- أب father - عم cousin - جد grand father - جد الجد grand grand father
- في تسمية العقد والعلاقات بينها .
- 5- كل عقدة لها أب واحد one father .

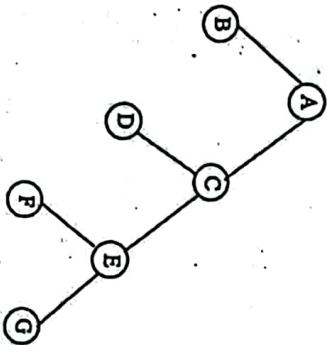


الشكل (6-13)



الشكل (6-15) - أ

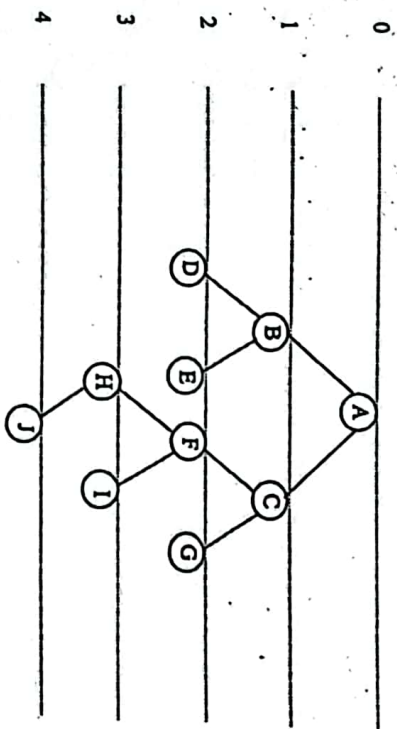
أما الشجرة في الشكل (6-15 - ب) فهي شجرة ثنائية ارتفاعها $h = 3$ إلا أن الحد الفعلي للعقد فيها هو (7).



الشكل (6-15) - ب

ملاحظات أخرى:

- عدد أوراق الشجرة الثنائية = (عدد العقد التي درجتها 2) + 1
- تكون الأشجار الثنائية متكافئة مع بعضها (equivalent) إذا كان لها نفس التركيب أي نفس الهيئة من حيث عدد مواقع العقد وشكل التفرعات وتطبيق البيانات.



الشكل (6-14)

أكبر عدد من العقد في المستوى

في الشجرة الثنائية يكون أكبر عدد من العقد في مستوى معين (L) هو 2^L فالشجرة في الشكل السابق (6-14) نلاحظ في المستوى $L = 2$ فإن أكبر عدد ممكن من العقد فيه هو $2^2 = 4$ وهي G, F, E, D أما المستوى التالي ($L = 3$) فإن أكبر عدد ممكن من العقد هو ($2^3 = 8$) أي أن الحد الأقصى الذي لا يمكن تجاوزه هو (8) إلا أنه يمكن أن يكون أقل كما في الشجرة المذكورة حيث عدد عقد المستوى 3 هي اثنين هما (I, H)

أكبر عدد من العقد في الشجرة الثنائية:

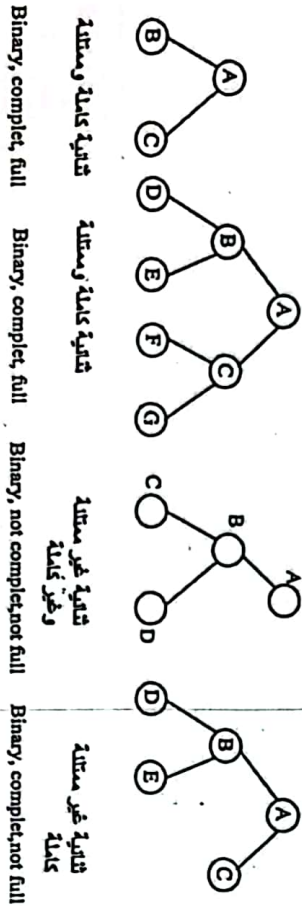
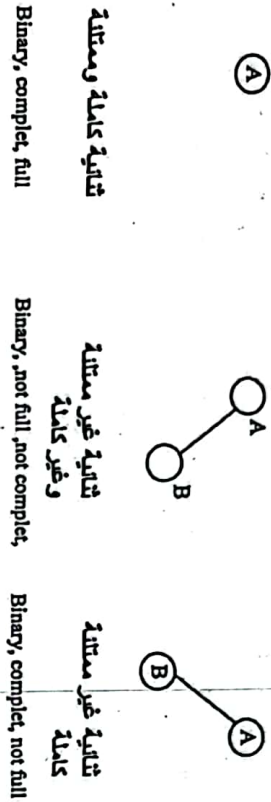
في الشجرة الثنائية التي ارتفاعها (h) فإن الحد الأقصى لعدد العقد فيها هو ($2^{h+1} - 1$)، وقد يكون الحد الفعلي للعقد أقل من هذا.

مثال:

نلاحظ الشجرة الثنائية في الشكل (6-15-أ) أن ارتفاعها $h = 3$ أكبر عدد من العقد فيها $2^{3+1} - 1 = 15$

$$15 = 2^4 - 1 = 16 - 1$$

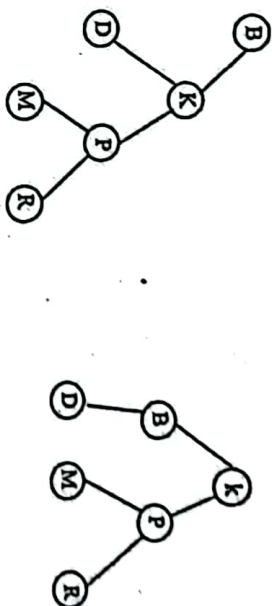
الشجرة الثنائية الكاملة Complete Binary Tree
هي الشجرة الثنائية التي تكون إما ممتلئة أو ممتلئة لحد المستوى قبل الأخير وتكون أوراق المستوى الأخير في أقصى اليسار.



الشكل (6-17)

ملاحظة: يكون شكل الشجرة مثلث Δ إذا كانت ممتلئة أو يكون شكلها ∇
شجرة AVL-Tree
وهي شجرة ثنائية غير خالية والفرق في ارتفاع اشجارها الارتفاعية لأية عقدة فيها لا يزيد على 1، أي أن ارتفاع الشجرة الارتفاعية اليسرى (T_L) لأية عقدة لا يزيد على ارتفاع الشجرة الارتفاعية اليمنى (T_R) بكثير من 1
أي $|h_L - h_R| < 1$

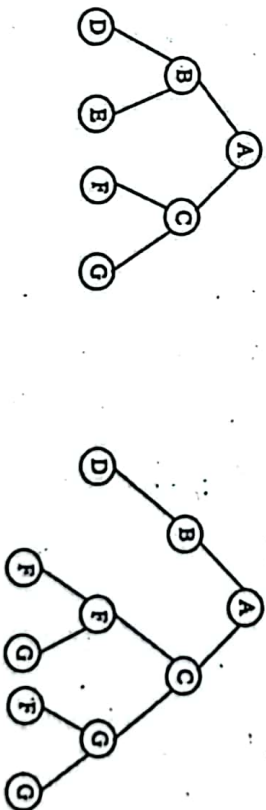
4-6 أنواع أخرى من الأشجار:
الشجرة الثنائية المتوازنة Binary Balanced Tree
هي شجرة ثنائية يكون الفرق بين ارتفاع اشجارها الارتفاعية لأية عقدة فيها إما (0) أو (1)



شجرة ثنائية غير متوازنة لأن العقدة (B) لها شجرة فرعية اليمنى ارتفاعها (3) والشجرة الفرعية اليسرى ارتفاعها (0).

شجرة ثنائية متوازنة

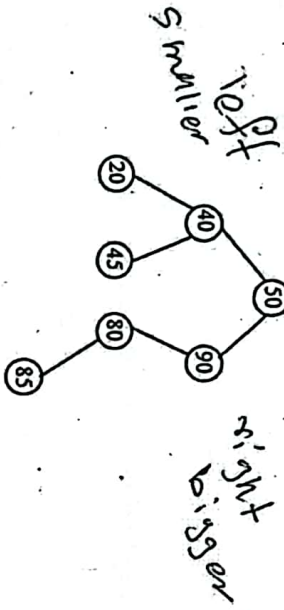
الشجرة الثنائية الممتلئة Full Binary Tree
هي الشجرة التي جميع أوراقها في مستوى واحد وأية عقدة مقترعة لها فرعان بالضبط (تكون مثلثة الشكل Δ).
بالضبط (تكون مثلثة الشكل Δ).



الشكل (6-16)

شجرة البحث الثنائية Binary Search Tree

هي الشجرة الثنائية التي تكون قيمة عنصر الفرع الأيسر (الابن) أقل من قيمة عنصر تلك العقدة باعتبارها الأب (father) وتكون قيمة عنصر الفرع الأيمن (الابن) أكبر من قيمة عنصر العقدة (الأب).



الشكل (6 - 18)

الشجرة m - way search Tree

هي شجرة بحث متوازنة تكون جميع عقدها بدرجة (m) أو أقل.

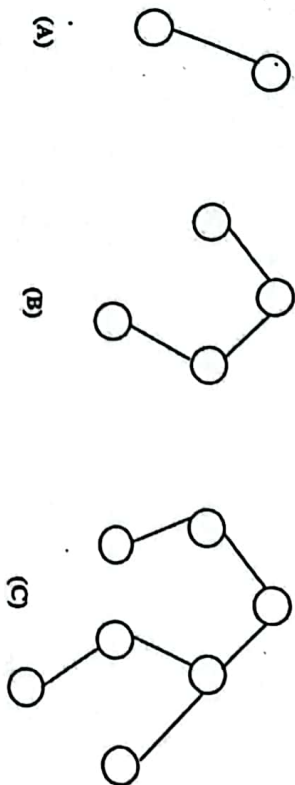
الشجرة B - Tree

في أنظمة قواعد البيانات حيث خزن البيانات على أوساط الخزن الثنائية (disks, tapes) فإن تقليص وقت الوصول للمعلومة يتحقق عند اختيار الهيكل البيناري المناسب ، ومن هذه الاختيارات المناسبة هو استخدام (B-tree).

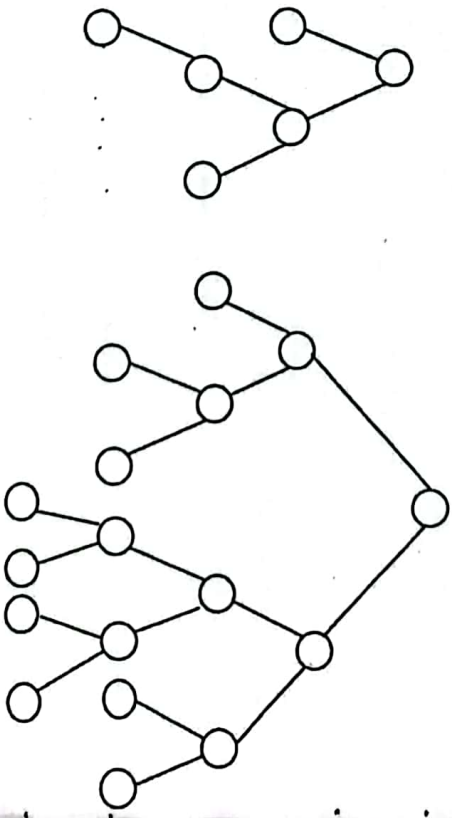
من أهم خصائص هذه الشجرة هو أن كل عقدة فيها يمكن أن يكون حجمها (سعتها) كبير لاستيعاب (Block of Data) ، وعدد القيم المفتاحية (keys) في كل عقدة يقدم على حجم كل مفتاح (key size) ، واستخدام هذه الشجرة بارتفاع أقصر يساعد على وصول أسرع إلى أكبر حجم من البيانات المخزنة في أوساط الخزن الثنائي لكل حالة (بحث) وصول (at each disk access) ، ويمكن تعريفها كالآتي :

ملاحظة : اسم الشجرة هو مختصر لاسماء الأشخاص الثلاثة الذين استخدموها وهم :

Adelson - Velskii - Landis



أمثلة شجرة AVL-Tree



Not-AVL-Tree

AVL-Tree

عند تطبيق شروط التعريف بالتتابع على هذه الشجرة ملاحظ ما يلي :

- + أي عقدة في الشجرة لا يزيد عدد فروعها عن (5).
- + أن عقدة الجذر (a) لها فرعان فقط هما (b) و (c).
- + العتدان المتفرعان هما : (b) ولها (4) فروع ، (c) ولها (3) فروع أي أكثر من $(m/2 = 5/2 = 2)$.
- + جميع الأوراق لا يزيد عدد القيم المفاتيحية في أي منها عن (4) أي $(m-1=5-1=4)$.
- + العتدان المتفرعان وهما :

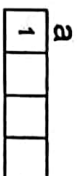
- (b) عدد القيم المفاتيحية فيها = 3 وعدد فروعها = 4
- (c) عدد القيم المفاتيحية فيها = 2 وعدد فروعها = 3
- + جميع عقد الأوراق (a, b, c, d, e, f, g, h, i, j) تقع في مستوى واحد
- ملاحظة : اسم الشجرة جاء من اسم الشخص الذي استخدمها وهو (Bayer)
- مثال تطبيقي لبناء شجرة B-Tree Construction a B-Tree

نتعرض لبناء مجموعة من القيم المفاتيحية (keys) وردتنا بالتسلسل التالي من اليمين إلى اليسار :

3, 68, 48, 16, 52, 7, 17, 28, 14, 6, 25, 2, 8, 12, 1, 45, 55, 53, 29, 26

المطلوب : بناء (تكوين) B-tree of order m=5
الخطوات :
1- لأن $(m=5)$ فإن عدد القيم المفاتيحية في أية عقدة يجب أن لا يزيد عن (4) ونضع أول أربع قيم مفاتيحية في عقدة الجذر (a) بترتيب تصاعدي وفق طريقة شجرة البحث الثنائي (binary search tree)

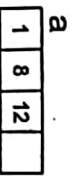
وبالمثل التالي :
عند وصول العنصر الأول (1) نضعه في العقدة (a)



عند وصول العنصر الثاني (12) نضعه في العقدة (a)



عند وصول العنصر (8) ، يستوجب تغيير مواقع القيم المفاتيحية في العقدة (a) لتصبح بترتيب تصاعدي

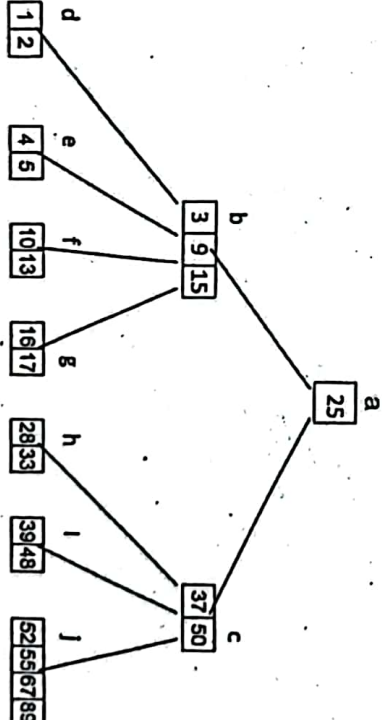


هي شجرة بحث متعدد تكون جميع عقدها بدرجة (m) أو أقل ، وتتصف هذه الشجرة بما يلي :

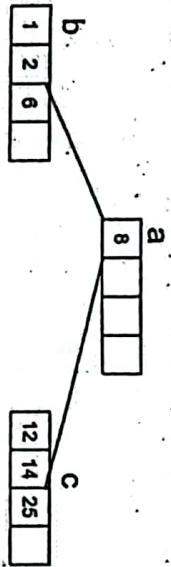
- 1- جميع العقد لا تزيد قدرعات أي منها لا يزيد عن (m).
- 2- عقدة الجذر (root) تكون إما ورقة (leaf) أو لها على الأقل فرعان.
- 3- أية عقدة أخرى (عدا الجذر) فإن عدد فروعها (ابنائها) يتراوح بين $(m, m/2)$
- 4- عدد القيم المفاتيحية (keys) في أية عقدة مفردة (non-leaf) يكون أقل بمقدار (1) من عدد فروعها .
- 5- جميع عقد الأوراق تكون في مستوى واحد (same level).

وفق هذه المواصفات يعني أن هذه الشجرة ستكون :
+ نصف مملوءة على الأقل
+ عدد قليل من المستويات
+ هي شجرة متوازنة تامة (perfectly balanced)

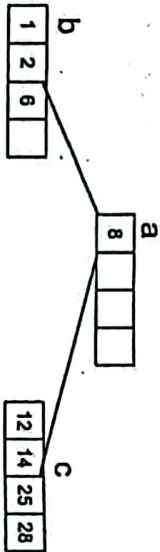
لنأخذ الشجرة B-tree (order 5) في المثال التالي



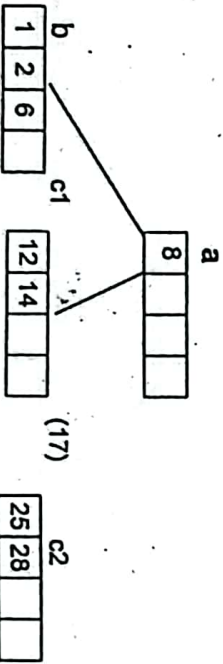
4- وصول العنصر (14)



5- وصول العنصر (28)



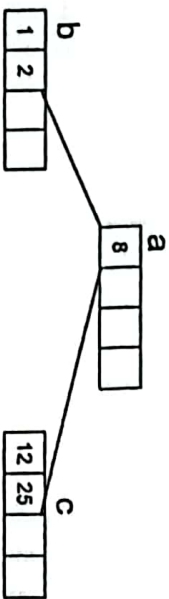
6- وصول العنصر (17) ، أن قيمته تكون ضمن مدى القيم المفتاحية للعقدة (c) وفق الترتيب التالي (12 ، 14 ، 17 ، 25 ، 28) ، ولأن سعة العقدة يجب أن لا تتجاوز (4) لأن ($m=5$) لذا نأخذ القيمة الوسطية (17) ونجعلها جذر "القيم المفتاحية الأربع الباقية التي سنخزنها في عقدين هما (c1) و (c2) .



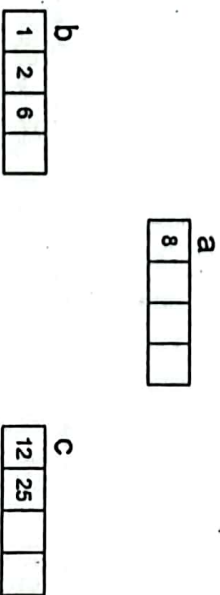
عند وصول العنصر (2) نضمه في العقدة (a) بترتيب تصاعدي أيضا :



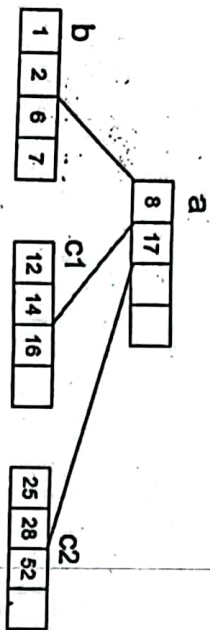
2- عند وصول العنصر (25) فإن قيمته تكون ضمن مدى القيم المفتاحية للعقدة (a) وفق الترتيب التالي: (1 ، 2 ، 8 ، 12 ، 25) ، ولأن سعة العقدة (أي عدد القيم المفتاحية فيها no. of keys) يجب أن لا يزيد عن (4) لأن ($m=5$) لذا نأخذ القيمة الوسطية وهي (8) ونجعلها هي عقدة الجذر للقيم المفتاحية الأربع المتبقية ، إذ سنخزنها في عقدين هما (b) و (c) .



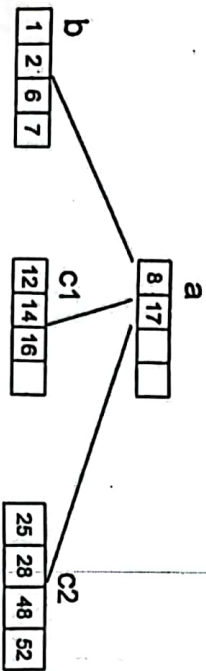
3- عند وصول عنصر جديد فيجب أن تبدأ البحث عن موقع له في إحدى الأوراق (leaf node) لذا عند وصول العنصر (6) سيكون موقعه في العقدة (b) .



9- وصول العنصر (16)

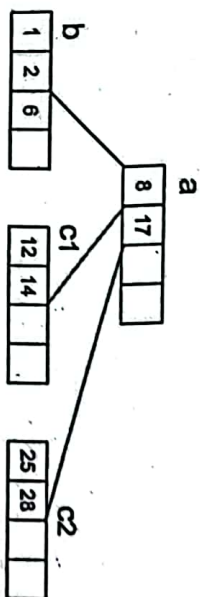


10- وصول العنصر (48)

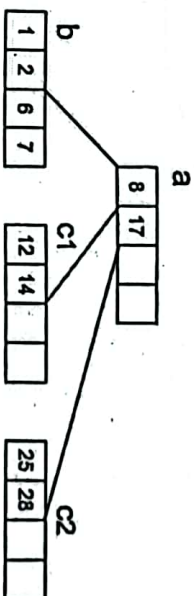


11- وصول العنصر (68) ، وقيمته تكون ضمن مدى القيم المفتاحية للعقدة (c2) وفق الترتيب التالي (25 ، 28 ، 48 ، 52 ، 68) ، ولأن سعة العقدة يجب ان لا تزيد عن (4) (لان $m=5$) لذا نأخذ القيمة الوسطية (48) ونجعلها جذر القيم المفتاحية الاربعة المتبقية اذ سنحزنها في عقليتين هما (c2) و (c3).

