

جامعة بغداد / كلية التربية للعلوم الصرفة ابن الهيثم / قسم علوم الحاسبات
المرحلة الاولى / نظري
السنة الدراسية 2024-2025

Structured Programming

Lecture1 – Algorithms

Algorithm

Algorithm can be defined as: “A sequence of activities to be processed for getting desired output from a given input”.

Then we can say that:

1. Getting specified output is essential after the algorithm is executed.
2. One will get output only if the algorithm stops after a finite time.
3. Activities in an algorithm to be clearly defined in other words for it to be unambiguous

Algorithm

Before writing an algorithm for a problem, one should find out what is the:

- The inputs to the algorithm.
- The output after running the algorithm.

While writing algorithms we will use the following symbols for different operations:

- '+' for Addition
 - '-' for Subtraction
 - '*' for Multiplication
 - '/' for Division
 - ' \leftarrow ' for assignment.
- For example $A \leftarrow X * 3$ means A will have a value of $X * 3$.

Examples of Algorithm

Example 1: Find the area of a Circle of radius r .

- Sol: Inputs to the algorithm:
- Radius r of the circle.
- Expected output:
- Area of the circle Algorithm.

Algorithm:

- Step1: Start
- Step2: Read\input the Radius r of the Circle
- Step3: Input PI 3.14
- Step4: $\text{Area} \leftarrow \text{PI} * r * r$ // calculation of area
- Step5: Print Area
- Step6: End

Examples of Algorithm

- **Example 2:** Write an algorithm to read two numbers and find their sum.
- Sol: Inputs to the algorithm:
- First num1.
- Second num2.
- Expected output:
- Sum of the two numbers.

Algorithm:

- Step1: Start
- Step2: Read\input the first num1.
- Step3: Read\input the second num2.
- Step4: $\text{Sum} \leftarrow \text{num1} + \text{num2}$ // calculation of sum
- Step5: Print Sum
- Step6: End

Types of Algorithms

- The algorithm and flowchart, classification to the three types of control structures. They are:
- 1. Sequence The sequence structure is the construct where one statement is executed after another.
- 2. Selection The selection structure is the construct where statements can be executed or skipped depending on whether a condition evaluates to TRUE or FALSE.
- There are three selection structures in C:
 - 1. IF
 - 2. IF – ELSE
 - 3. SWITCH

Types of Algorithms

- 3. Loop (Repetition) The repetition structure is the construct where statements can be executed repeatedly until a condition evaluates to TRUE or FALSE.
- There are three repetition structures in C:
 - 1. WHILE
 - 2. DO – WHILE
 - 3. FOR

Examples of Algorithm

- **Example 3:** Write an algorithm to find the greater number between two numbers
- Sol:
- Step1: Start
- Step2: Read/input A and B
- Step3: If $A \geq B$ then $C \leftarrow A$ // A greater than or equal B
- Step4: if $B \geq A$ then $C \leftarrow B$ // B greater than or equal A
- Step5: Print C
- Step6: End

Examples of Algorithm

- **Example 4:** Write an algorithm to find the average of any three numbers.
- Sol: Step1: Start
- Step2: Read values of X, Y, Z
- Step3: $S \leftarrow X + Y + Z$
- Step4: $A \leftarrow S / 3$
- Step5: Print value of A
- Step6: End

Examples of Algorithm

- **Example 5:** Write an algorithm to find the largest value of any three numbers.
- Sol:
- Step1: Start
- Step2: Read/input A, B and C
- Step3: If $(A \geq B)$ and $(A \geq C)$ then $\text{Max} \leftarrow A$
- Step4: If $(B \geq A)$ and $(B \geq C)$ then $\text{Max} \leftarrow B$
- Step5: If $(C \geq A)$ and $(C \geq B)$ then $\text{Max} \leftarrow C$
- Step6: Print Max
- Step7: End

Examples of Algorithm

- **Example 6:** Write an algorithm to calculate and print even numbers between 0 and 99.
- Sol:
- 1. Start
- 2. $I \leftarrow 0$
- 3. Print I
- 4. $I \leftarrow I + 2$
- 5. If $(I \leq 98)$ then go to line 3
- 6. End

Exercises

- **Q1:** Write an algorithm to calculate even numbers between 9 and 100.
- **Q2:** Write an algorithm to calculate and print odd numbers between 1 and 95.
- **Q3:** Write an algorithm to find the sum of 50 numbers.
- **Q4:** Write an algorithm to find the value of A, B, C.

$$A = X + 6Y$$

$$B = 2X - A$$

$$C = A + XB$$

- **Q5:** Write an algorithm to print the series. 2,4,8,16,32.....1024.

جامعة بغداد / كلية التربية للعلوم الصرفة ابن الهيثم / قسم علوم الحاسبات
المرحلة الاولى / نظري
السنة الدراسية 2024-2025

Structured Programming







Lecture2 – Flowcharts

Flowcharts

The flowchart is a diagram that visually presents the flow of data through processing systems. Algorithms are nothing but a sequence of steps for solving problems. So a flowchart can be used for representing an algorithm. A flowchart will describe the operations are required to solve a given problem.

Flowchart Symbols There are 6 basic symbols commonly used in the flowcharting of assembly language Programs: Process, input/output, Decision, Connector, Terminal, and Flow Line Process. This is not a complete list of all the possible flowcharting symbols, it is the ones used most often in the structure of Assembly language programming.

Flowcharts

Symbol	Name	Function
	Process	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	connects sections of the flowchart, so that the diagram can maintain a smooth, linear flow
	Terminal	indicates start or end of the program or algorithm
	Flow Lines	Shows direction of flow

General Rules for flowcharting

1. All boxes of the flowchart are connected with Arrows. (Not lines)
2. Flowchart symbols have an entry point on the top of the symbol with no other entry points. The exit point for all flowchart symbols is on the bottom except for the Decision symbol.
3. The Decision symbol has two exit points; these can be on the sides or the bottom and one side.
4. Generally, a flowchart will flow from top to bottom.
5. Connectors are used to connect breaks in the flowchart. Examples are:
 - From one page to another page.
 - From the bottom of the page to the top of the same page.
6. All flowcharts are start and end with a terminal symbol.

Types of flowchart