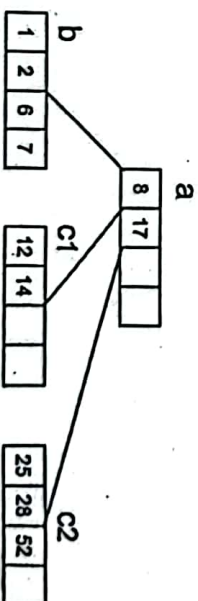
وفي هذه الحالة فإن العنصر (17) يجب ان يكون في العقدة (a) ونصبح كالآتي



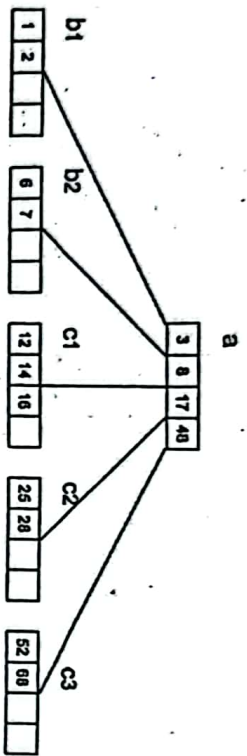
7- وصول العنصر (7)



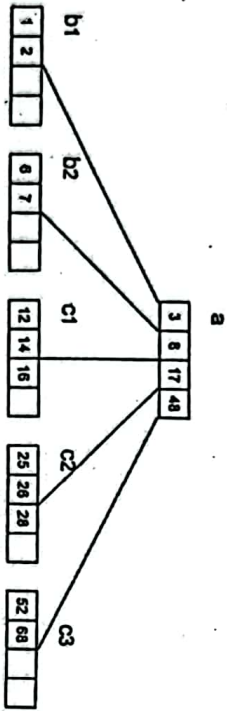
8- وصول العنصر (52)



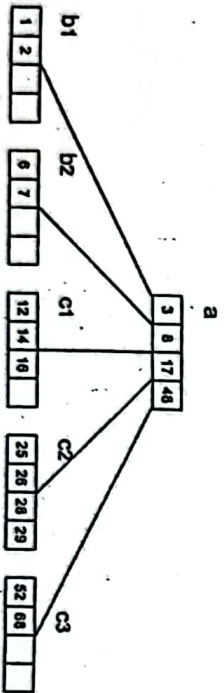
وفي هذه الحالة فإن العنصر (3) يجب أن يكون في العقدة (a) وتصبح كالآتي



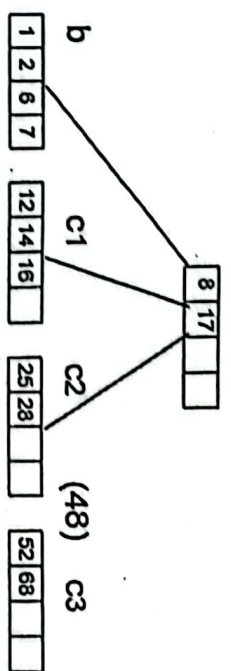
13 - وصول العنصر (26) وسيكون موقعه في العقدة (c2)



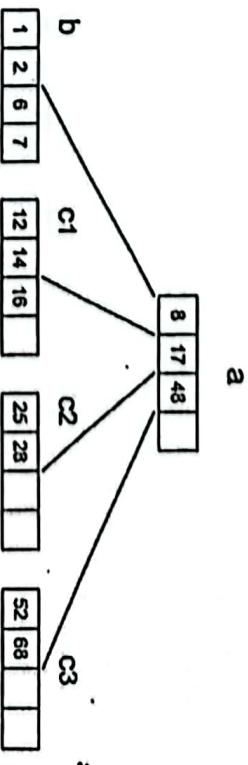
14 - وصول العنصر (29) وسيكون موقعه في العقدة (c2)



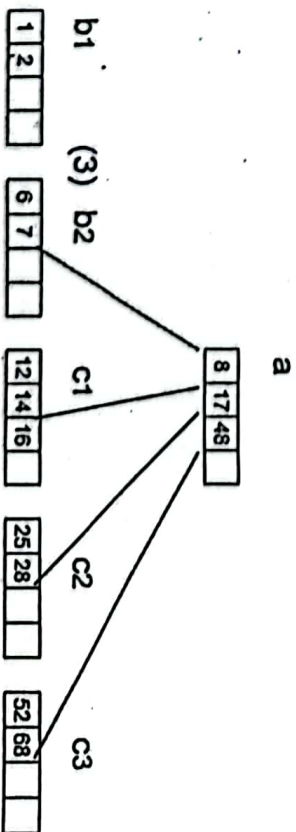
a



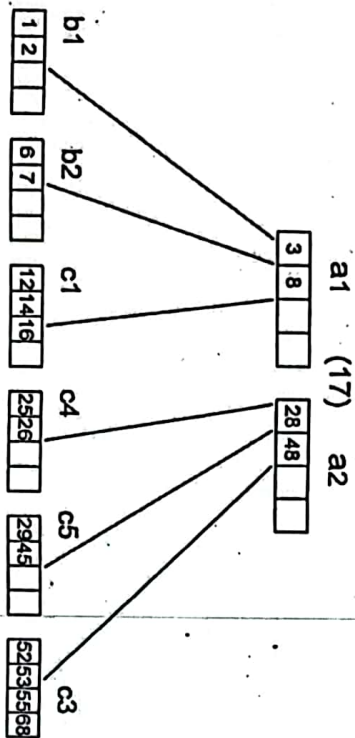
وفي هذه الحالة فإن العنصر (48) يجب أن يكون في العقدة (a) وتصبح كالآتي



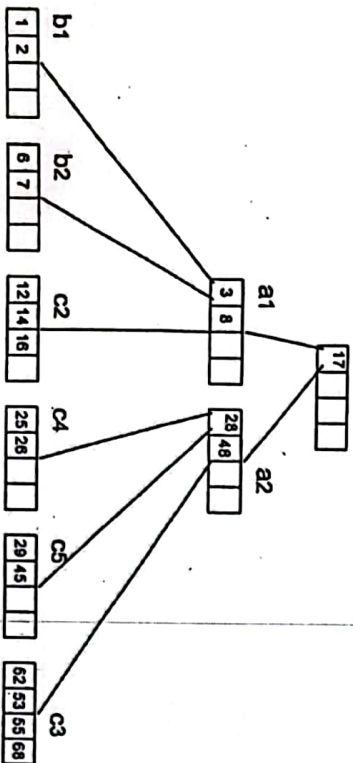
12 - وصول العنصر (3) وستكرر نفس الخطوات التي تمت في الفترة السابقة (11) إذ سيصبح جذر "العقدين (b1) و (b2) :



وفي هذه الحالة فإن العنصر (28) يجب أن يكون في العقدة (a) وفق الترتيب التالي (3، 8، 17، 28، 48)، إلا أن هذه العقدة هي في حالة امتلاء لأحزائها (4) قيم مفتاحية (4 keys) لذا سنأخذ القيمة الوسطية (17) ونجعلها جذر "لباقى القيم المفتاحية الأربع إذ سنخزنها إلى عقدين هما (a1) و (a2).

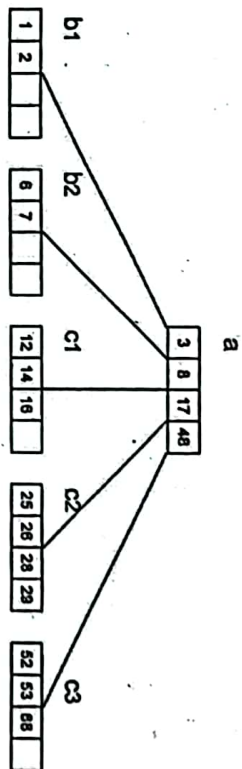


أي ستكون الحالة النهائية كالآتي :

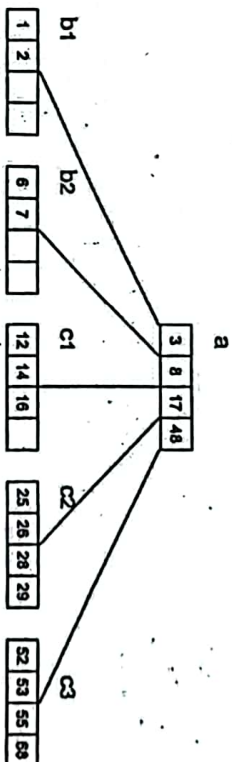


219

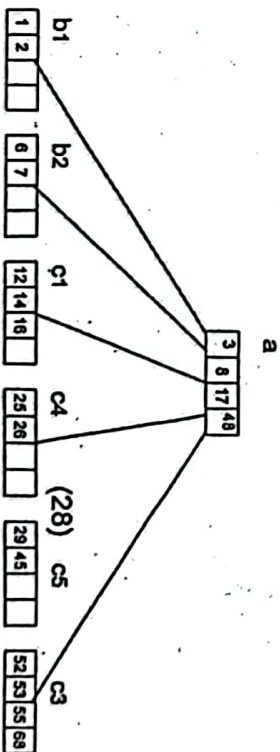
15 - وصول العنصر (53) وسيكون موقعه في العقدة (c3)



16 - وصول العنصر (55) وسيكون موقعه في العقدة (c3)



17 - وصول العنصر (45)، وهو أقل من قيمة العنصر (48) في العقدة (a) وضمن مدى عناصر العقدة (c2) وفق الترتيب التالي (25، 26، 28، 29، 45)، لذا نأخذ القيمة الوسطية (28) ونجعلها جذر " للقيم المفتاحية الأربع المتبقية إذ سنخزنها في عقدين هما (c4) و (c5).



218

ب / الاستعراض من أسفل إلى أعلى Bottom - Up Traversing

وتتلخص خطوات هذه الخوارزمية بالآتي :-

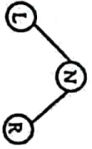
- 1- البدء بالورقة في أقصى اليسار بالتي مستوى .
- 2- التحرك نحو العقدة في اليمين منها بنفس المستوى لحين الانتهاء من زيارة جميع عقد ذلك المستوى .
- 3- الانتقال إلى المستوى الأعلى وزيارة العقد فيه أيضا من اليسار إلى اليمين ، وهكذا تستمر العملية لحين الوصول إلى جذر الشجرة .

أي أن نتيجة استعراض نفس الشجرة بهذه الطريقة تكون :-

البداية → F G H D E B C A

ملاحظة :-

نلاحظ أن التعامل مع الشجرة وأجزائها يوضح أن التشكيل الأساسي والمكرر فيها هو تكوينها من عقدة الجذر (N) وقد تحتوي ورقة أو أكثر أو بدون أوراق لذا فإن التعامل مع الشجرة يمكن أن يبدأ بالجذر ولتوضعه (N) أو بالورقة في أقصى اليسار (L) أو بالورقة في أقصى اليمين (R) .



أي أن الشكل العام للشجرة

ولهذا فإن احتمالات الاستعراض هي ستة

RLN RNL LRN LNR NRL NLR

وسنأخذ فقط الحالات التي تمثل الاستعراض من اليسار إلى اليمين وهي ثلاثة :

NLR أي البدء بالجذر (N) ثم التحرك نحو اليسار (L) ثم اليمين (R) ولكون الجذر

ونكر هنا مميتا تسمى هذه الطريقة بالترتيب السابق (Preorder)

LRN أي البدء باليسار (L) ثم اليمين (R) والانتهاء بالجذر (N) أي أن نكرر الجذر

وأي لاحقا وتسمى هذه الطريقة بالترتيب اللاحق (Postorder)

(N)

LNLR البدء باليسار (L) ثم الجذر (N) وتليه اليمين (R) أي أن الجذر يأتي في

الوسط وتسمى هذه الطريقة (inorder) نسبة إلى الجذر (N).

5-6 استعراض (مسح) عقد الشجرة Tree Traversing

أن عملية المسح تعني المرور (زيارة visit) كل عقدة في الشجرة مرة واحدة فقط

ولا يجوز تكرار الزيارة .

وبالنظر لكون هيكل الشجرة هو هيكل يسمي لا خطي (Non-linear data structure) لذا فإن عمليات البحث (search) عن عنصر (عقدة) معين في هذا الهيكل أو إضافة عنصر إليه أو حذف عنصر منه تختلف عن أسلوب التعامل مع الهياكل الأخرى وأن اختيار إحدى هذه الطرق يعتمد على كيفية تمثيل الشجرة في الذاكرة . وفيما يأتي أهم الطرق المستخدمة لهذا الغرض :

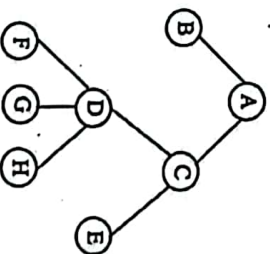
أولاً - الاستعراض حسب المستويات level by level Traversing

1- الاستعراض من أعلى إلى أسفل Top-down Traversing

وتتلخص الخوارزمية بالخطوات التالية :-

- 1- البدء بعقدة الجذر ..
- 2- استعراض عقد المستوى التالي ومن أقصى اليسار إلى اليمين .
- 3- الاستمرار بنفس الطريقة في الانتقال إلى المستويات الأدنى بالتتابع والبدء بالعقدة في أقصى اليسار ثم اليمين .

مثال :- لناخذ الشجرة الآتية :-



الشكل (6 - 19)

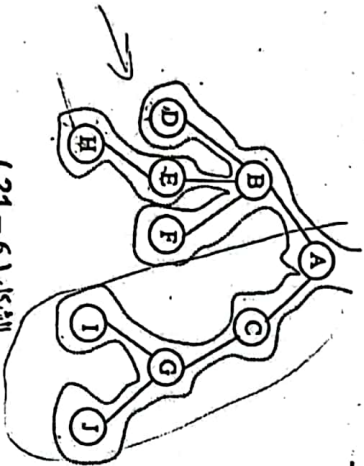
تكون نتيجة استعراض عقدها بطريقة Top - Down هي :-

البداية → A B C D E F G H

ثانياً : الاستعراض بالترتيب اللاحق (LRN) Postorder Traversing

وتتلخص خطوات هذه الخوارزمية بالآتي :-

- 1- البدء بالعقدة الوريقة في أقصى يسار الشجرة ثم الأوراق التي على يمينها (ان : وجدت) .
- 2- الانتقال الى العقدة الاعلى (father) (أي اب تلك العقدة)
- 3- مسح الشجرة الفرعية التالية في اليمين بنفس الطريقة لحين الوصول الى الجذر .



الشكل (6 - 21)

وستكون نتيجة الاستعراض كالآتي :-

البدائية → D H E F B I J G C A

في هذه الطريقة نلاحظ ما يأتي :-

1- جميع الاباء يكونون بعد الابناء فمثلا B بعد F, E, D

H بعد E

G بعد J, I

وهكذا

2- لو مثلنا هذا الاستعراض (المسح) بالمسير حول الشجرة (الخط المنقط) لوجدنا ان العقدة تذكر بعد معانيتها بدء من الورقة في أقصى اليسار وانتهاء بالجزء .

3- تستخدم هذه الطريقة لتمثيل المعايير الحسابية

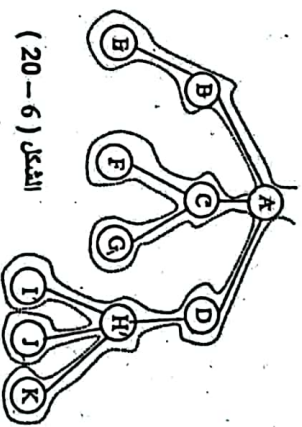
بصيغة Reverse Polish Notation (RPN)

ثانياً : الاستعراض بالترتيب السابق (NLR) Preorder Traversing

وتتلخص خطوات هذه الخوارزمية بالآتي :-

- 1- البدء بعقدة الجذر (N)
- 2- استعراض الشجرة الفرعية في أقصى اليسار .
- 3- داخل الشجرة الفرعية يتم الاستعراض من أقصى اليسار (يمثل اكبر الابناء) ثم التفرع لليمين .
- 4- في حالة لا يوجد فرع في اليمين (لا يوجد أخ brother) يكون الانتقال الى السم (father's brother) .

نتأخذ الشجرة التالية :



الشكل (6 - 20)

ستكون نتيجة الاستعراض كالآتي :-

البدائية → A B E C F G D H I J K

في هذه الطريقة نلاحظ ما يأتي :-

1- جميع الاباء يكونون قبل الابناء A قبل D, C, B

C قبل G, F

H قبل J, I, K

وهكذا

2- لو مثلنا هذا الاستعراض بالمسير حول الشجرة (الخط المنقط) لوجدنا ان العقدة تذكر عند اول مرور بها يبدأ من الجذر ..

3- تستخدم هذه الطريقة لتمثيل المعايير الحسابية بصيغة Polish Notation

6-6 تمثيل الأشجار Tree Representation

هناك عدة طرق لتمثيل الأشجار عند الخزن في الحاسوب وتحديد أفضلها يعتمد على :-

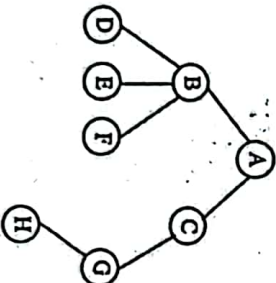
- + العمليات التي تتطلبها المسألة المعنية بالحل
- + منظومة الحاسوب المستخدمة .
- + لغة البرمجة

1-6-6 تمثيل الأشجار الاعتيادية General Tree Representation

|| عدد المؤشرات بقدر أكبر عدد من الفروع

لكل عقدة في الشجرة عدد معين من الإبناء different No. of children وبإستخدام القائمة الموصولة (linked list) لتمثيل مثل هذه الشجرة فيجب تحديد مؤشر (pointer) لكل ابن child ولهذا :
فالعقدة التي لها ابن واحد (one child) تحتاج الى مؤشر واحد .
والعقدة التي لها اثنان (two children) تحتاج الى مؤشرين .
والعقدة التي لها ثلاثة أبناء (three children) تحتاج الى ثلاثة مؤشرات .

وهكذا



الشكل (6 - 23)

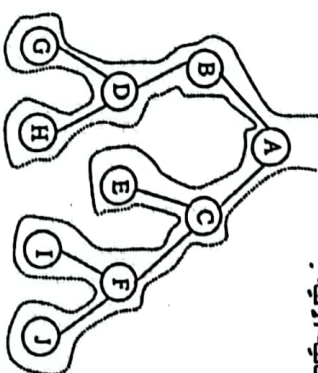
225

وهذه القائمة الموصولة يجب ان تعرف فيها عدد المؤشرات بقدر أكبر عدد من الإبناء لاية عقدة في الشجرة ، وهذا يعني ان كل عقدة سيكون لها نفس العدد من المؤشرات حتى لو كان عدد فروعها (ابناءها) اقل من ذلك ، وهذا سيعطي ضياع كبير في استخدام المساحة التخزينية .
لنأخذ الشجرة التالية :

رابعاً : الاستعراض بالترتيب البيني Inorder Traversing

ان هذه الطريقة تستخدم في مسح الأشجار الثنائية فقط وتلخص خطوات الخوارزمية فيها بالاتي :

- 1- البدء بالعقدة الورقة في أقصى يسار الشجرة
 - 2- الانتقال الى عقدة الجذر (اب تلك العقدة)
 - 3- زيارة العقدة التي في اليمين (ان وجدت) وفي حالة عدم وجودها يكون الانتقال الى الجد (grand father)
- لنأخذ الشجرة الثنائية الاتية :



الشكل (6 - 22)

- ان نتيجة استعراض (مسح) هذه الشجرة هي :
- البدائية B G D H A E C I F J →
- وفي هذه الطريقة تلاحظ ما يلي :-
- 1- لو مثلنا هذا الاستعراض بالسير حول الشجرة (الخط المنقط) لوجدنا ان العقدة تفكر عند المرور تحتها
 - 2- تستخدم هذه الطريقة لتمثيل التعابير الحسابية بصيغة Infix Noatation

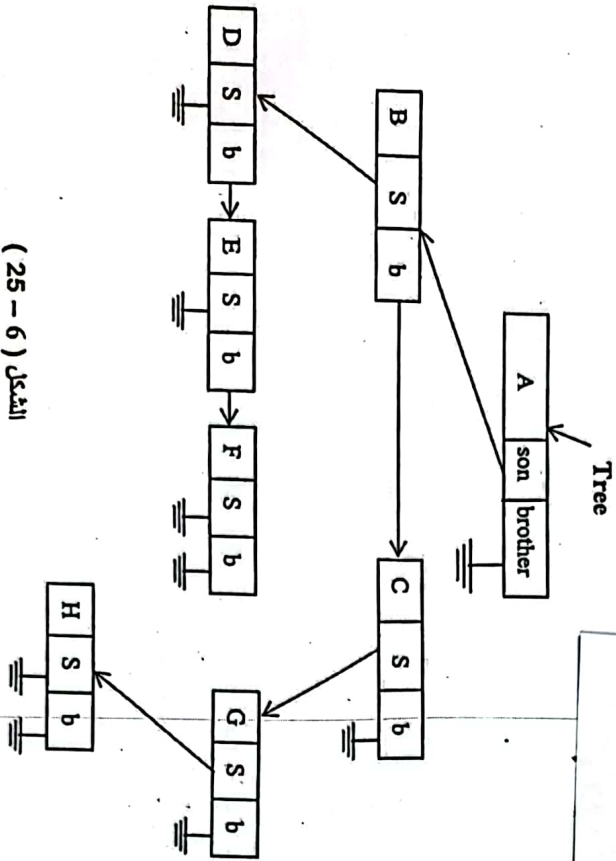
224

```

struct node
{
    char data;
    node son, brother;
} *tree;

```

ب / مؤشر لكل عقدة
 يكون لكل عنصر من عناصر القائمة مؤشرين هما :
 + مؤشر يشير الى اكر الابناء في اليسار
 eldest son in the left
 + مؤشر يشير الى الاخ التالي
 next brother
 فيكون تعريف القائمة الموصولة لنفس الشجرة في المثال السابق كالآتي :-



الشكل (6 - 25)

227

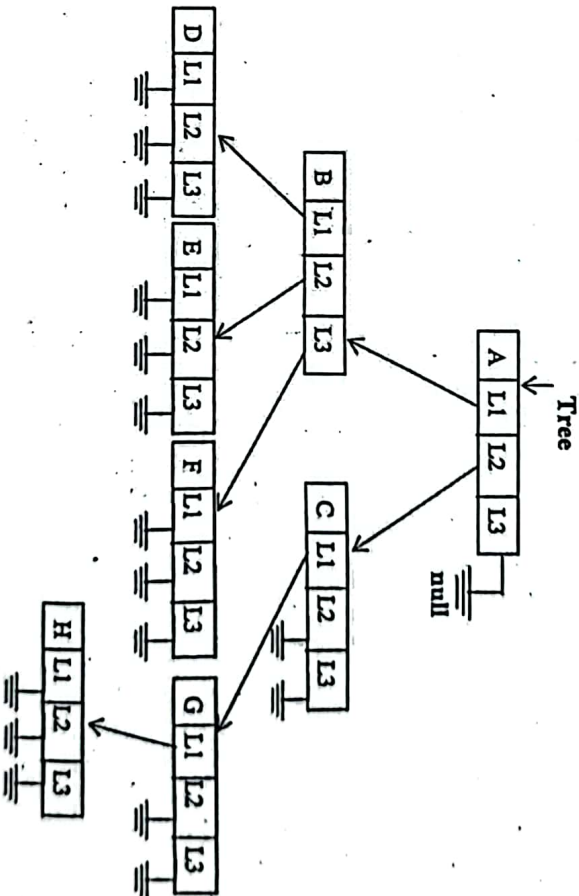
نلاحظ ان عدد الفروع من كل عقدة يختلف واكر الفروع هو ثلاثة للمعدة (B) اذ لها ثلاثة ابناء D, E, F ويكون تعريف جميع عناصر القائمة الموصولة بعدد مؤشرات يساوي (3) لكل عنصر وكالآتي :-

```

struct node
{
    char data;
    node link1, link2, link3;
} *tree;

```

ويمكن تمثيل ذلك بالرسم التالي :



لاحظ ان المؤشر اللزاد في العقدة يشير null
 شكل (6 - 24)

226

2-6-6 تمثيل الاشجار الثنائية Binary Tree Representation

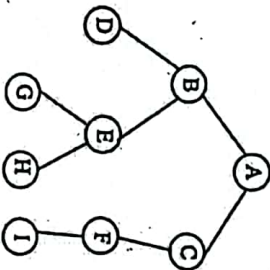
1- استخدام المصفوفة Array Representation

تستخدم مصفوفة احادية بسعة مساوية لأكبر عدد ممكن لعقد الشجرة الثنائية التي ارتفاعها (h) باعتماد العلاقة $(2^h - 1)$ وتخزن القيم البينائية للعقد وفق الاتي:-
1- تخزن عقدة الجذر في الموقع الاول من المصفوفة وليكن T[0] (لأننا نستخدم لغة C++) .

2- نخزن العقد الأخرى بحيث :-

- عقدة الابن الايسر (left child) للعقدة في الموقع (i) تكون في الموقع $(2 * i + 1)$
- عقدة الابن الايمن (right child) في الموقع (i) تكون في الموقع $(2 * i + 2)$
- 3- ان عقدة الابن لاية عقدة في الموقع (i) تكون في الموقع $(i + 1) \text{ DIV } 2$

مثال : فلنأخذ الشجرة التالية :-



الشكل (6 - 27)

بما ان ارتفاع الشجرة $h = 3$ لذا فان اكبر عدد ممكن من العقد في مثل هذه

$$= 2^{h+1} - 1$$

الشجرة سيكون

$$= 2^{3+1} - 1 = 2^4 - 1 = 16 - 1 = 15$$

اذا سعة المصفوفة لتمثيل هذه الشجرة هو (15)

ولكن المصفوفة هي T[15]

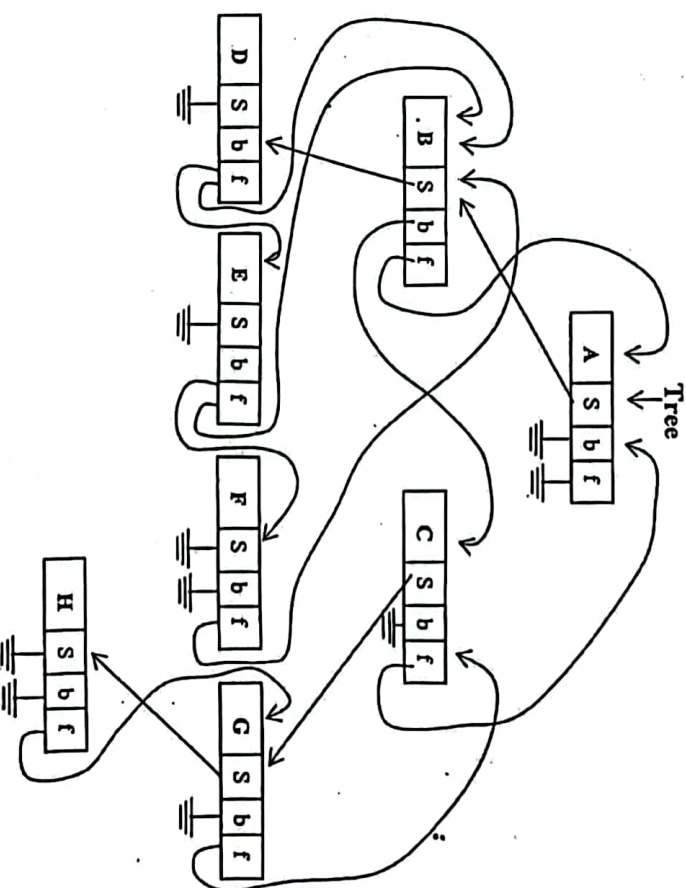
T[15]:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	B	C	D	E	F	-	-	-	G	H	I	-	-	-	-

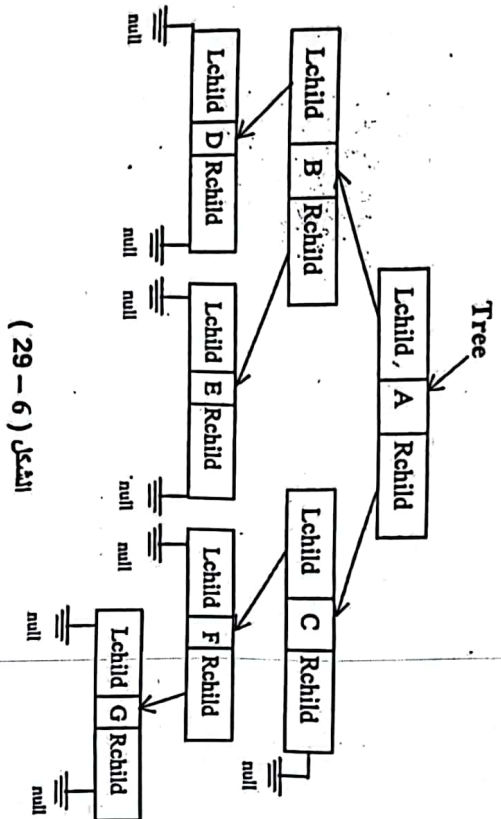
ج / ثلاثة مؤشرات لكل عقدة

يكون لكل عنصر من عناصر القائمة ثلاثة مؤشرات هي :
+ مؤشر يشير الى اكبر الابناء eldest son in the left
+ مؤشر يشير الى الاخ التالي next brother
+ مؤشر يشير الى الاب node's father
وطيه يكون تعريف القائمة الموصولة لنفس الشجرة في المثال السابق الشكل (6 - 23) كالآتي :

```
struct node
{
    char data;
    node son, brother, father;
} *tree;
```



الشكل (6 - 26)

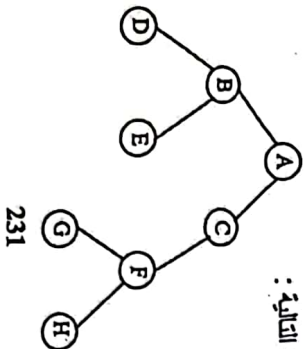


الشكل (6 - 29)

2- تمثيل العقدة بثلاثة مؤشرات (Three pointers)
تعرف كل عقدة تحتوي على مؤشر يشير إلى الإبن الأيسر ومؤشر يشير إلى الإبن الأيمن ومؤشر ثالث يشير إلى العقدة الأب.

```
struct node;
{
    char data;
    node Lch, Rch, f;
} *tree;
```

فيما يلي تمثيل الشجرة التالية :

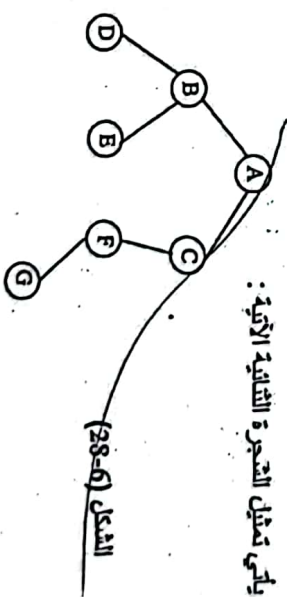


ان عقدة الجذر (A) تخزن في الموقع الاول T[0]
- والعقدة (B) هي الابن الايسر للعقدة (A) تكون في الموقع T[1]
لأن $2 * 1 + 1 = 2 * 0 + 1 = 1$
- والعقدة (C) هي الابن الايمن للعقدة (A) تكون في الموقع T[2]
لأن $2 * 1 + 2 = 2 * 0 + 2 = 2$
- والعقدة (D) هي الابن الايسر للعقدة (B) تكون في الموقع T[3]
لأن $2 * 1 + 1 = 2 * 1 + 1 = 3$ وهكذا تحدد مواقع باقي العقد
ملاحظة: بعد توزيع العقد في المصفوفة يمكن ان نلاحظ ان الوصول إلى العقدة
الابن لأية عقدة مثل (G) التي هي في الموقع T[9] تكون في الموقع T[4] أي العقدة
(E) لأن $2 * 4 + 1 = 9$ DIV 2 = (9-1) T[4]
ب - استخدام Struct

في هذه الطريقة نستخدم القائمة الموصولة (linked list) لتمثيل الشجرة وبعدها اساليب هي:
1- تمثيل العقدة بمؤشرين (two pointers)
حيث يشير أحد المؤشرين إلى الابن الأيسر (left child) والمؤشر الآخر يشير إلى الابن الأيمن (right child)
وتعرف عناصر القائمة الموصولة كالآتي :-

```
struct node;
{
    char data;
    node Lchild, Rchild;
} *tree;
```

وفيما يلي تمثيل الشجرة التالية الآتية :



الشكل (6-28)

البرامج الفرعية للشجرة الثنائية

سبق ان ذكرنا ان هيكل الشجرة فو من الهياكل التي تكون اجزاءها (subtrees) اشجارا ايضا اي ان الجزء يشبه الكل من حيث ان الشجرة للترية هي شجرة ايضا وهنا نستطيع الاستفادة من صيغة الاستدعاء الذاتي (Recursion) في كتابة البرامج الترية (functions) لتمثيل الشجرة والعمليات عليها كالآتي:

برنامج فرعي لاستعراض الشجرة الثنائية بطريقة الترتيب المسبق

(Preorder Traversing)

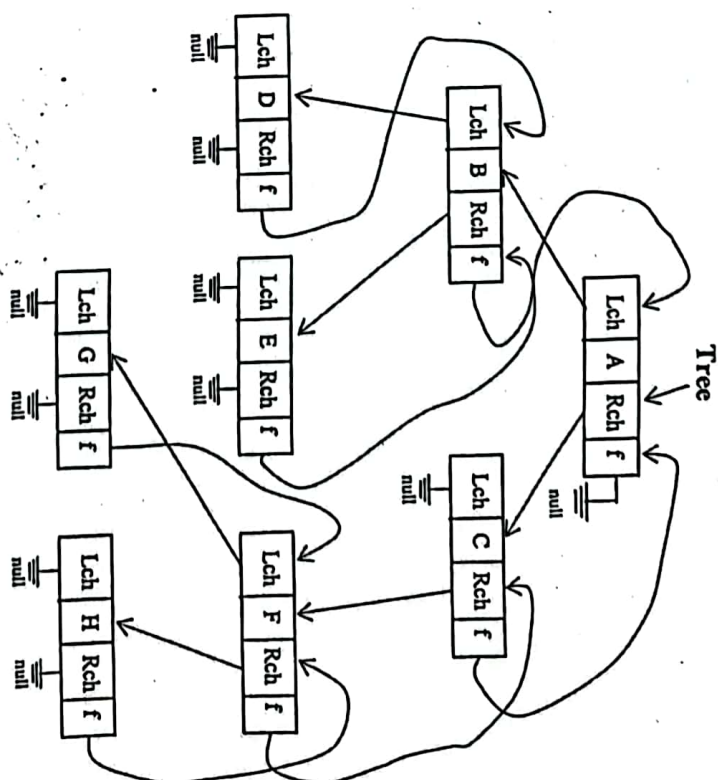
```
void preorder (Bnode *p );
{
    if (p!=0)
    {
        cout<<(p->data) ;
        preorder (p->left) ;
        preorder (p->right) ;
    }
}
```

من الملاحظ ان هذا البرنامج التري يعكس خطوات الخوارزمية المتبعة بمرسح (الجذر / الفرع الايسر / الفرع الايمن) وكذا ان هذه الخطوات عند كل عقدة باعتمادها شجرة فرعية وله اذا فانه برنامج فرعي ذاتي التكرار (Recursive Function).

برنامج فرعي لاستعراض الشجرة الثنائية بطريقة الترتيب اللاحق

(Postorder Traversing)

```
void postorder (Bnode *p );
{
    if (p!=0)
    {
        postorder (p->left) ;
        postorder (p->right) ;
        cout<<(p->data) ;
    }
}
```



الشكل (6 - 30)

برنامج فرعي: بصفحة التكرار (Iteration) لاستعراض (مسح) عقد الشجرة
الثانية بطريقة الترتيب البيني (Inorder Traversing).

Void Iterationinorder (Bmode *root)

```
{
    int top=0;
    p=root;
    while( p!=NULL && top!=0);
    {
        while( p!=NULL )
        {
            push( stack,top,p);
            p = p->left;
        }
        If ( top!=0 )
        {
            pop( stack,top,p);
            cout<<(p->data);
            p = p->right;
        }
    }
}
```



ان هذا البرنامج الفرعي هو (Recursive Function) لتمثيل خطوات الخوارزمية
(الفرع الايسر / الفرع الايمن / الجذر) المتكررة عند كل عقدة (باعتبارها شجرة
فرعية).

برنامج فرعي لاستعراض الشجرة الثانية بطريقة الترتيب البيني

(Inorder Traversing)

```
void inorder (Bmode *p );
{
    if ( p!=0 )
    {
        inorder ( p->left);
        cout<<(p->data);
        inorder ( p->right);
    }
}
```

و هذا البرنامج هو ايضا من نوع (recursive function) لتمثيل خطوات
الخوارزمية (الفرع الايسر / الجذر / الفرع الايمن) المتكررة عند كل عقدة (باعتبارها
شجرة فرعية).

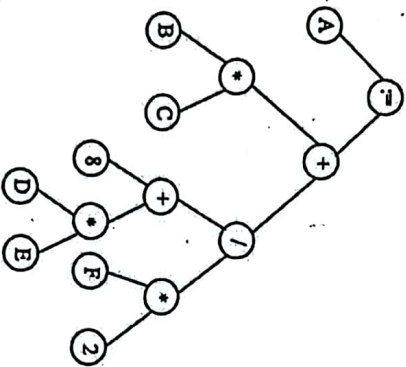
7-6 تمثيل التعبيرات الحسابية باستخدام الشجرة الثنائية

Representation of Arithmetic Expressions using Binary Tree

من التطبيقات المهمة للأشجار الثنائية هو استخدامها في تمثيل التعبير الحسابي إذ أن العملية الحسابية (+ , - , * , /) تمثل بعبارة مقترعة أما العوامل الحسابية فتمثل بالأوراق مع ملاحظة أن مستويات الشجرة تعكس أساليب تنفيذ العمليات الحسابية في ذلك التعبير الحسابي .

مثال : استخدم الشجرة الثنائية لتمثيل التعبير الحسابي التالي :

$$A := B * C + (8 + D * E) / (F * 2)$$



الشكل (6 - 31)

عند استعراض (مسح) هذه الشجرة بكل من :-

1- طريقة الترتيب البيني (inorder) فنحصل على التعبير الحسابي نفسه وهو

بصيغة (Infix Notation) .

2- طريقة الترتيب السابق ونحصل على صيغة (prefix notation) للتعبير الحسابي

أي :-

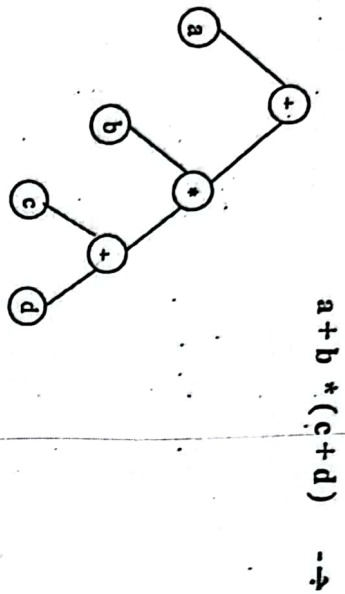
$$:= A + * B C / + 8 * D E * F 2$$

برنامج فرعي : بصيغة الاستدعاء الذاتي (Recursion) لاحتساب عدد الأوراق في الشجرة الثنائية (No. of Leaves) .

```
void Leaves(Btnode *p)
{
    if (p != 0)
    {
        if (p->left==0) && (p->right==0)
        {
            Count++;
            Leaves(p->left);
            Leaves(p->right);
        }
    }
}
```

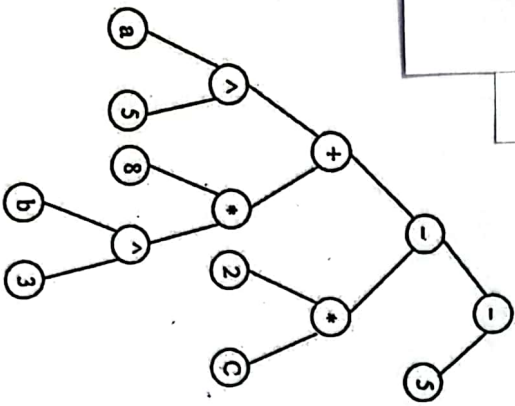
برنامج فرعي : بصيغة الاستدعاء الذاتي (Recursion) لمبادلة (Swap) قيمة الفصص في الفرع الأيسر مع قيمة العنصر في الفرع الأيمن لكل عقدة في الشجرة الثنائية .

```
void swapnodes(Btnode *p)
{
    Bnode *q;
    if (p != 0)
    {
        q = p->left;
        p->left = p->right;
        p->right = q;
        swapnodes(p->left);
        swapnodes(p->right);
    }
}
```



الشكل (6 - 34)

$a \wedge 5 + 8 * b \wedge 3 - 2 * c - 5$ - 1



الشكل (6 - 35)

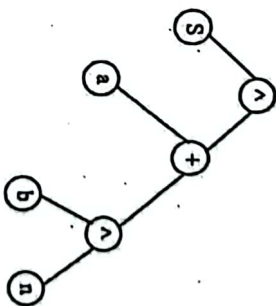
وتسمى أيضاً صيغة (Polish Notation) للتعبير الحسابي
3- طريقة الترتيب اللاحق ونحصل على صيغة (postfix notation) للتعبير الحسابي
أي

$ABC * 8 DE * + F 2 * / =$

وتسمى أيضاً صيغة (Reverse Polish Notation)

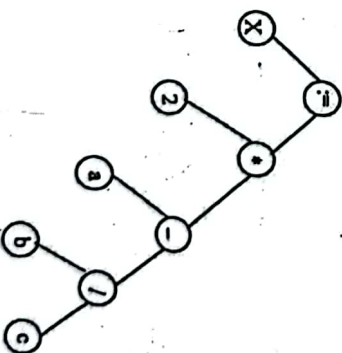
تبرين : استخدم الشجرة الثنائية لتحويل كل من التعبيرات الحسابية التالية :-

1- $S \wedge (a + b \wedge n)$



الشكل (6 - 32)

ب- $X := 2 * (a - b / c)$

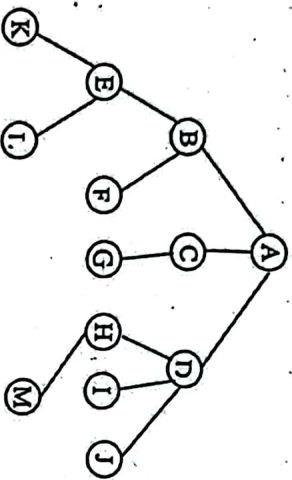


الشكل (6 - 33)

هيكل الشجرة والمفاهيم الرياضية

Tree Structure and Mathematical concepts

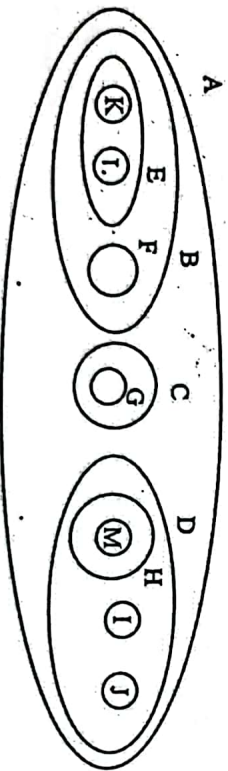
يمكن استخدام هيكل الشجرة لتمثيل بعض المفاهيم الرياضية. وبالعكس يمكن التعبير عن هيكل الشجرة بمفاهيم رياضية ..
لنأخذ الشجرة التالية :-



الشكل (6 - 37)

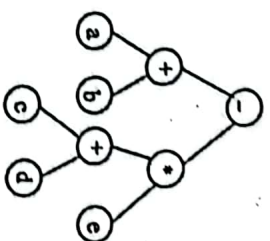
أ - ان هذه الشجرة يمكن التعبير عنها رياضيا باستخدام مخططات فن

كالاتي (Venn Diagrams) :-



الشكل (6 - 38)

$$a + b - [(c + d) * e]$$



الشكل (6 - 36)

تمرين :

1- ارسم الشجرة الثنائية التي تمثل التعبير الحسابي التالي :-

$$M := L \wedge (2-b) - T \wedge (3*b*5) / 4 * P$$

2- ما هو عدد العقد المتفرعة فيها ..

3- ما هو عدد العقد التي درجتها (0)

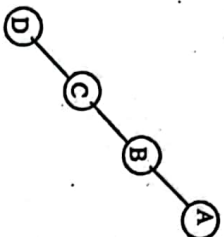
4- اقطع من اسفل الشجرة لتحصل على شجرة ثنائية ارفقها (3)

5- استعرض الشجرة الثنائية المتكونة بموجب الفترة (4) بطريقتي الترتيب السابق

والترتيب اللاحق .

6- استخدم المصفوفة لتمثيل الشجرة المتكونة بموجب الفترة (4) اعلاه .

ب - الترتيب اللاحق (Postorder) والترتيب البيني (Inorder)
الجواب : هي الأشجار التي لا تحتوي أي من عقدها على فرع اليمن أي ان
RLink = null كما في الشكل



ج - الترتيب السابق (Preorder) والترتيب اللاحق (Postorder)
الجواب : هي الأشجار التي تتكون من عقدة واحدة تمثل الجذر وليس لها فرع اليمن او
فرع يسر وهذا يعني من الناحية البرمجية :

R Link : = null , L link : = null

8-6 تحويل الشجرة الاعتيادية الى شجرة ثنائية

Transformation of a tree into a Binary tree

لتحويل الشجرة الاعتيادية الى شجرة ثنائية تتبع خطوات الخوارزمية التالية :

- 1- جذر الشجرة الاعتيادية يصبح هو جذر الشجرة الثنائية .
- 2- الابن الايسر left child للشجرة الثنائية يكون نفسه الابن الايسر من الشجرة الاعتيادية
- 3- ان اخوة (brothers) هذا الابن الايسر في الشجرة الاعتيادية (الاعتيادية) يصبحون النسخ (الابن) الايمن له في الشجرة الثنائية
- 4- نميز نفس الخطوات واعتبار الابن الايسر هو الجذر

ب - ويمكن استخدام الأقواس المداخلة (Nested Parenthesis) للتعبير عن هذه
الشجرة

A (B (E (K, L), F), C (G), D (H (M), I, J))

ج - وقد نستعمل الشجرة لعكس هيكمل ما (file structure) موزل من قود
تحتوي عدة حقول مختلفة كما توصف في لغة كوبرول في قسم البيانات
(Data Division)

01 A - Record

02 B - field 1

03 E - field

04 K - field

04 L - field

03 F - field

02 C - field

03 G - field

02 D - field

03 H - field

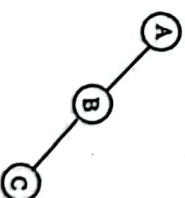
04 M - field

03 L - field

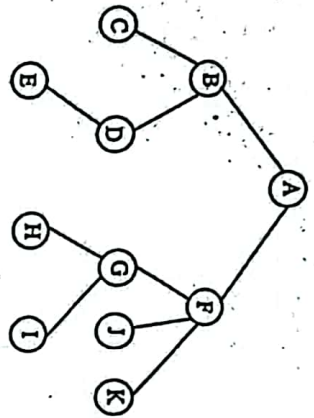
03 J - field

تمرين : ماهي صفات الاشجار الثنائية التي تغطي نفس المخرجات عند استعراضها
تدوين (Traversed) بكل من طريقي :

أ - الترتيب السابق (Preorder) والترتيب البيني (Inorder)
للجواب : هي الاشجار التي لا تحتوي أي من عقدها على فرع يسر أي ان
left link = null كما في الشكل

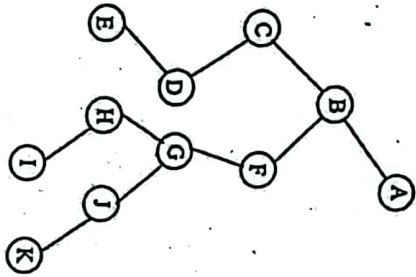


مثال : حول الشجرة الاعتيادية اثنائه الى شجرة ثنائية



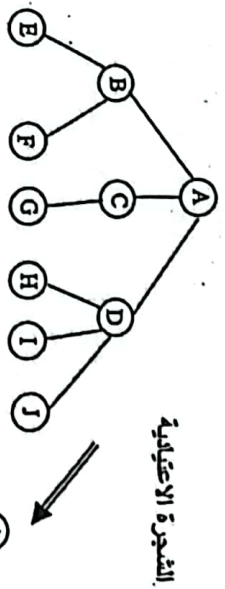
الشجرة الاعتيادية

الشجرة الثنائية

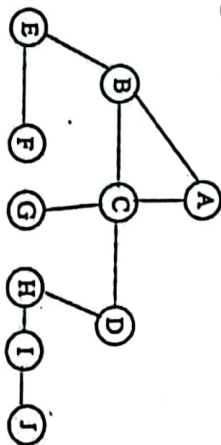


الشكل (6 - 40)

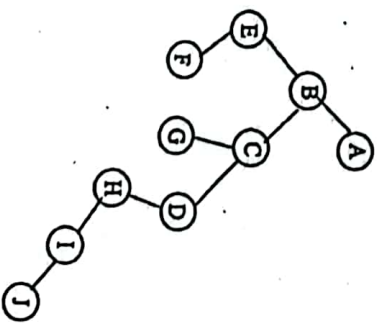
مثال : لناخذ الشجرة الاعتيادية اثنائه ونحولها الى شجرة ثنائية



الشجرة الاعتيادية



الشجرة الثنائية



الشكل (6 - 39)

تمرين : ارسم شجرة البحث التالي للعناصر التالية :

20, 4, 6, 12, 8, 3, 7, 9, 5.

الحواب :

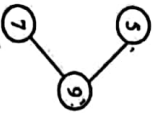
1- نأخذ العنصر الاول (5) ونعتبر العقدة الجذر (5)

(5)

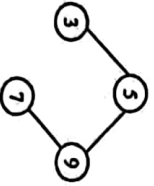
2- نأخذ العنصر التالي (9) ولكنه اكبر من الجذر (5) فيكون هو فرعا ايما للجذر



3- نأخذ العنصر التالي (7) وهو اكبر من عنصر الجذر (5) فنذهب للفرع اليمين ولجده اصغر من (9) فيكون فرعا ايسر للعقدة (9)



4- نأخذ العنصر (3) وهو اصغر من عنصر الجذر (5) فنضعه في يساره



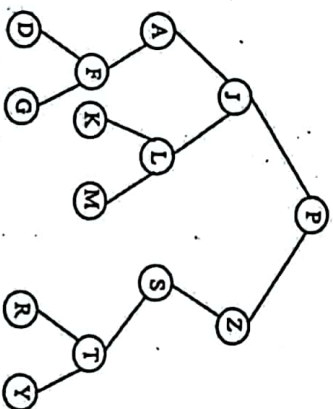
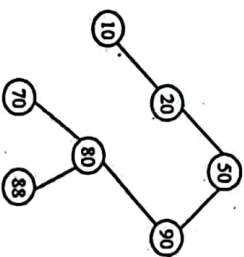
247

9-6 شجرة البحث الثنائية Binary Search Tree

هي شجرة ثنائية مرتبة تكون قيمة عنصر الفرع (الابن) الايسر لاية عقدة فيها هو اقل من قيمة عنصر تلك العقدة باعتبارها الاب (father) وتكون قيمة عنصر الفرع (الابن) الابن اكبر من قيمة عنصر العقدة (الاب)

او بعبارة اخرى :

هي شجرة ثنائية مرتبة (Binary ordered type) تكون القيمة البينائية لاية عقدة فيها هي اكبر من القيمة البينائية للفرع الايسر واصغر من القيمة البينائية للفرع اليمين كما في الامكان الآتية :

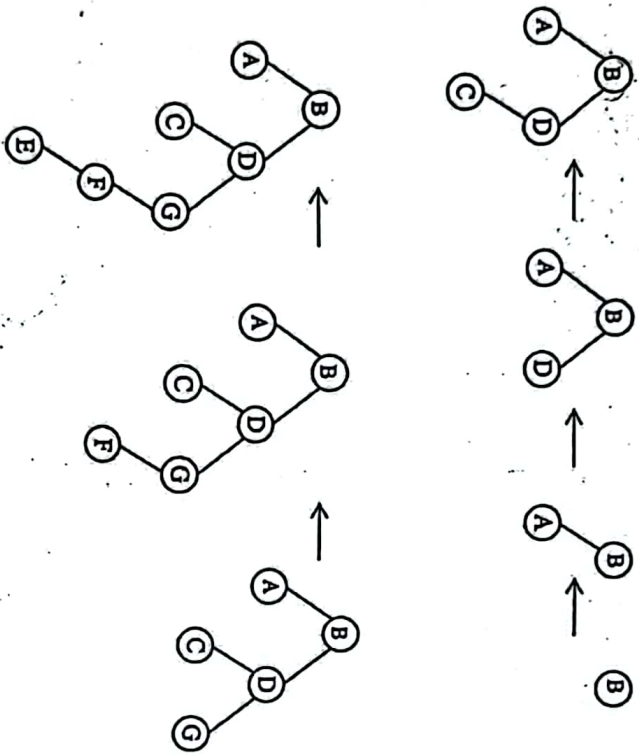


الشكل (6 - 41)

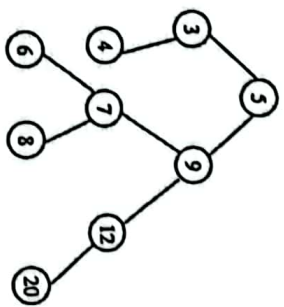
246

تمرين : ارسم شجرة البحث الثنائية للعناصر التالية : A, B, C, D, E, F, G.

الجواب :-

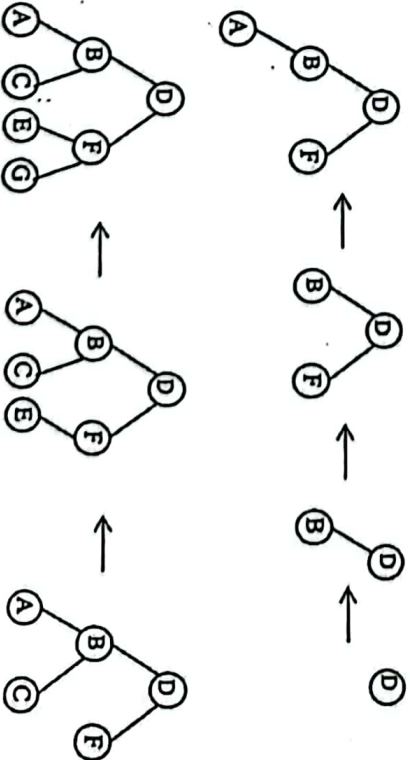


5- فستمر بهذه الطريقة بأخذ العنصر الجديد ومقارنته مع عناصر الشجرة ابتداءً من الجذر ومنفصل على الشجرة وبشكلها النهائي



تمرين : ارسم شجرة البحث الثنائية للعناصر التالية : D, B, F, A, C, E, G.

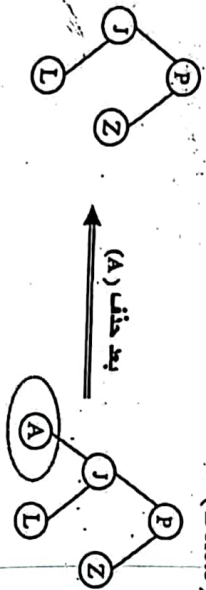
الجواب :



خطوات حذف عنصر من شجرة البحث الثنائية

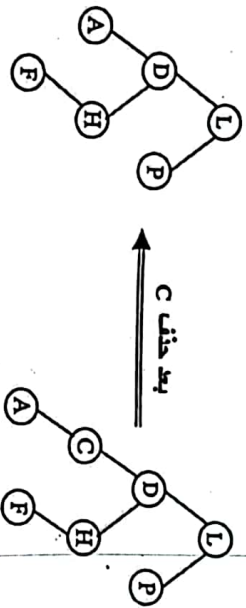
أ- حذف عقدة نهائية (ورقة)

- 1- نأخذ العقدة ونجعل قيمة مؤشر عقدة الاب اليها (null)
- 2- نلغي (Delete) العقدة



ب- حذف عقدة لها فرع (ابن) واحد

- 1- نجعل مؤشر اب العقدة (node father) ونشير الى العقدة الابن
- 2- نلغي (Delete) العقدة المقصودة



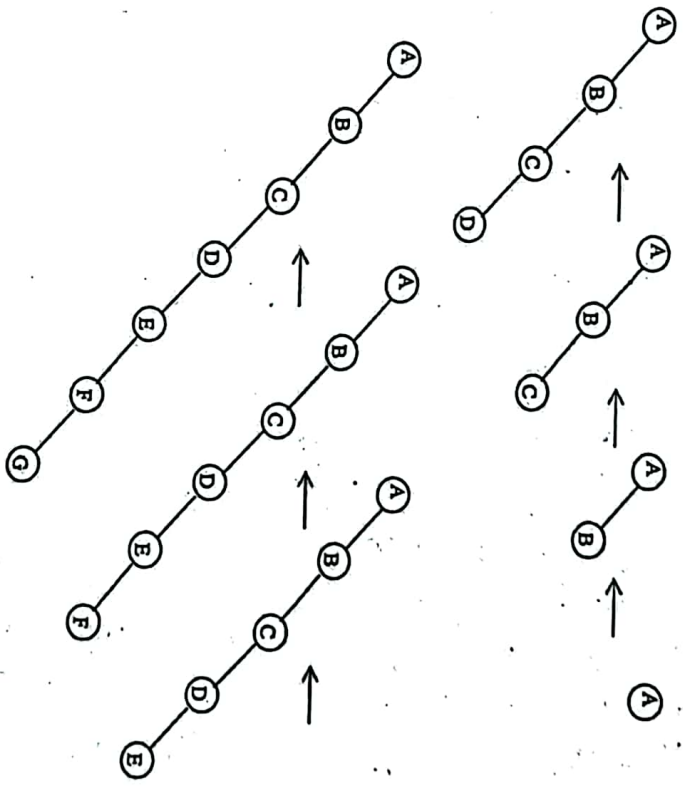
ج- حذف عقدة لها فرعان

- 1- نستبدل العقدة المطلوب حذفها بالعقدة التالية لها بالقيمة وهذه تستحصل من الشجرة الفرعية اليسرى او الشجرة الفرعية اليمنى بالنسبة للعقدة .
- 2- نأخذ الشجرة الفرعية اليسرى للعقدة (أي العقدة التي في يسار العقدة المطلوب حذفها)

- + اذا لم يكن لها فرع اليمنى فأنها تصبح الجديل
- + اذا كان لها فرع اليمنى فأننا نأخذ العقدة في أقصى اليمين لتصبح الجديل

تمارين : ارسم شجرة البحث الثنائية للعناصر التالية : A, B, C, D, E, F, G

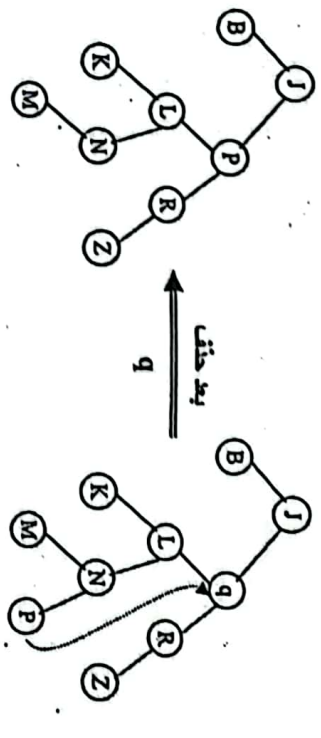
الاجاب :-



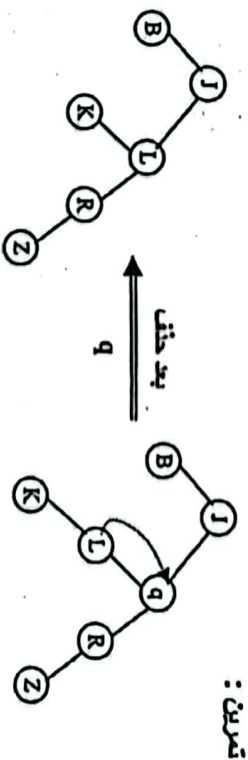
برنامح فرعي : بصفة التكرار (Iteration) لإيجاد عقدة في شجرة البحث الثنائية
(Binary Search Tree) .

```
int Findnode(Btnode *p, int value)
{
    while ( ( p!=0 ) && ( found==0 ) )
    {
        if( p->data==value )
            found = 1 ;
        else
        {
            if ( p->data > value )
                p=p->left;
            else
                p=p->right;
        }
    }
    return found;
}
```

253



لحذف العقدة (q) التي لها فرعان (L), (R)
نجد ان الشجرة الفرعية اليسرى (L) لها فرع ايمن ولهذا فنأخذ أقصى ويمين (L)
وهو (P) لتصبح بديل العقدة (q)



لحذف العقدة (q) التي لها فرعان (L), (R).
نجد ان الشجرة الفرعية اليسرى (L) ليس لها فرع ايمن ولهذا فلنأخذ أقصى اليمين
أي ان العقدة (L) تحل في مكان العقدة (q)

252

```

    {
        if (p != 0)
        {
            inorder(p->left);
            cout<<(p->data)<<" ";
            inorder(p->right);
        }
        void postorder(Btnode *p)
        {
            if (p != 0)
            {
                postorder(p->left);
                postorder(p->right);
                cout<<(p->data)<<" ";
            }
        }
        void insert(Btnode *root2, Btnode *q)
        {
            if (root2->right != NULL && q->data > root2->data)
            {
                root2 = root2->right;
                insert(root2, q);
            }
            else
            {
                if (root2->left != NULL)
                {
                    root2 = root2->left;
                    insert(root2, q);
                }
                else
                {
                    if (x->data < root2->data)
                    {
                        root2->left = q;
                    }
                }
            }
        }
        void swapnodes(Btnode *p)
        {
            Btnode *q;
            if (p != 0)
            {
                q = p->left;
                p->left = p->right;
                p->right = q;
                swapnodes(p->left);
                swapnodes(p->right);
            }
        }
        int Findnode(Btnode *p, int value)
        {
            while (p != 0 && (found == 0))

```

255



برنامج الفصل
تمثيل شجرة البحث الثنائية (Binary Search Tree)
و عمليات الأضافة فيها .
برنامج رقم - 27 :

```

#include <iostream>
using namespace std;
struct Btnode
{
    int data;
    Btnode *left, *right;
} *root;
int found, int count;
class Binarytree
{
public:
    Btnode *p, *p1, *q, *root; Btnode *x;
    int firstnode;
    int value;
    Binarytree()
    {
        root = NULL;
    }
    void Addnode(Btnode *root2, Btnode *p)
    {
        int D;
        D = p->data;
        if (root2 != NULL)
        {
            if (D < root2->data)
            {
                if (root2->left == NULL)
                {
                    root2->left = p;
                }
                else
                {
                    Addnode(root2->left, p);
                }
            }
            else
            {
                if (root2->right == NULL)
                {
                    root2->right = p;
                }
                else
                {
                    Addnode(root2->right, p);
                }
            }
        }
    }
    void preorder(Btnode *p)
    {
        if (p != 0)
        {
            cout<<(p->data)<<" ";
            preorder(p->left);
            preorder(p->right);
        }
    }
    void inorder(Btnode *p)

```

254

```

        BTAddnode(root,q1);
    }
    break;
case 2:cout<<"\n The Output Of The Preorder Traversing is:";
    BTpreorder(root);
    break;
case 3:cout<<"\n The output Of The Inorder Traversing is:";
    BTinorder(root);
    break;
case 4:cout<<"\n The Output Of The Postorder Traversing is:";
    BTpostorder(root);
    break;
case 5:if(root;
    x=new BNode;
    cout<<"\n To Insert a New Node to The Binary Tree ";
    cout<<"\n Enter The Data Value Of The new Node ";
    cin>>x->data;
    x->left=NULL;
    x->right=NULL;
    BTinsert(H,x);
    break;
case 6:if(root;found=0;
    int x2;
    cout<<"\n Enter The Data Value Of the Node required ":";
    cin>>x2;
    BT.Findnode(H,x2);
    if(found==1)
        cout<<"\n The node required Exist in the tree ";
    else cout<<"\n The node required is NOT Exist in the tree ";
    break;
case 7:if(root;count=0;
    BT.Leaves(H);
    cout<<"\n The No. of leaves in this Tree is : ";
    cout<<count;
    break;
case 8:BT.swapnode(root);
    cout<<"\n The tree has been swapped the left & right branches of ALL its nodes ";
    cout<<"\n You can select any of traversing options (2,3,4) to show the results";
    break;
} // end of switch
} // end of DO
while(choice !=9);
return 0;
} // end of main

```

257

```

    { if(p->data==value)
        found=1;
    }
    else
    { if(p->data > value)
        p=p->left;
    }
    else p=p->right;
}
return found;
}
void Leaves(BNode *p)
{ if(p!=0)
    { if(p->left==0 && (p->right==0))
        count++;
    Leaves(p->left);
    Leaves(p->right);
    }
}
int main()
{
    BNode *T,*H,*q1,*x; int choice,m;
    Binarytree BT;
    do
    {
        cout<<"\n"; cout<<"\n";
        cout<<"\n Representation Of The Binary Tree ";
        cout<<"\n";
        cout<<"\n";
        cout<<"\n 1 - Creation of the TREE ";
        cout<<"\n 2 - Preorder Traversing Output ";
        cout<<"\n 3 - Inorder Traversing Output ";
        cout<<"\n 4 - Postorder Traversing Output ";
        cout<<"\n 5 - Inserting One Node To The Tree ";
        cout<<"\n 6 - Find a Node in the TREE ";
        cout<<"\n 7 - Count No of leaves in the tree ";
        cout<<"\n 8 - SWAP the Left & Right branches of all nodes in the tree ";
        cout<<"\n 9 - Exit ";
        cout<<endl;
        cout<<"\n Select Your Choice : ";
        cin>>choice;
        switch(choice)
        {
            case 1:cout<<"\n Howmany Nodes You Like To Create in This Binary search Tree ? ";
                cin>>m;
                for( int i=1; i<=m; i++)
                {
                    q1=new BNode;
                    cout<<"\n Enter the data value of the node : ";
                    cin>>q1->data;
                    q1->left=NULL;
                    q1->right=NULL;
                    if(i==1)
                        root=q1;
                    else

```

256

c- Give the record representation using two pointers
d- Give the record representation using three pointers

5. the following figure shows an array representation for a binary tree

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
X	M	F	R	-	T	N	-	A	-	-	Y	-	-	W

a- Draw this tree

b- Traverse the tree using preorder, postorder, inorder methods:

6. Write a procedure to traverse the binary tree using

a - preorder method
b- postorder method
c- inorder method

7. Write an algorithm to transfer the general tree into binary tree

8. Write a procedure to find a node (with certain value) in the binary search tree .

9. The following figure shows an array representation for a

binary tree

0	1	2	3	4	5	6	7	8	9
L	M	X	X	A		B	S	R	

14	15	16	17	18	19
	N		T	P	F

29	30
	K

a- Draw this tree

b- Traverse this tree in postorder

c- Cut from this tree the right branch (subtree) of node(A)

and represent the remaining tree using record representation with three pointers .

10. Explain how a dynamically allocated record can be used to represent the binary tree ?

what are the advantages and disadvantages of this representation over the array representation .

11. Draw the binary search tree for the following sequence of keys:

259

أسئلة الفصل

1. Draw the following tree :

height = 5 ,

No . of nodes > = 17

At least two nodes of degree 3

a- Represent this tree using two ways

+ Venn diagrams

+ Nested parenthesis

b- Traverse the tree in Preorder and Postorder

c- Transfer the above tree into binary tree

d- Traverse the resulting binary tree using inorder method

e- Represent the resulting binary tree using array

2. Draw the binary trees which represents the following

arithmetic expressions :

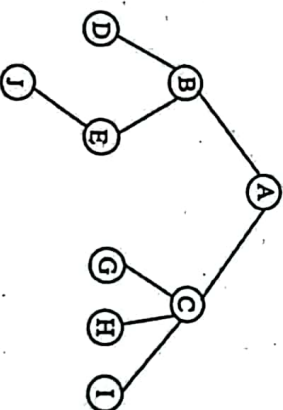
a- $R * T \wedge 2 - M * (L - E / F)$

b- $a/b + c/d \wedge 2 * (e - f)$

c- $A := b * c + (3 - d * e) / f \wedge 4$

3. Write an algorithm for postorder traversing of the tree

4. Given the following tree



a- Represent the tree using two pointers

b- Represent the tree using three pointers

258

الفصل السابع الترتيب والبحث

Sorting and Searching

Sorting

الترتيب 1-7

الغرض من الترتيب 1-1-7

خطوات عملية الترتيب 2-1-7

أنواع خوارزميات الترتيب 3-1-7

أ- الترتيب الداخلي

ب- الترتيب الخارجي

Internal Sort 4-1-7

External Sort 4-1-7

العوامل الرئيسية المحددة لاختيار خوارزمية الترتيب

Insertion Sort 5-1-7

Selection Sort 6-1-7

Bubble Sort 7-1-7

Quick Sort 8-1-7

Heap Sort 9-1-7

Merge Sort 10-1-7

Searching 2-7

أنواع خوارزميات البحث 1-2-7

Sequential Search 2-2-7

Binary Search 3-2-7

80, 10, 44, 25, 30, 8, 96, 70, 90, 33, 100
and perform the following operations in sequence showing the tree after each operation

a- Insert 75, 20

b- Delete 44

c- Insert 72, 92

d- Delete 8, 100

12. Put true or false for each of the following :

a- the terminal nodes of a binary tree occur in the same relative position in preorder, inorder and postorder traversing [True or False].

b- Show that if we are given the preorder and inorder of the nodes of a binary tree, can the binary constructed.

c- Show that if we are given the postorder and inorder of the nodes of a binary tree, can the binary constructed.

d- Show that if we are given the preorder and postorder of the nodes of a binary tree, can the binary constructed.

e- The terminal nodes of a tree (not binary) occur in the same relative position in preorder & postorder.

13. Show that how we can construct (draw) the binary tree if we are given the post order & in order traversing output.

1-7 الترتيب Sorting

هي عملية ترتيب مجموعة من العناصر البينائية وفق قيمة حقل (أو حقول) يسمى المفتاح (key) بصورة تصاعدية (ascending) أو بصورة تنازلية (descending).

1-1-7 الغرض من الترتيب

تعدد اغراض عملية الترتيب واهمها :

- زيادة كفاءة خوارزمية البحث عن عنصر ما.
- تبسيط معالجة الملفات :
- لحل مشكلة تشابه التويد.

2-1-7 خطوات عملية الترتيب

تتلخص خطوات خوارزميات الترتيب بكافة انواعها بالمراحل التالية :

- 1- قراءة حقل المفتاح .
- 2- الانتقال (استنتاج) موقع العنصر في الترتيب الجديد .
- 3- نقل العنصر البينائي الى موقعه الجديد.

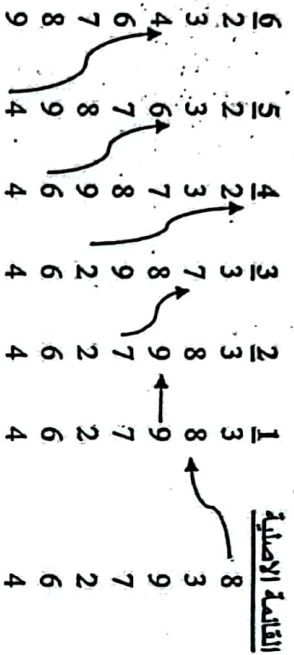
3-1-7 انواع خوارزميات الترتيب

أ- الترتيب الداخلي Internal Sort

وهو الترتيب الذي يحدث في الذاكرة الرئيسية للحاسوب (main memory) عندما يكون حجم البيانات مناسباً (ليس كبيراً) للفرن في الذاكرة. ومن أهم أنواعه :-

Insertion Sort	* ترتيب الإدخال
Selection Sort	* الترتيب بالاختيار
Bubble Sort (exchange)	* ترتيب الفقاعة
Quick Sort	* الترتيب السريع
Heap Sort	* الترتيب الكرومي
Merge Sort	* ترتيب الدمج
Radix Sort	* ترتيب الأساس
Shell Sort	* ترتيب شيل

مثال: نرتب عناصر القائمة التالية تصاعدياً.



عدد عناصر القائمة $n = 7$

عدد المراحل $n-1 = 6$

أشارة السهم ($\leftarrow$) توضح العنصر الذي اختيف.

ملاحظات:

- معدل عدد المقارنات Average no. of comparisons هو $(n^2/4)$ حيث (n) يمثل عدد عناصر القائمة.
- معدل عدد التبادلات Average no. of exchanges هو $(n^4/4)$.
- أن خطوات هذا الترتيب تتضمن الكثير من حالات التبدل خصوصاً عندما تكون البيانات غير مرتبة أو معكوسة الترتيب ، ولذا فإنه من غير المفيد استخدامه عندما تكون البيانات في قائمة مرتبطة (linked list) لأنه سيتطلب الكثير من تغيير المؤشرات (pointers) جدد كل تبدل (comparison) . ويفضل استخدامه في حالة البيانات الشبه مرتبة ومخزونة في مصفوفة (array).

بيد الترتيب الخارجي External Sort

وهو ترتيب البيانات المخزنة في اوساط الفرز الثانوية على شكل ملفات عندما يكون حجم البيانات كبير جداً بحيث يعجز استيعابها كلها في الذاكرة في وقت واحد أثناء عملية الترتيب ومن أهم أنواعه:

- * الترتيب بالدمج في المسارين
- * K-way - Merge Sort
- * الترتيب بالدمج المتوازن في المسارين
- * Balanced Two-way - Merge
- * الترتيب بالدمج متعدد الاطوار
- * Polypase Two-way - Merge

4-1-7 العوامل الرئيسية المحددة لاختيار خوارزمية الترتيب

ان لاختيار أي من خوارزميات الترتيب يجب ان يكون في ضوء عدد من العوامل من أهمها:

- 1- حجم البيانات المخزونة.
- 2- نوع الفرز (الذاكرة الرئيسية، قرص، شريط).
- 3- درجة ترتيب البيانات (غير مرتبة، شبه مرتبة).

5-1-7 الترتيب بالإضافة بالإنشاء Insertion Sort

تتلخص خطوات هذه الخوارزمية بما يلي:

1. تقارن العنصر الثاني (3) مع العنصر الأول (8) ، ولأن العنصر الثاني هو الأصغر يتم وضعه في الموقع الأول ونضع (8) في الموقع الثاني.
2. في الخطوة التالية تقارن العنصر في الموقع الثالث (9) مع العنصر في الموقع الثاني (8) الذي يسبقه ونجده ليس الأصغر، فنتركه ويبقى في الموقع الثالث.

3. تقارن العنصر في الموقع الرابع (7) مع العنصر في الموقع الثالث الذي يسبقه وهو (9) فنجد أنه أصغر منه فيتم استبدالهما [ليصبح (7) في الموقع الثالث، (9) في الموقع الرابع] ، ثم نستمر بمقارنة العنصر (7) بموقعه الجديد الثالث مع العنصر في الموقع الثاني الذي يسبقه وهو (8) فنجد أنه أصغر منه ويتم استبدالهما أيضاً [ليصبح (7) في الموقع الثاني، (8) في الموقع الثالث] . وعند مقارنة العنصر (7) بموقعه الجديد الثاني مع العنصر في الموقع الأول الذي يسبقه نجده ليس أصغر منه فنتركهما في موقعيهما.

4. وهكذا تستمر العملية بالمرور على جميع العناصر في المواقع التالية الخمس و السادس و السابع، إذ تنتهي عملية الترتيب.

مثال: رتب القائمة التالية تصاعدياً.
القائمة الأصلية

4	6	2	7	9	3	8	
6	5	4	3	2	1		القائمة الأصلية
2	2	2	2	2	2	8	
3	3	3	3	3	3	3	
4	4	4	4	4	4	9	
6	6	6	6	6	6	7	
7	7	7	7	7	7	2	
8	8	8	8	8	8	6	
9	9	9	9	9	9	4	

عدد عناصر القائمة $n = 7$
عدد المراحل $n-1 = 6$ no. of passes

ملاحظات: معدل عدد المقارنات Average No. of comparisons هو

$$\frac{n}{2} * (n-1)$$

معدل عدد التبادلات Average No. of exchanges هو $(n-1)$.

إعدادات برمجية (Class) للترتيب بالاضافة

```
template<class T>
void insertionsort (T data[], int n)
{
    for(int i=1, j; i<n; i++)
    {
        T item = data[i];
        for(j=i; j>0 && item < data[j-1]; j--)
            data[j] = data[j-1];
        data[j] = item;
    }
}
```

ملاحظة: استخدم الرمز T ليغير الى النوع البياني للمعلومات المطلوب ترتيبها ان كانت (int, char, ...). فمثل (int) بدلاً من (T) عندما تكون البيانات من النوع integer وهكذا لغير ذلك من الأنواع البيانية.

6-1-7 الترتيب بالاختيار Selection Sort

ويتلخص خوارزمية هذا الترتيب بالخطوات الآتية:

- 1- إيجاد أصغر عنصر في القائمة واستبداله من موقعه مع العنصر في الموقع الأول في القائمة.
- 2- إيجاد أصغر عنصر في المتبقي من القائمة واستبداله من موقعه مع العنصر في الموقع الثاني في القائمة.
- 3- نمتص في هذه العملية لحين الوصول الى نهاية القائمة.

مثال: لقم بترتيب القائمة 2,7,9,3,8 تصاعديا

المرحلة الرابعة	المرحلة الثالثة	المرحلة الثانية	المرحلة الاولى	القائمة الاصلية
1 2 3 4 5	1 2 3 4 5	1 2 3 4 5	1 2 3 4 5	1 2 3 4 5
2 3 4 5 1	2 3 4 5 1	2 3 4 5 1	2 3 4 5 1	2 3 4 5 1
3 4 5 1 2	3 4 5 1 2	3 4 5 1 2	3 4 5 1 2	3 4 5 1 2
4 5 1 2 3	4 5 1 2 3	4 5 1 2 3	4 5 1 2 3	4 5 1 2 3
5 1 2 3 4	5 1 2 3 4	5 1 2 3 4	5 1 2 3 4	5 1 2 3 4

لاحظ ان عدد العناصر في القائمة هو ($n = 5$) وعدد المراحل ($n-1 = 4$)
 وعدد الخطوات في كل مرحلة يتناقص بمقدار واحد عن عدد خطوات المرحلة السابقة لها.

ملاحظات :

- معدل عدد المقارنات comparisons هو ($n^2/2$)
- حيث (n) يمثل عدد عناصر القائمة .
- معدل عدد التبادلات exchanges هو ($n^2/4$)
- هذه الطريقة جيدة عندما تكون العناصر شبيه مرتبة وعددها ليس كبيرا ، ولاحتياج الى مساحة خزنية كبيرة وبسيطة .
- ان وقت التنفيذ يبلغ $O(n^2)$.

إيعازات برمجية (Class) لترتيب القائمة

```
template <class T>
void BubbleSort(T data[], int n)
{
    for(int i=0 ; i<n-1 ; i++)
        for(int j = n-1 ; j>i ; j--)
            if(data[j] < data[j-1])
            {
                T item = data[j] ;
                data[j] = data[j-1] ;
                data[j-1] = item ;
            }
}
```

إيعازات برمجية (Class) لترتيب بالاختيار

```
template <class T>
void selectionsort ( T data[], int n)
{
    for(i=0 , j , pos ; i<n-1 ; i++)
    {
        for(j=i+1 , pos=i ; j<n ; j++)
            if (data[j] < data[pos])
                pos=j ;
        T item = data[pos] ;
        data[pos] = data[i] ;
        data[i] = item ;
    }
}
```

7-1-7 ترتيب القائمة Bubble Sort

ان فكرة هذه الطريقة تتضمن اختيار اصغر القيم ووضعها في قمة القائمة (اي ان القيمة الصغرى تنقل على السطح) وقبما يأتي خطوات الخوارزمية :

- 1- في المرحلة الاولى (first pass) :
 نقارن العناصر في الموقعين (n) و ($n-1$) ونبادل موقعهما ليكون الاصغر قبل الاخر ، ونستمر الى اعلى القائمة لحين الوصول الى مقارنة العنصر في الموقع الثاني مع العنصر في الموقع الاول .
- 2- في المرحلة الثانية (second pass) :
 نقارن بنفس الطريقة السابقة ولكن من العنصر في الموقع (n) الى العنصر في الموقع الثاني [لان الموقع الاول اختير فيه العنصر الاكبر قيمة في الخطوة السابقة هو -1] .
- 3- نكرر الخطوات اعلاه لـ ($n-1$) من المراحل .

البرنامج الفرعي الأول للترتيب السريع عند اختيار عنصر المقارنة في الوسط (mid)

```
void quicksortn(int data[], int first, int last)
{
    int lower=first+1, upper=last;
    swap(data[first], data[(first+last)/2]);
    int bound=data[first];

    while( lower<=upper )
    {
        while ( data[lower]<bound )
            lower++;
        while ( bound<data[upper] )
            upper--;
        if (lower<upper)
            swap( data[lower++], data[upper--] );
        else lower++;
    }

    swap( data[upper], data[first] );
    if ( first<upper-1 )
        quicksortn(data, first, upper-1);
    if ( upper+1<last )
        quicksortn(data, upper+1, last);
}
```

271

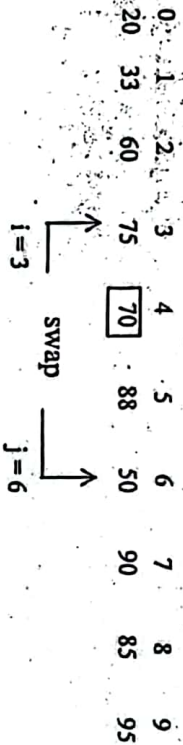
8-1-7 الترتيب السريع Quick Sort

إن خوارزمية هذا الترتيب تعتمد فكرة التجزئة والدمج (divide & conquer) التي ترتب القائمة بتم مقارنة وتبديل العناصر المتجاورة لتلك إذا كان العنصر بعيد عن موقعه الصحيح في هذه الحالة يحتاج إلى عدد كبير من المقارنات .
إن خوارزمية الترتيب السريع تعالج هذا الضعف وتسمح بأجراء المقارنات بين العناصر في المواقع البعيدة وبأقل عدد من المقارنات ، إذ تعتمد فكرة التجزئة والدمج (divide & Conquer) وتتلخص خطوات هذه الخوارزمية بالآتي :-

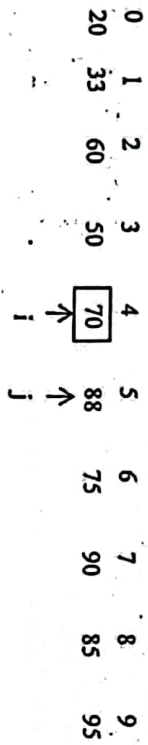
- 1- اختيار أحد عناصر القائمة (في الوسط تقريباً) وليكن (x) أي تقسم القائمة إلى جزأين.
 - 2- يبدأ المصحح من الاتجاهين، أي يبحث في النصف الأول (اليسار) من القائمة عن العنصر الذي قيمته أكبر من (x) ، ويبحث في النصف الثاني (اليمين) من القائمة عن العنصر الذي قيمته أصغر من (x) . يستقبل هذين العنصرين وذلك يجعل النصف الأول من القائمة يحتوي على عناصر أصغر من (x) والنصف الثاني يحتوي على عناصر أكبر من (x) .
 - 3- تأخذ النصف الأول من القائمة وتعالجه بنفس الإسلوب السابق (أي التجزئة والدمج) . وهكذا مع النصف الثاني أي نقسم بالتجزئة والدمج تبعاً لحين ترتب جميع عناصر القائمة الكلية .
- لاحظ أن عمل هذا الترتيب وفق خاصية الاستدعاء الذاتي (Recursion) ، إذ تتكرر الخطوات نفسها بعد كل تجزئة للبيانات .

270

$i=3, j=6, \text{List}[3] \neq x, x \neq \text{List}[6], i \leq j$

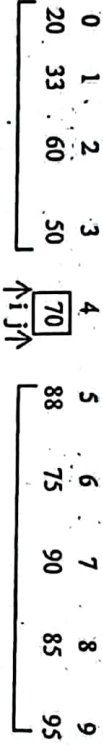


1/د



1/و

$i=5, j=5, \text{List}[5] \neq x, x \neq \text{List}[5]$



عند هذه الخطوة جرت القائمة الى قسمين القسم الايسر يحتوي على جميع الاعداد التي قيمتها اقل من (70) ، والقسم الايمن يحتوي على جميع الاعداد التي قيمتها اكبر من (70).

اما الخطوة التالية فهي تنفيذ الخوارزمية بصورة مكررة على كل جزء بنفس الطريقة اي استدعاء QuickSort(0,3) فيما يتعلق بالجزء الذي عايناه في المراتع من (0) الى (3) ، واستدعاء QuickSort(5,9) فيما يتعلق بالجزء الذي عايناه في المراتع من (5) الى (9).

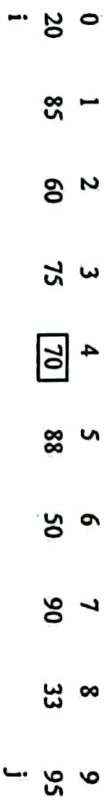
وهكذا تستمر عملية تكرار التجزئة والترتيب.

مثال: استخدم الخوارزمية quicksort للترتيب السريع (باطعلا العنصر الوسيط محورا للترتيب) لترتيب مجموعة القيم التالية ترتيبا تصاعديا.
(20,85,60,75,70,88,50,90,33,95)

الحل:

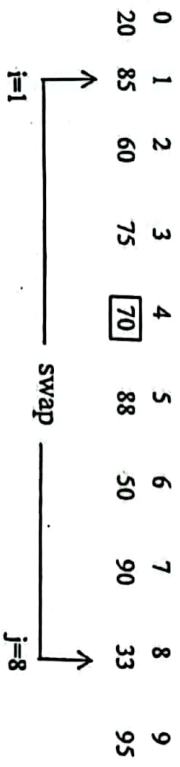
نفترض ان العناصر مخزونة في المصفوفة (List) وبالصورة الآتية:

$F = 0, L = 9, X = \text{List}[4] = 70, i = 0, j = 9$

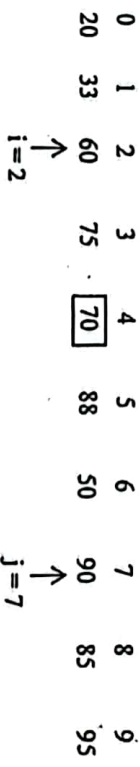


1/ب

$i = 1, j = 8, \text{List}[1] \neq x, x \neq \text{List}[8], i = j$



1/ج



البرنامج الفرعي للترتيب السريع
في هذه الخوارزمية يستخدم العنصر في الموقع الأول محورا التجزئة (pivot).

```
void quicksortf( int data[], int first, int last )
{
    while ( last > first )
    {
        int lower=first ; int upper=last ;
        int bound=data[first] ;
        while ( lower < upper )
        {
            while( data[upper] > bound )
                upper-- ;
            data[lower]=data[upper] ;
            while( (lower < upper) && (data[lower] <= bound) )
                lower++ ;
            data[upper]=data[lower] ;
        }
        data[lower]=bound ;
        quicksortf(data, first, lower-1) ;
        first=lower+1 ;
    }
}
```

طريقة أخرى:
في الخطوات السابقة لاحظ اختيار العنصر الرابع وسط القائمة ليكون محور (مركز) المقارنة ليكون (pivot) وتقارن معه العناصر الأخرى ، وهنا طريقة أخرى تتضمن اختيار العنصر في الموقع الأول لهذا الغرض وبموجب خطوات الخوارزمية الآتية :

- 1- اختيار العنصر في الموقع الأول ليكون محور (pivot) التجزئة .
- 2- نقل هذا العنصر وأخلاه موقعه.
- 3- نبدأ مسح العناصر من الجهة الأخرى أي اليمين وتقارن كل عنصر مع العنصر المحور (pivot value) .
- 4- عند إيجاد عنصر أصغر من العنصر المحور ونقل ذلك العنصر الى الموقع الذي كان فيه العنصر المحور ويبقى موقعه خالياً.
- 5- نسمح للعناصر من جهة اليسار باتجاه اليمين وتقارن هذه العناصر مع العنصر المحور فإذا وجدنا عنصراً أكبر منه ننقله الى الموقع الخالي ونترك موقعه خالياً .
- 6- ننقل الى جهة اليسار نسمح للعناصر باتجاه اليمين لحين الوصول الى عنصر أكبر من العنصر المحور وننقله الى الجهة الأخرى وننقل الأسلوب لنقل الى الجهة اليمنى.
- 7- بعد توزيع العناصر التي أكبر من العنصر المحور في اليمين والعناصر التي أصغر منه في اليسار نعيد العنصر المحور الى الموقع الخالي ليصبح هو الفاصل بينهما.
- 8- تكرر الخطوات السابقة على عناصر القائمة عدا العنصر الأول ثم تكرر مرة أخرى على عناصر القائمة عدا العنصرين الأولين وهكذا لحين انتهاء عملية الترتيب.

$i=1, j=8, \text{List}[i] \neq X$

0	1	2	3	4	5	6	7	8	9
40	85	10	75	38	90	30	70	40	95
								$\downarrow$	$\downarrow$
								$\downarrow$	$\downarrow$

move it $\rightarrow$

$i=1, j=8, \text{List}[i] = \text{List}[j]$

0	1	2	3	4	5	6	7	8	9
40	85	10	75	38	90	30	70	85	95
	$\downarrow$							$\downarrow$	
	$\downarrow$							$\downarrow$	

i j

$i=1, j=6, \text{List}[i] \neq X$

0	1	2	3	4	5	6	7	8	9
40	85	10	75	38	90	30	70	85	95
						$\downarrow$			
						$\downarrow$			

move it $\leftarrow$ i j

$i=1, j=6, \text{List}[i] = \text{List}[j]$

0	1	2	3	4	5	6	7	8	9
40	30	10	75	38	90	30	70	85	95
	$\downarrow$					$\downarrow$			
	$\downarrow$					$\downarrow$			

i j

277

مثال: استخدم الخوارزمية quicksort للترتيب السريع (باعتماد العنصر الاول محورا للترتيب) لترتيب مجموعة القيم الآتية ترتيبا تصاعديا.
(50, 85, 10, 75, 38, 90, 30, 70, 40, 95)

الحل: نفترض ان العناصر مخزنة في المصفوفة List وبالصورة الآتية:

$F=0, L=9, i=0, j=9, X = \text{List}[0] = 50$

0	1	2	3	4	5	6	7	8	9
50	85	10	75	38	90	30	70	40	95
$\downarrow$									
$\downarrow$									

i j

$i=0, j=8, \text{List}[i] \neq X$

0	1	2	3	4	5	6	7	8	9
50	85	10	75	38	90	30	70	40	95
								$\downarrow$	
								$\downarrow$	

Move it $\leftarrow$ i j

$i=0, j=8, \text{List}[i] = \text{List}[j]$

0	1	2	3	4	5	6	7	8	9
40	85	10	75	38	90	30	70	40	95
								$\downarrow$	
								$\downarrow$	

i j

276

i=4, j=54

0	1	2	3	4	5	6	7	8	9
40	30	10	38	38	90	75	70	85	95

i=4, j=4, i ≠ j, List[i] = X

0	1	2	3	4	5	6	7	8	9
40	30	10	38	50	90	75	70	85	95

عد هذه الخطوة جزئت القائمة الى قسمين، القسم الايسر يحتوي على جميع الاعداد التي قيمتها اقل من (50)، والقسم الايمن يحتوي على جميع الاعداد التي قيمتها اكبر من (50).
في الخطوة التالية تنفذ نفس الخوارزمية على الجزء الايسر اي باستدعاء (QuickSort (F_{i-1}, I-1) ثم على الجزء الايمن لحين ترتيب جميع عناصر القائمة.

ان تحليل خوارزمية الترتيب السريع يشير الى ان معدل عدد المقارنات (Average No. of comparisons) هو $(N \log_2 N)$ ومعدل عدد التبادلات (Average No. of exchanges) هو $(2 N \log_2 N)$ وان استخدام هذه الخوارزمية لترتيب عناصر قائمة مرتبة سيطلب وقت تنفيذ طويل ويتناسب مع (N^2) بدلا من المعدل $(N \log_2 N)$.

i=3, j=6, List[i] ≠ X

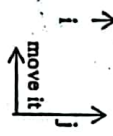
0	1	2	3	4	5	6	7	8	9
40	30	10	75	38	90	30	70	85	95

i=3, j=6, List[i] = List[j]

0	1	2	3	4	5	6	7	8	9
40	30	10	75	38	90	75	70	85	95

i=3, j=4, List[i] ≠ X

0	1	2	3	4	5	6	7	8	9
40	30	10	75	38	90	75	70	85	95



i=3, j=4, List[i] = List[j]

0	1	2	3	4	5	6	7	8	9
40	30	10	38	75	90	75	70	85	95

وصف خوارزمية الترتيب الكرمي

في ترتيب الاختيار (Selection Sort) يكون البحث عن أصغر عنصر في مجموعة القيم ونقله إلى موقعه الصحيح ، ثم يستمر البحث في المتبقي من عناصر القائمة عن أصغر عنصر فيها لوضع في موقعه الصحيح بعد موقع العنصر السابق ، ... وهكذا إلى أن يتم ترتيب القائمة . أما في هذا الترتيب (Heap Sort) فيمكن تلخيص خطواته بما يلي:

1 - تستخدم المصفوفة لحزن البيانات الأولية غير المرتبة ، ثم تعتمد فكرة الكرمي (Heap) في ترتيب العناصر كشجرة ثنائية جذرها هو أكبر العناصر ويكون في الموقع الأول للمصفوفة ، أما فرعي الجذر (الأيسر و الأيمن) فيكونا في الموقعين التاليين مباشرة .

2- نأخذ الجذر من الكرمي (heap) لأنه يمثل أكبر قيمة ونضمه في موقعه الصحيح .

3 - إعادة ترتيب (Reheap) العناصر المتبقية على شكل كرمي جديدة (heap) أي شجرة ثنائية أخرى ، وهذا يعني أن العنصر الأكبر التالي سيكون هو الجذر فيها ونضمه في الموقع التالي للعنصر السابق .

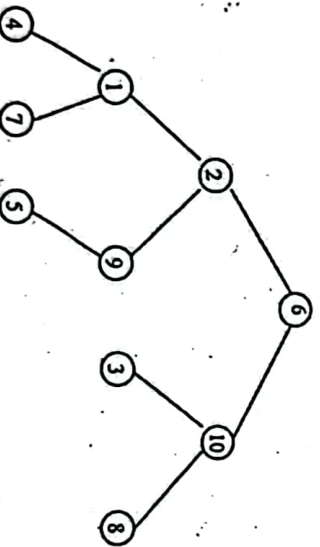
4 - تكرر الخطوات أعلاه لحين الحصول على القائمة المرتبة .

ملاحظة : من الضروري المحافظة على بناء أوراق المستوى الأخير في خفة البسار .

مثال : نأخذ المدخلات التالية : 6 ، 2 ، 10 ، 1 ، 3 ، 8 ، 4 ، 5 ، 7

الحل :

1 - يمكن تمثيل هذه المدخلات في شجرة ثنائية ونحزن عناصرها في مصفوفة

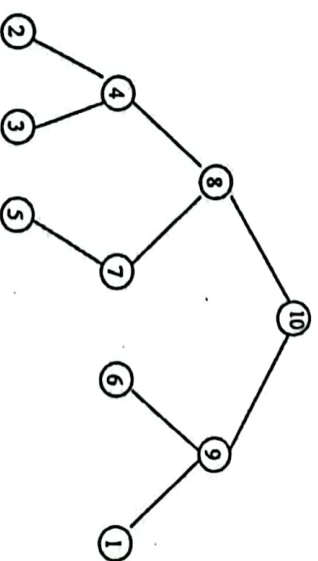


9-1-7 الترتيب الكرمي Heap Sort

يعتمد هذا الترتيب فكرة بناء الأشجار الثنائية وتمثيلها في مصفوفة كما شرحنا ذلك في الفصل السادس . ولغرض وصف أسلوب خوارزمية هذا الترتيب نحتاج إلى توضيح بعض المفاهيم .

الكرمي Heap

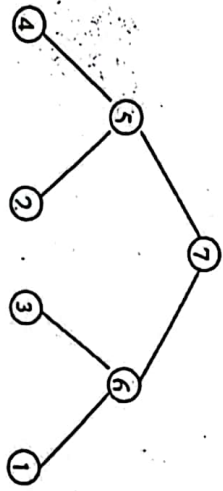
هي صيغة بيانية تتوفر فيها خاصيتين ، الأولى تتعلق بالشكل (shape) والتي يجب أن يكون شجرة ثنائية متوازنة تامة (Perfectly Balanced Binary Tree) . وتقع أوراقها للمستوى الأخير في جهة اليسار . والخاصية الثانية تتعلق بترتيب العناصر (Order) والتي يعني أن قيمة كل عقدة يجب أن تكون أكبر أو تساوي قيمة كل من عقديها الفرعيتين (طفلها) اليسرى واليمنى وترتيبها من حيث القيمة غير مهم كما في الشكل التالي :



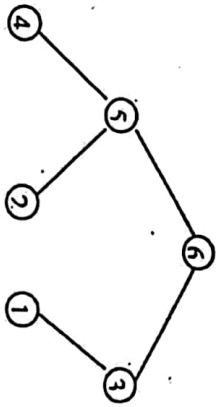
Reheap

هي عملية إعادة توزيع عقد الشجرة بحيث تتوفر خاصيتي الشكل والترتيب لعناصرها ابتداء من أنقى عقدة مقسمة في أنقى مستوى ثم صعوداً نحو الأعلى من خلال استبدال عقد الأبناء التي قيمتها أكبر من عقدة الجذر لتصبح عقدة الجذر هي الأكبر .

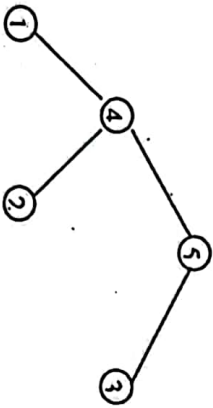
5 - توضع قيمة الجذر (8) ويعاد ترتيب العناصر المتبقية في كومة جديدة للتصحيح :



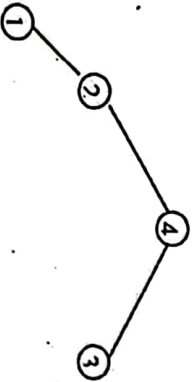
6 - تؤخذ قيمة الجذر (7) ويعاد ترتيب العناصر المتبقية في كومة جديدة للتصحيح :



7 - تؤخذ قيمة الجذر (6) ويعاد ترتيب العناصر المتبقية في كومة جديدة للتصحيح :

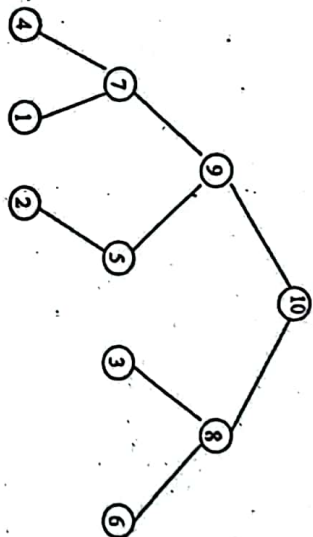


8 - تؤخذ قيمة الجذر (5) ويعاد ترتيب العناصر المتبقية في كومة جديدة للتصحيح :

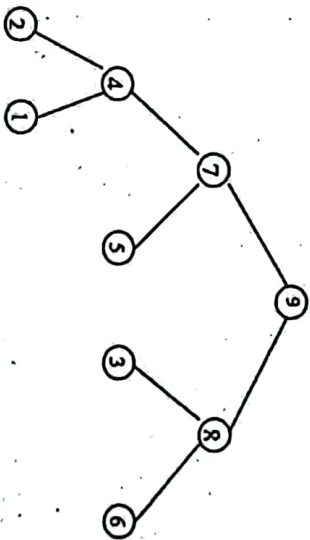


283

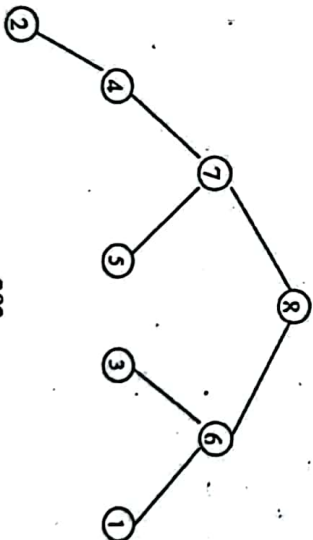
2 - عند إعادة ترتيب عناصر هذه الشجرة لتكوين الكومة (heap) تصبح :



3 - تؤخذ قيمة الجذر (10) وهي أعلى قيمة ويعاد ترتيب العناصر المتبقية في كومة جديدة للتصحيح :



4 - تؤخذ قيمة الجذر (9) وتوضع بالترتيب بعد العنصر السابق (10) ويعاد ترتيب العناصر المتبقية في كومة جديدة للتصحيح :



282

برنامج فرعي للترتيب الكوسم

```
void heap( int data[], int size)
{
    for (int i=size/2-1; i>= 0; i--)
        reheap( data, i, size-1);
    for (int i=size-1; i>=1; i--)
        swap( data[0], data[i]);
    reheap( data, 0, i-1);
}

ملاحظه: هذا البرنامج الفرعي يستدعي برنامج فرعي آخر هو
reheap في الصفحة التاليه:
```

البرنامج الفرعي (Reheap)

```
void reheap( int data[], int first, int last)
{
    int largest= 2*first+1;
    while ( largest <= last)
    {
        if ( ( largest < last) && ( data[largest] < data[largest+1]) )
            largest ++;
        if ( data[first] < data[largest] )
        {
            swap( data[first], data[largest] );
            first= largest;
            largest= 2 * first + 1;
        }
        else ( largest= last + 1 ) ;
    }
}
```

9- تؤخذ قيمة الجذر (4) ويعاد ترتيب العناصر المتبقية في كومة جديدة لتصيح :



10- تؤخذ قيمة الجذر (3) ويعاد ترتيب العناصر المتبقية في كومة جديدة لتصيح :



11- تؤخذ قيمة الجذر (2) ويعاد ترتيب العناصر المتبقية في كومة جديدة لتبقى فيها العقدة الأخيرة وقيمتها (1).

12- أن العناصر التي أخذت قايما بعد تشكيل كل كومة وخرزت في المصفوفة التي أصبحت مرتبة كالآتي :

10	9	8	7	6	5	4	3	2	1
----	---	---	---	---	---	---	---	---	---

الحل:

- 1 - نفترض ان حجم كل مجموعة سيكون (3-records) ، أي سيصبح الملف الأصلي كالآتي:
(77,80,20) , (30,90,44) , (11,70,88) , (33,66,10) , (55,22,40) , (60,99,50)
- 2 - نأخذ المجموعة الأولى ونرتب عناصرها ونضعها في الملف (file1) ، ونأخذ المجموعة الثانية ونرتب عناصرها ونضعها في الملف (file2) . نستمر بهذا التتابع (نرتب عناصر المجموعة ونقلها الى احد الملفين) فنضع المجموعة الثالثة في الملف (file3) والمجموعة الرابعة في الملف (file4) ، ثم الخامسة في الملف (file1) و السادسة في الملف (file2) .
file1: (20,77,80) , (11,70,88) , (22,40,55)
file2: (30,44,90) , (10,33,66) , (50,60,99)
- 3 - نأخذ المجموعة الأولى من الملف (file1) و ندمجها مع المجموعة الأولى من الملف (file2) لتكوين مجموعة أكبر (ستكون سعتها 6-records) ونرتب عناصرها ونضعها في الملف (file3) . ونستمر بنفس الطريقة فنأخذ المجموعة الثانية في كلا الملفين file1 و file2 فندمجها معا" لتكوين مجموعة أكبر (سعتها 6-records) ونرتب عناصرهما ونضعها في الملف (file4) . ونكرر ذلك أيضا" مع المجموعة الثالثة في كلا الملفين file1 و file2 ونضع الناتج في الملف file3 ، وهما ستكون قد أخذنا جميع مجموعات الملفين file1 و file2 (أي أصبحا خاليين) وتم توزيعها في الملفين file3 و file4 .
file3: (20,30,44,77,80,90) , (22,40,55,60,99)
file4: (10,11,33,66,70,88)
- 4 - نأخذ المجموعة الأولى من الملف (file3) و ندمجها مع المجموعة الأولى من الملف (file4) لتكوين مجموعة أكبر (ستكون سعتها 12-records) ونرتب عناصرها ونضعها في الملف (file1) . ثم نأخذ المجموعة الثانية من الملف (file3) التي لم يدمجها مقابلها مجموعة في الملف (file4) لانه أصبح خاليا" ، فنأخذ تلك المجموعة ونضعها في المجموعة المتبقية (مهما كان عدد عناصرها البيانية) ونضعها في الملف (file2) .
file1: (10,11,20,30,33,44,66,70,77,80,88,99)
file2: (22,40,50,55,60,99)

7- 10 ترتيب النسخ الثنائي المتوازن (Balanced Two-Way Merge Sort)

إن خوارزمية هذا الترتيب الخارجي هي تطوير أو تحديث لخوارزمية ترتيب النسخ الثنائي التي تقدم نشرها وذلك من خلال زيادة عدد الملفات المستخدمة ، إذ في هذه الخوارزمية يستخدم ملفين لتوزيع البيانات المحملة (input files) وهما (file1) و (file2) ، ثم نخزن مجموعات البيانات بعد ترتيبها (sorted runs) في ملفين آخرين بالتتابع هما (file3) و (file4) .

أي سيتم هنا الاستثناء عن خطوة إعادة توزيع المجموعات من الملف (file3) الى الملفين (file1) و (file2) قبل دمجها و توزيعها في كل مرة.

أي سيكون تاملل خطوات هذه الخوارزمية كالآتي:

- 1 - نأخذ مجموعات البيانات من الملف الأصلي ونوزعها في ملفين (file1) و (file2) .
- 2 - نأخذ المجموعة الأولى من الملف (file1) والمجموعة الأولى من الملف (file2) فيتم دمجها و ترتيبهما وفق المطلوب (تصادبيا" أو تنازليا") لتكوين مجموعة واحدة نخزن في الملف (file3) .

3 - نأخذ المجموعة الثانية من الملف (file1) والمجموعة الثانية من الملف (file2) فيتم دمجها و ترتيبهما وفق المطلوب (تصادبيا" أو تنازليا") لتكوين مجموعة واحدة نخزن في الملف (file4) .

- 4 - نكرر الخطوات (2) و (3) بالتتابع لحين خلو الملفين (file1) و (file2) ، فنأخذ المجموعة الأولى من الملف (file3) والمجموعة الأولى من الملف (file4) ، فنأخذ و ترتيبهما وفق المطلوب لتكوين مجموعة واحدة نخزن في الملف (file1) . ثم نأخذ المجموعة الثانية من الملف (file3) والمجموعة الثانية من الملف (file4) فيتم دمجها و ترتيبهما وفق المطلوب لتكوين مجموعة واحدة نخزن في الملف (file2) .

6 - بعد تكرار هذه العمليات سفصل في النهاية على مجموعة واحدة كبيرة تحوي جميع العناصر البيانية وهي مرتبة وفق المطلوب في ملف واحد.

مثال: لنفترض ان الملف الخارجي يحتوي القيم البيانية التالية:

77, 80, 20, 30, 90, 44, 11, 70, 88, 33, 66,
10, 55, 22, 40, 60, 99, 50

المطلوب هو ترتيب هذه البيانات تصادبيا" باستخدام خوارزمية ترتيب النسخ الثنائي المزوج.

خوارزمية البحث التسلسلي

- 1- Read in the (key)
- 2- Initialize the (Marker) to the first array index (0)
- 3- While (key != Item at location Marker) & (not at the end of the list) Increment the Marker ++
- 4- If (key == Item) at location Marker Then Return Marker Else Return (-1)

أيعازات برمجية لبرنامج فرعي (function) للبحث التسلسلي

```
void SeqSearch(int data[], int key)
{
    int pos=-1;
    for(int i=0, i<=size-1, i++)
        if(key==data[i])
            pos=i;
    if(pos==-1)
        then cout<<"\n The key is not exist in the data list ";
    else
        { cout<<"\n The key is found at the position : ";
          cout<<pos ; }
}
```

5 - الآن نأخذ المجموعة الأولى من الملف (file1) (عدد عناصرها أصبح 12 records) ونمحيها مع المجموعة المناظرة لها في الملف (file2) (عدد عناصرها المتبقية 6-records) لتكوين مجموعة أكبر (عدد عناصرها 18-records) ونرتب عناصر هذه المجموعة الكبيرة ونضعها في الملف (file3). هنا أصبح كل من الملف (file1) و الملف (file2) خاليين ، وحصلنا على الملف (file3) الذي يحتوي جميع عناصر الملف الأصلي وهي مرتبة تصاعدياً حسب المطلوب ، أي تم ترتيب عناصر الملف الأصلي ليصبح :-
file3: 10,11,20,22,30,33,40,44,50,55,60,66,70,77,80,88,90,99.

2-2 البحث Searching

هي عملية إيجاد عنصر معين في مجموعة من البيانات إذا كان ذلك العنصر موجوداً.

مثلاً إيجاد اسم شخص في دليل الهاتف.

إن عملية البحث قد تكون إيجابية عند وجود العنصر المطلوب و قد تكون سلبية في حالة كون العنصر غير موجود في قائمة البحث. وتكون عملية البحث فعالة عندما تكون قائمة البحث مرتبة (sorted) وفق نسق (قيمة حقل) معين .

1-2-7 أنواع خوارزميات البحث

- 1- البحث التسلسلي Sequential Search
- 2- البحث الثنائي Binary Search
- 3- البحث الكلي Block Search
- 4- بحث الشجرة الثنائية Binary Search Tree

2-2-7 البحث التسلسلي Sequential Search

وهي عملية البحث عن عنصر معين في قائمة من العناصر من خلال مسح (استعراض) جميع عناصر القائمة من بدايتها وانتهاءً وبالتسلسل لحين الوصول الى العنصر المطلوب في حالة وجوده أو الوصول الى نهاية القائمة عندما يكون غير موجود لذا فإن معدل عدد المقارنات سيكون $O(n^2)$ ، أي إن وقت تنفيذ هذه الخوارزمية سيكون $O(n^2)$.

4- If item at location (top) is the target (key)
Then Return location (top)
[indicating that the target has been found]
Else Return (-1)
[indicating that the target has not been found]

البرنامج الفرعي للبحث الثنائي

```
int binary2( int data[], int key, int lower, int upper)
{
    pos=(-1); int mid;
    if (lower <= upper)
    {
        mid=( (lower+upper)/2 );
        if ( key == data[mid] )
        {
            pos=mid; return pos;
        }
        else
        {
            if (key < data[mid])
                binary2(data, key, lower, mid-1);
            else
                binary2(data, key, mid+1, upper);
        }
    }
    return pos;
}
```

3-2-7 Binary Search

ان خوارزمية هذا البحث تفترض التقسيم عن عنصر معين بشرط ان تكون القائمة مرتبة (sorted) حسب تامل (ترتيب) معين باستخدام إحدى طرق الترتيب ويمكن تلخيصها بالخطوات الآتية :

- 1- تحديد موقع العنصر الذي يقع في منتصف القائمة تقريبا .
 - 2- مقارنة العنصر المراد البحث عنه وليكن (X) مع العنصر الذي يقع في منتصف القائمة .
 - 3- اذا كان العنصر المطلوب (X) مساويا للعنصر في الوسط مستنتجى عملية البحث هنا .
 - 4- اذا كان العنصر المطلوب (X) اقل من قيمة العنصر الذي يقع في المنتصف فان البحث سيتجهض في الجزء الذي يضم القيم الاصغر وليكن الجزء الذي في القسم الايسر .
 - 5- اذا كان العنصر المطلوب (X) اكبر من قيمة العنصر الذي يقع في المنتصف فان البحث سيتجهض في الجزء الذي يضم القيم الاكبر وليكن الجزء الذي يقع في القسم الايمن .
 - 6- في أي من الحالتين (4 و 5) تتم معالجة ذلك الجزء بنفس الطريقة ، أي اختيار نقطة المنتصف والمقارنة لعين الوصول الى العنصر المطلوب .
- في هذه الخوارزمية فان كل مقارنة مستؤدي الى تقليص عدد المقارنات اللاحقة الى النصف ولهذا فان اكبر عدد المقارنات سيبلغ تقريبا (log₂n) عند البحث في قائمة عدد عناصرها (n) مع ملاحظة انه يجب ان تكون العناصر مخزونة في مصفوفة لادها ستكون في مواقع متعاقبة .

خوارزمية البحث الثنائي

- 1- Set top=size-1 , Set bottom=0
- 2- While (top > bottom)
Do :
Set middle=(top + bottom) / 2 [using integer division]
If key > Item at index(middle)
Then set bottom=middle + 1
Else set top = middle
- 3- If top = (-1) , Return (-1)
[indicating that the list is empty]

d) iteration 3 : First <= Last and Not Found

$$\text{mid} = (4+4) \div 2 = 4$$

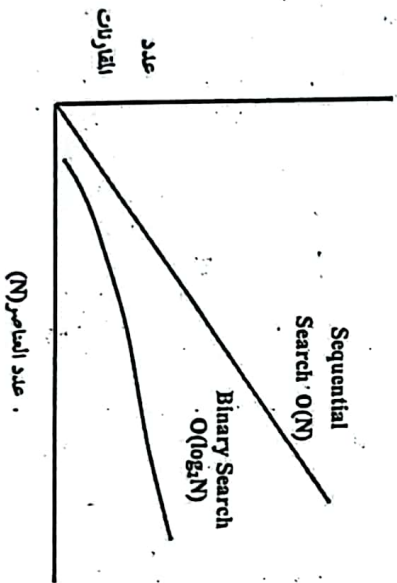
List [4] = 25, So Found is true

9	11	16	18	25	29	32	35
---	----	----	----	----	----	----	----

0 1 2 3 4 5 6 7
mid

Found is true, so location ← mid

ان البحث الثاني يكون فعالا وسريعا في التوائم التي يكون عدد عناصرها (n) كبيرا
اما عندما يكون عدد العناصر قليلا فان البحث التسلسلي يصبح هو الاسرع والشكل
التالي يوضح مقارنة بين عدد المقارنات وعدد العناصر لكل من البحث التسلسلي
والبحث الثاني والعلاقة بينهما .



الشكل (7-1) : مقارنة بين البحث التسلسلي والبحث الثاني

مثال: في هذا المثال نوضح تنفيذ الـ BinSearch لإيجاد القيمة المقابلة (25) في قائمة عدد عناصرها n=8.

a) initialize : Found = false, Key = 25

9	11	16	18	25	29	32	35
---	----	----	----	----	----	----	----

0 1 2 3 4 5 6 7
First Last

b) iteration 1 : First <= Last and Not Found

$$\text{mid} = (0+7) \div 2 = 3$$

List [3] < 25 so move First to (mid + 1)

9	11	16	18	25	29	32	35
---	----	----	----	----	----	----	----

0 1 2 3 4 5 6 7

mid First Last

c) iteration 2 : First <= Last and Not Found

$$\text{Mid} = (4+7) \div 2 = 5$$

List [5] > 25, So move Last to (mid - 1)

9	11	16	18	25	29	32	35
---	----	----	----	----	----	----	----

0 1 2 3 4 5 6 7

First Last mid

برنامج رقم 29: تمثيل خوارزمية ترتيب الاختيار (Selection Sort)

```
#include <iostream>
using namespace std;
const int size=10;
int table[size];
class selection
{
private:
    int item;
public:
    int table[size];
    void selectionsort(int data[],int n)
    {
        for(int i=0; i<n-1; i++)
        {
            for(int j=i+1; j<n; j++)
                if(data[i]>data[j])
                {
                    pos=i;
                    int item=data[j];
                    data[pos]=data[i];
                    data[i]=item;
                }
        }
    }
};

int main()
{
    int i,k;
    selection ss;
    cout<<"\n How many values you like to sort? "; cin>>k;
    for(int j=0; j<k; j++)
    {
        cout<<"\n Input the item value : "; cin>>table[j];
    }
    cout<<"\n the original list before sorting is : ";
    for(int j=0; j<k; j++)
        cout<<table[j]<<" ";
    cout<<"\n ";
    ss.selectionsort(table,k);
    cout<<"\n the sorted list elements after applying the selection sort algorithm : ";
    for(int j=0; j<k; j++)
        cout<<table[j]<<" ";
    cout<<table[j]<<" ";
    return 0;
}
```

295

برنامج الفصل

برنامج رقم 28: تمثيل خوارزمية الترتيب بالاضافة (Insertion Sort)

```
#include <iostream>
using namespace std;
const int size=10;
int table[size];
class insertion
{
private:
    int item;
public:
    int table[size];
    void insertionSort(int data[],int n)
    {
        for(int i=1; i<n; i++)
        {
            int item=data[i];
            for(int j=i; j>0 && item<data[j-1]; j--)
                data[j]=data[j-1];
            data[j]=item;
        }
    }
};

int main()
{
    int i,k;
    insertion ins;
    cout<<"\n How many values you like to sort? "; cin>>k;
    for(int j=0; j<k; j++)
    {
        cout<<"\n Input the item value : "; cin>>table[j];
    }
    cout<<"\n the original list before sorting is : ";
    for(int j=0; j<k; j++)
        cout<<table[j]<<" ";
    cout<<"\n ";
    ins.insertionsort(table,k);
    cout<<"\n the sorted list elements after applying the insertion sort algorithm : ";
    for(int j=0; j<k; j++)
        cout<<table[j]<<" ";
    cout<<table[j]<<" ";
    return 0;
}
```

294

برنامج رقم - 31 : تمثيل الخوارزمية الأولى للترتيب السريع (Quick Sort) باستخدام صيغة الاستدعاء الذاتي . (Recursion)

```
#include <iostream>
using namespace std;
const int size=11;
int table[size];
class quicksort
{
private:
    int item;
public:
    void quicksort(int data[],int first,int last)
    {
        while (last>first)
        {
            int lower=first; int upper=last;
            int bound=data[first];
            while ( lower<upper )
            {
                while( data[upper]>bound )
                    upper--;
                data[lower]=data[upper];
                while( (lower<upper) && (data[lower]<=bound) )
                    lower++;
                data[upper]=data[lower];
            }
            data[lower]=bound;
            quicksort(data,first,lower-1);
            first=lower+1;
        }
    }
};

int main()
{
    int k;
    quicksort qs;
    cout<<"\n How many values you like to sort ? ";cin>>k;
    for(int j=0;j<k;j++)
    {
        cout<<"\n Input the item value ";cin>>table[j];
    }
    cout<<"\n the original list before sorting is : ";
    for(int j=0;j<k;j++)
        cout<<table[j]<<" ";
    cout<<"\n ";
    qs.quicksort(table,0,k-1);
    cout<<"\n the sorted list elements after applying the quick sort algorithm ";
    for(int j=0;j<k;j++)
        cout<<table[j]<<" ";
    return 0;
}
```

297

برنامج رقم - 30 : تمثيل خوارزمية ترتيب الفقاعة (Bubble Sort)

```
#include <iostream>
using namespace std;
const int size=10;
int table[size];
class bubble
{
private:
    int item;
public:
    int table[size];
    void bubblesort(int data[],int n)
    {
        for(int i=0; i<n-1; i++)
        {
            for( int j=n-1; j>i; j--)
            {
                if(data[j]<data[j-1])
                {
                    int item=data[j];
                    data[j]=data[j-1];
                    data[j-1]=item;
                }
            }
        }
    }
};

int main()
{
    int k;
    bubble bs;
    cout<<"\n How many values you like to sort ? ";cin>>k;
    for(int j=0;j<k;j++)
    {
        cout<<"\n Input the item value ";cin>>table[j];
    }
    cout<<"\n the original list before sorting is : ";
    for(int j=0;j<k;j++)
        cout<<table[j]<<" ";
    cout<<"\n ";
    bs.bubblesort(table,k);
    cout<<"\n the sorted list elements after applying the Bubble sort algorithm ";
    for(int j=0;j<k;j++)
        cout<<table[j]<<" ";
    return 0;
}
```

296

برنامج رقم - 33 : تمثيل خوارزمية الترتيب الكسي (Heap Sort)

```
#include <iostream>
using namespace std;
const int size=11;
int table[size];
class heapSort
{
private:
    int item;
public:
    void reheap(int data[],int first, int last)
    {
        int largest=2*first+1;
        while( largest<last)
        {
            if((largest<last) && (data[largest]<data[largest+1]))
                largest++;
            if ( data[first]<data[largest] )
                swap( data[first],data[largest]);
            first=largest;
            largest=2*first+1;
        }
        else (largest=last+1);
    }
}

void heap( int data[], int size )
{
    for( int i=size/2-1; i>=0; i--)
        reheap(data,i,size-1);
    for( int i=size-1; i>=1; i--)
    {
        swap( data[0],data[i]);
        reheap(data,0,i-1);
    }
}

void display()
{
    int n;
    cout<<" the content of the table is : ";
    for(int i=0; i<5; i++)
        cout<<table[i]<<" ";
}

int main()
{
    int k;
    heapSort hs;
    cout<<"\n How many values you like to sort ? ";cin>>k;
    for(int j=0;j<k;j++)
    {
        cout<<"\n Input the item value ";cin>>table[j];
        cout<<"\n The original list before sorting is : ";
        for(int j=0; j<k; j++)
```

299

برنامج رقم - 32 : تمثيل الخوارزمية الترتيب الثنائي للترتيب السريع (Quick Sort2) باستخدام طريقة الاستدعاء الذاتي (Recursion).

```
#include <iostream>
using namespace std;
const int size=11;
int table[size];
class quicksort
{
private:
    int item;
public:
    void quicksortm(int data[],int first, int last)
    {
        int lower=first+1,upper=last;
        swap(data[first],data[(first+last)/2]);
        int bound=data[first];
        while( lower<upper)
        {
            while (data[lower]<bound )
                lower++;
            while (bound<data[upper])
                upper--;
            if (lower<upper)
                swap( data[lower++],data[upper--]);
        }
        swap( data[upper],data[first]);
        if ( first<upper-1)
            quicksortm(data,first,upper-1);
        if ( upper+1<last )
            quicksortm(data,upper+1,last);
    }

    int main()
    {
        int k;
        quicksort qs;
        cout<<"\n How many values you like to sort ? ";cin>>k;
        for(int j=0;j<k;j++)
        {
            cout<<"\n Input the item value ";cin>>table[j];
            cout<<"\n the original list before sorting is : ";
            for(int j=0; j<k; j++)
                cout<<table[j]<<" ";
        }
        cout<<"\n ";
        qs.quicksortm(table,0,k-1);
        cout<<"\n the sorted list elements after applying the quick sort algorithm ";
        for(int j=0; j<k; j++)
            cout<<table[j]<<" ";
        return 0;
    }
}
```

298


```

        int mid = floor((p + r) / 2);
        merge_sort(input, p, mid);
        merge_sort(input, mid + 1, r);
        merge(input, p, r);
    }

    int main()
    {
        int input[INPUT_SIZE];
        cout << "\n Enter any 10 integer values you like to sort ";
        for (int i = 0; i < INPUT_SIZE; i++)
        {
            cout << "\n Enter the element : "; cin >> input[i];
        }
        print(input);
        merge_sort(input, 0, 9);
        cout << "\n The Sorted list is : ";
        print(input);
        return 0;
    }
}

```

برنامج رقم - 35 : تمثيل خوارزمية البحث التتبعي (Sequential Search)

```

#include <iostream>
using namespace std;
const int size=9;
int table[size];
int pos;
class seqsearch
{
private:
    int item;
public:
    void search(int data[],int key)
    {
        int pos=-1;
        for (int i=0; i<size-1; i++)
        {
            if (key==data[i])
            {
                pos=i;
            }
        }
        if (pos==-1)
        {
            cout << "\n The KEY is Not exist in the data list ";
        }
        else
        {
            cout << "\n The KEY is found at the position " << pos+1 << "\n";
        }
    }
    void display()
    {
        int i;
        cout << "\n The content of the table is : ";
        for (int i=0; i<size; i++)
        {
            cout << table[i] << " ";
        }
    }
};

int main()
{
    seqsearch sqs;
    int k, key;
    sqs.search sqs;
}

```

301

```

        cout << table[i] << " ";
        cout << "\n ";
        heapstable[i];
        cout << "\n The sorted list after applying Heap Sort algorithm : ";
        for (int i=0; i<size; i++)
        {
            cout << table[i] << " ";
        }
        return 0;
    }
}

```

برنامج رقم - 34 : تمثيل خوارزمية ترتيب الدمج (Merge Sort)

```

#include <iostream>
#include <cmath>
using namespace std;
const int INPUT_SIZE = 10;
// A simple print function
void print(int *input)
{
    for (int i = 0; i < INPUT_SIZE; i++)
    {
        cout << input[i] << " ";
    }
    cout << endl;
}

void merge(int *input, int p, int r)
{
    int mid = floor((p + r) / 2);
    int l2 = p;
    int r2 = r;
    int l3 = mid + 1;
    // Temp array
    int temp[p-r+1];
    // Merge in sorted form the 2 arrays
    while (l2 <= mid && l3 <= r)
    {
        if (input[l2] < input[l3])
        {
            temp[l2-l+1] = input[l2++];
        }
        else
        {
            temp[l2-l+1] = input[l3++];
        }
    }
    // Merge the remaining elements in left array
    while (l2 <= mid)
    {
        temp[l2-l+1] = input[l2++];
    }
    // Merge the remaining elements in right array
    while (l3 <= r)
    {
        temp[l2-l+1] = input[l3++];
    }
    // Move from temp array to master array
    for (int i = p; i <= r; i++)
    {
        input[i] = temp[i-p];
    }
}

void merge_sort(int *input, int p, int r)
{
    if (p < r)
    {
        // Merge Sort
    }
}

```

300

أسئلة الفصل

1. What are the main stages (steps) of any sort process .
2. What are the purposes of the sort .
3. What are the main factors which effect the choice of the sorting process.
4. Perform selection sort algorithm in ascending order on a list of (8) data elements.
5. Name one type of the external sort you know and perform its algorithm on a list of (19) integer values to be sorted in descending order.
6. Write an algorithm for Quick sort , perform this algorithm on the key elements :
90 20 50 80 40 95 70 25 15 17
and show the algorithm iteration.
7. State four types of search algorithms.
8. Write an algorithm for sequential Search .
9. Write an algorithm for binary search.
10. Consider the following keys
100 120 150 170 200 250 290 300 310 340
and perform the binary search algorithm to find the key = 250.

```
binaryrec: brec;  
cout<<"\n Create the list of data values which will be searched ";  
cout<<"\n How many values you like to Input ? "; cin>>k;  
for(int j=0;j<k;j++)  
{  
    cout<<"\n Input the item value "; cin>>table[j];  
}  
cout<<"\n The created list is: ";  
for(int i=0; i<k; i++)  
    cout<<table[i]<<" ";  
cout<<"\n ";  
cout<<"\n Enter the required KEY : "; cin>> key1;  
brec.binary2(table, key1, 0, k-1);  
if( pos==1 )  
    cout<<"\n The required key is Not exist in the data list ";  
else  
{ cout<<"\n The key is found in this list at position : ";cout<<pos+1;}  
return 0;  
};
```


الفصل الثامن تحليل الخوارزمية وإعداد البرنامج

Analysis of Algorithm and program preparation

Analysis of Algorithm	تحليل الخوارزمية	1-8
Good Quality Program	مزايا البرنامج الجيد	2-8
Program Design	تصميم البرنامج	3-8
Program Resources	موارد البرنامج	4-8

1-8 تحليل الخوارزمية

من العوامل المهمة في قياس كفاءة أو فعالية أي خوارزمية هو مقدار وقت التنفيذ (execution time) ، أي الوقت الذي يستغرقه تنفيذ الخوارزمية ولا توجد قواعد بسيطة لتحديد الوقت لذا تلجأ إلى التقدير المسبق (prior estimate) لوقت التنفيذ باستخدام بعض الأساليب الرياضية بعد معرفة عدد من العوامل المهمة المتعلقة بالمسألة التي تعالجها الخوارزمية.

إن هذا الأسلوب في التحليل الرياضي النظري يُعرف بما يسمى Complexity ، أي تضمين تحليل تفصيلي لعدد خطوات الخوارزمية في معالجة كميات (size) مختلفة من البيانات المقرر تنفيذ الخوارزمية عليها ، وهذا يساعد على :

- + فهم المحددات الأساسية المتوقعة للخوارزمية عند التصميم .
- + تقدير أسوء (أطول) وقت (worst case running time) لتنفيذ الخوارزمية لأفضل حل للمسألة من بين عدة خوارزميات لحل نفس المسألة قبل اعتمادها للتطبيق .

+ التقدير المسبق لمدى كفاءة الخوارزمية المقرر اختيارها من خلال الأحساب التقريبي لمقدار الجهد المطلوب لتطبيق الخوارزمية ، وهذا يعتمد كلفة تنفيذ الخوارزمية من خلال عاملين هما وقت التنفيذ (execution time) والمساحة التخزينية (storage space) ، وأهمهما هو العامل الأول لأن العامل الثاني أصبح أقل أهمية في ظل التطور التكنولوجي في صناعة الأجهزة (hardware) وخصوصا وسائط الذاكرة (storage media) .

تُعرف الدالة التي تحدد الوقت المتوقع للتنفيذ اعتمادا على بعض المتغيرات ذات العلاقة بخطوات الخوارزمية، لتفترض أن الخوارزمية تتضمن العبارة التالية:

$$X_i = X + 1$$

هنا يجب حساب مقدار الوقت المطلوب لتنفيذ هذه العبارة لوحدها ومن ثم يجب أن نعرف عدد مرات تكرار تنفيذها وهو ما يسمى Frequency Count (مقدار التكرار) وهو يختلف حسب صيغة البيانات .

ومن حاصل ضرب الكميتين (وقت تنفيذ العبارة و مقدار التكرار) منفصل على مجموع وقت التنفيذ المتوقع (Total Execution Time).
1- نوع أجهزة الحاسوب التي تنفذ فيها الخوارزمية.
2- لغة البرمجة المستخدمة في الحاسوب.

أما المصطلح (order of magnitude of algorithm) فيقصد به مجموع تكرارات جميع العبارات التنفيذية ويوجهه سيحدد التقدير المسبق لوقت تنفيذ الخوارزمية .
والمثال اعلاه يظهر ان الخوارزمية (أ) هي اسرع تنفيذًا من الخوارزمية (ب) ويبررها اسرع من (ج).

مثال:
لدينا المصفوفة (A) ابعادها (N*N) والمطلوب حساب مجموع كل صف (Row) وخزونه في مصفوفة اخرى لتكون (Sum) ومن ثم حساب المجموع الكلي لعناصر المصفوفة (A).
الحل: لنحول الحل بطريقتين هما :

الطريقة الاولى

```
Grandtotal = 0;
for(i=0; i < n; i++)
{
    Sum[i] = 0;
    for( j = 0; j < n; j++)
    {
        Sum[i] = Sum[i] + A[i][j]
        Grandtotal = Grandtotal + A[i][j]
    }
}
```

هذه الدورة
تكرر (N)
من المرات

2N additions

يصبح مجموع عمليات الجمع $N * (2N) = 2N^2$

3- وقت تنفيذ كل ايصال.
4- نوع المترجم او المفسر المستخدم (Compiler/ Interpreter used).

بالامكان معرفة هذه المعلومات باختبار الة (حاسوب) حقيقية او تعريف حاسوب افتراضي ، وفي كلا الحالتين فان الوقت المحسوب قد لا يكون دقيقا وبمالمنا لعدد من الحواسيب او لاي حاسوب ، اذا ان مترجم اللغة قد يختلف من حاسوب الى اخر بالإضافة الى عوامل اخرى . وهذه الاعتبارات نجعلنا نركز هدفنا في التقدير المسبق لوقت التنفيذ على عدد مرات التكرار (frequency count) للايجازات (العبارات البرمجية).

لتأخذ الامثلة الثلاثة التالية (ا ، ب ، ج) :

for(i=1, i<=n, i++)	_____	for(i=1, i<=n, i++)	_____
_____	_____	_____	_____
For(j=1, j<=n, j++)	_____	_____	_____
_____	_____	x = x + 1	_____
_____	_____	_____	_____
x = x + 1	_____	_____	_____
(ج)	(ب)	(ا)	

في المثال (أ):

ان عبارة الجمع (x = x + 1) لم ترد داخل اية صيغة تكرارية ، اي ان عدد مرات تنفيذها (frequency Count = 1)

في المثال (ب):

عبارة الجمع تتكرر (n) من المرات.

في المثال (ج):

عبارة الجمع تتكرر (n²) من المرات.

نظر القوسنا (n=10) فان هذه التكرارات تكون (100,10,1) وهذه تناظر ركوب دراجة موائية ، ركوب سيارة ، ركوب الطائرة بالمقارنة مع المسافة التي مستطعها كل مركبة في وحدة الزمن (مساحة مثال) وهذا يستخدم التعبير (order of magnitude of a statement) يقصد به عدد مرات تكرار تنفيذ العبارة.

مثال :

لنأخذ برنامجاً حساب (استخراج) عدد (fibonacci) الذي يقع في الترتيب (n) علماً ان سلسلة اعداد (fibonacci) هي 0, 1, 1, 2, 3, 5, 8, 13, 21, ... أي هي الاعداد التي أي عدد فيها يساوي مجموع العددين السابقين له وصيغتها العامة هي: $(F_n = F_{n-1} + F_{n-2})$ مع ملاحظة ان العدد الأول (0) في الموضع (n=0).

```

1- void file( int n)
2- { int fnum1, fnum2, fn, i ;
3-   cout<<"\n Enter the number: ";
4-   cin>>n;
5-   if ( n <= 1 )
6-     cout<<n;
7-   else
8-   {
9-     fnum2 = 0 ; fnum1 = 1 ;
10-    for ( i = 2 ; i < n+1 ; i++ )
11-    {
12-      fn = fnum1 + fnum2 ;
13-      fnum2 = fnum1 ;
14-      fnum1 = fn ;
15-    }
16-    cout<<fn ;
17-  }
18- }

```

الشكل (1 - 8) برنامج استخراج اعداد (fibonacci).

الطريقة الثانية

```

Grandtotal = 0
for( i = 0 ; i < n ; i++ )
{
  Sum [i] = 0
  for( j = 0 ; j < n ; j++ )
  {
    Sum [i] = Sum [i] + A [i][j]
  }
  Grandtotal = Grandtotal + sum [i]
}

```

يصبح مجموع عمليات الجمع $N^2 + N$

العمليات N^2 additions

العمليات N additions

نلاحظ هنا ان عدد عمليات الخوارزمية الاولى $(2N^2)$ اكبر من عدد عمليات الخوارزمية الثانية $(N^2 + 1)$ ولهذا فان الاولى تستغرق وقتاً أطول من الثانية لاختلاف خطوات الحل .

بعد اكمال الترابيت (3, 5) يصبح وقت تنفيذ هذا البرنامج $O(n)$ وهذه الصيغة

تعني (the order of magnitude) بالنسبة لـ (n) ان هذه الصيغة في التحليل وتغير وقت تنفيذ الخوارزمية تسمى

(Big-O - Notation)

تتطلب صياغة دالة رياضية (f) بدلالة الوقت (t) ونختار من هذه الدالة المتغير الأكثر قيمة (dominant term).

فمثلا الصيغة $f(n) = O(g(n))$ نقرا

[f of n equals big-oh of g of n]

ويمكن تعريفها رياضيا كالآتي :-

Definition

$F(n) = O(g(n))$ iff there exist two constants C and n_0 such that :

for all $n > n_0$

حيث ان $f(n)$ تمثل وقت احتساب (computing time) الخوارزمية بعد اعتماد

المتغير المبسط (dominant term)

فندما نقول ان وقت احتساب (Computing time) الخوارزمية هو $O(g(n))$ فهذا

يعني ان وقت التنفيذ لن يستغرق اكثر من $C * g(n)$ حيث ان :

(C) كمية ثابتة

(n) يمثل عدد العمليات المطلوبة بموجب الخوارزمية لمختلفات حجمها

اما وقت تنفيذ البرنامج فهو $O(n)$ في حالات عديدة منها :

ونستخدم المصطلح $O(n)$ في حالات عديدة منها :

$O(1)$ يعني ان وقت الاحتساب (computing time) ثابت ، أي ان عدد

الخطوات ثابت ، وقد يكون اي عدد ، ولا يعني (1) دائما

$O(n)$ يعني ان وقت الاحتساب ذو صيغة خطية (linear) مثل طبع جميع

عناصر مصفوفة حجمها (n) ، او ايجاد عنصر في قائمة موصولة .

أي ان وقت الاحتساب يتناسب مع قيمة (n) .

$O(n^2)$ يعني ان الوقت (Quadratic) مثل وقت ترتيب عناصر قائمة باستخدام

خوارزمية ترتيب الفقاعة .

$O(n^3)$ يعني ان الوقت مكعب (cubic) ، وهو الوقت اللازم لجعل جميع

عناصر المصفوفة $(N * N * N)$ مساويا للصفر .

تحليل (Time complexity) لهذا البرنامج نأخذ الاحتمالين الآتيين وهما

$n = 1$ او $n = 0$ -1

$n > 1$ -2

مع حساب مجموع وقت تنفيذ الايعازات الواردة في البرنامج نستخدم هذا المقياس يكونه الـ (Time Complexity) لهذا البرنامج .

الحالة الاولى، $n = 0$ او $n = 1$

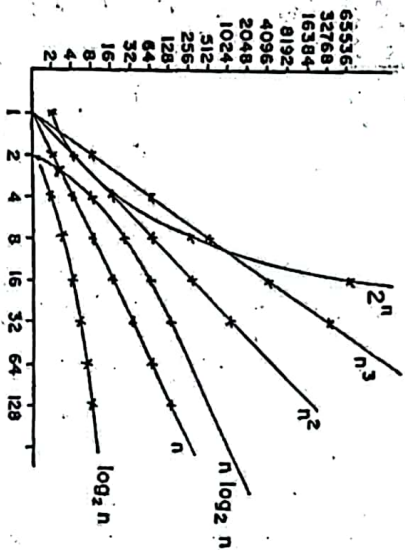
ان ايعازات السطور 2, 3, 4, 5, 6, 18 مستندة لمرة واحدة أي (count = 1) لذا لكل منها وهما ان السطرين (5, 6) يقسمان ايعاز (if) ومكمله بمعنى (then) فان (count = 2) ، وهذا يصبح المجموع (count = 4)

الحالة الثانية $n > 1$

في هذه الحالة ننفذ تكرار (for ...) في السطور (10 - 15) حيث ان السطر (9) يقسمان عبارتين لذا فإن (count = 2) والسطر (11) ينفذ (n) من المرات [من $i = 2$ الى $i = (n + 1)$] سيزداد (i) الى $(n + 1)$ ولكن خطوة التكرار الاخيرة ان تنفذ والسطور (11 - 15) تنفذ $(n - 1)$ من المرات والسطور (16, 17, 18) تنفذ مرة واحدة . مع ملاحظة ان الوقت الفعلي لتنفيذ كل عبارة يختلف عن وقت تنفيذ العبارة الاخرى ويمكن تلخيص هذه المعلومات في الجدول التالي :

Line	Count	Line	Count
4	1	13	$n - 1$
5	1	14	$n - 1$
6	2	15	$n - 1$
9	1	16	1
10	2	17	1
12	n	18	1
	$n - 1$		
Total		$5n + 3$	

+ ان وقت $O(n)$, $O(n \log n)$ يزداد كل منهما بصورة ابطا من الدوال الاخرى
 + عندما يكون حجم البيانات كبيرا تصبح الخوارزمية التي وقت تنفيذها اكبر
 من $O(n \log n)$ غير واقعية (غير عملية)
 + الخوارزمية التي وقت تنفيذها يأخذ الصيغة الاسية (exponential) بالامكان
 اعتمادها عندما تكون (n) صغيرة جدا.



الشكل (2-8) : معدلات نمو بعض دوال احسابي وقت التنفيذ
 (Computing Time Functions)

اما الجدول في الشكل (8 - 3) فيوضح مقارنة بين قيم متغيرات تلك الدوال

$\log_2 n$	N	$N \log_2 n$	n^2	N^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	2,147,483,648

الشكل (8 - 3) جدول مقارنة بين قيم دوال الاحساب

$O(2^n)$ يعني الوقت اللازم لجعل جميع عناصر المصفوفة $(N * N * \dots * N)$ مساوريا للصفر . وهذه هي الصيغة الاسية (exponential).
 $O(\log n)$ تمثل وقت الوصول الى عدة نهائية (ورقة) في شجرة ثنائية . وهذه هي صيغة لو غاريتمية (logarithmic).

عندما تكون لدينا خوارزمتان تنفذ نفس العملية ووقت تنفيذ احدهما $O(n)$ والاخرى $O(n^2)$ فان الاولى هي الافضل لان قيمة (n) تحدد هذا المقدار ويصبح وقت تنفيذ الثانية اكبر بكثير من الاولى .

فلما افترضنا ان قيمة ثبات الخوارزمية الاولى = 10 وثابت الخوارزمية الثانية = 0.5 فان وقت التنفيذ لكل قيمة من (n) يكون حسب الجدول التالي :

N	الاولى $10 * n$	الثانية $0.5 * n^2$
1	10	0.5
5	50	12.5
10	100	50
15	150	112.5
20	200	200

25	250	312.5
30	300	450

نلاحظ عندما تكون قيمة $(n < 20)$ فان وقت تنفيذ الخوارزمية الثانية يكون صغيرا ولكن بعد تلك القيمة فان الفرق يكون معكوسا بصورة كبيرة ولصالح الخوارزمية الاولى ولهذا يوضح لماذا يقسم اختيار الخوارزمية بالصفر (order of magnitude) ان اختار قيمة الثوابت يعتمد على عدة عوامل منها لغة البرمجة والحاسوب الذي يستخدم .

ان الرسم البياني في الشكل (8 - 2) يوضح معدلات نمو دوال وقت التنفيذ باقراض $(C = 1)$ حيث نلاحظ :

Example 3:

Use Big-O notation to analyze the time efficiency of this C++ segment code in term of the integer N

```
k = N
while (k > 1)
{
    -
    -
    k = k % 2
}
```

الجواب:

ان عدد مرات تكرار هذا المقطع الذي يتناقص الى النصف في كل مرة فانه يمكن تمثيله بالعلاقة $(\log_2 N)$ وباستخدام صيغة (Big-O) لهذه الخوارزمية فان وقت التنفيذ يكون $O(\log_2 N)$

Example 1: Use big-O notation to analyze the time efficiency of the following C++ code in terms of the integer N.

```
for (k = 1; k <= n % 2; k++)
```

```
for (j = 1; j <= (n * n); j++)
```

This will be executed $N/2$ times

This algorithm will be executed $N^2 * N/2 = \frac{1}{2} N^3$

ولذا فيستخدم (big-O) ولأن قيمة الثابت $(C = \frac{1}{2})$ فيكون وقت التنفيذ $O(N^3)$

Example 2:

```
for (k = 1; k <= N % 2; k++)
```

will be executed $N/2$ times

This algorithm will be executed $N/2 + N^2$ (big-O) $O(N^2)$ فانه سيكون

Will be executed N^2 times

4- المباشرة بوضع الحل المناسب للمسألة الذي يكون اختياره عادة من بين مجموعة حلول أو بدائل باعتباره الأفضل في ضوء المتطلبات وحاجة المستقبل.

5- كتابة خوارزمية الحل وهي مجموعة محددة من التعليمات (خطوات الحل) التي تؤدي إلى إنجاز العمل المطلوب.

ويجب أن تتوافر فيها الموصفات الآتية :

- أ- لها مخلات Input
- ب- لها مخرجات output
- ج- تتصف بالوضوح (definiteness) أي أن كل خطوة فيها واضحة المعنى وغير غامضة
- د- محددة الخطوات finiteness
- هـ- فعالة (effectiveness) أي أن خطوات الحل عملية وقابلة للتنفيذ.

8-2 مزايا البرنامج الجيد

إن الجانب التطبيقي في علوم الحاسبات مهم جداً في معرفة ورؤية الخالق أو تطبيق الأفكار بحيث نستطيع قوبرها. والبرمجة هي إحدى الأدوات المساعدة لهذا الغرض ، لذا فإن كتابة برنامج ينتج الحل فقط دون توفر الشروط النوعية الجيدة تكون له مردودات سلبية كبيرة وتعجز معها قول النتائج.

إن أهم الخصائص النوعية الجيدة التي يجب توافرها في البرنامج هي كالآتي :-

1- البرنامج يؤدي العمل :

يجب أن يؤدي البرنامج العمل المطلوب وفق الموصفات المعروفة لوظيفته ومخرجاته ومخرجاته والمعالجات التي يؤديها .

2- البرنامج قابل للقراءة والفهم

إن البرنامج المصمم يجب أن يكون واضح القراءة بصياغة تسهل فهم خطواته ووظائفه من قبل الآخرين .

3- البرنامج قابل للتطوير

يجب أن يكون البرنامج مصمماً بطريقة يمكن تطويره لاحقاً أو تعديله بسهولة إذا تطلب الأمر ذلك بجهود مقبولة ووقت مناسب ، إذ أن الضرورة قد تقتضي تعديل البرنامج في مرحلة التصميم أو الترميز أو الاختبار .

4- إنجاز البرنامج

يجب أن يعد البرنامج بصورته المتكاملة ويمر لحله المختلفة ضمن الفترة الزمنية والتخصيص المالي المحددين له وأي تجاوز على ذلك سيقته جواه الاقتصادية .

مع مراعاة الخصائص آنفة الذكر فإن البرنامج يعد بموجب موصافه المعدة مسبقاً (Program Descriptions) ويفضل اعتماد الأفكار المبينة إن شاء عند إعداد البرنامج:

- 1- التفكير الجيد بالمسألة وفيها في ضوء الموصفات التي قد يتطلب قراءتها أكثر من مرة .
- 2- الاستفسار وطلب زيادة التوضيح للقررات التي تتطلب ذلك من الاطراف ذات العلاقة وخصوصاً المستقبل من البرنامج أو المدير المسؤول أو رئيس فريق العمل .
- 3- التوسع في تفصيل الموصفات لكل من فعاليات الإدخال والمعالجة والإخراج .

8- 3 تصميم البرنامج Program Design

التصميم من الأعلى للأسفل Top-Down Design

وهي طريقة تعتمد على فكرة تجزئة المسألة الى مسائل فرعية اصغر قابلية للاستعاب والحل .

ان أسلوب التجزئة هذا يسمح بوضوح الهيكل العام للمسألة من خلال معرفة مكوناته ، وهنا يمكن مناقشة هذا الهيكل مع الآخرين وخصوصا الطرف المنتقد لمعرفة مدى وضوح الصورة للجميع بحيث يمكن معالجة حالات الغموض او عدم الفهم التي قد تكون لدى احد الاطراف سواء كن منهم المستفيد او المراقبات المكثورة او فريق العمل المكلف باعداد البرنامج ، ... الخ في وقت مبكر قبل المباشرة بالخطوات التنفيذية اللاحقة التي يكون للراجع عليها مكانا من الناحية الزمنية والمالية .

ان تجزئة المسألة الى مسائل اصغر يسمح بمعالجتها كوحدات مستقلة الى حد ما تدعى (Modules) وقد نستطيع تجزئة هذه الوحدات الى فعاليات اصغر للنفس الغرض مثال : تصميم حل مسألة ترتيب طلاب احد الصفوف حسب معدلهم العام وطبع النتيجة .

الحل :

المستوى الاول :

حسب المتطلبات فان البرنامج قد يتكون من الوحدات المنطقية الآتية :

- افعال البيانات

- ترتيبها

- طبع النتائج

أي ان الهيكل العام للبرنامج يصبح :

Program class rank ;

Begin

Get data

Sort data

Print data

End

المستوى الثاني :

يمكن تجزئة كل من الوحدات السابقة الى مستويات ادنى وتحديد تفاصيل مدخلاتها ومخرجاتها ، فالوحدة (Get data) يمكن تجزئتها الى وصفها بأنها تتضمن :

Read data for single student from file as a Record

المستوى الثالث :

في هذا المستوى نحتاج الى تفاصيل اكثر للتعامل مع وحدة (Get data) بشأن كيفية التعامل مع قيد الطالب (Student Record) ومكوناته (رقم الطالب - الصف - المعدل العام) .

المستويات اللاحقة :

وهكذا نستمر في التدرج والانتقال من مستوى علم الى مستوى ادنى يتضمن تفاصيل اكثر وفق متطلبات تلك المستوى ، وستبرز عملية التصميم هذه وكأنها هيكل شجرة حيث الفكرة العامة للمسألة (خوارزمية الحل) في قمة الشجرة والتفاصيل تتفرع نحو المستويات الادنى كما في الشكل (4 - 8) .

وبهذه الطريقة نجد ان المستويات العليا في التصميم لا تعتمد على تصميم التفاصيل في المستويات الادنى :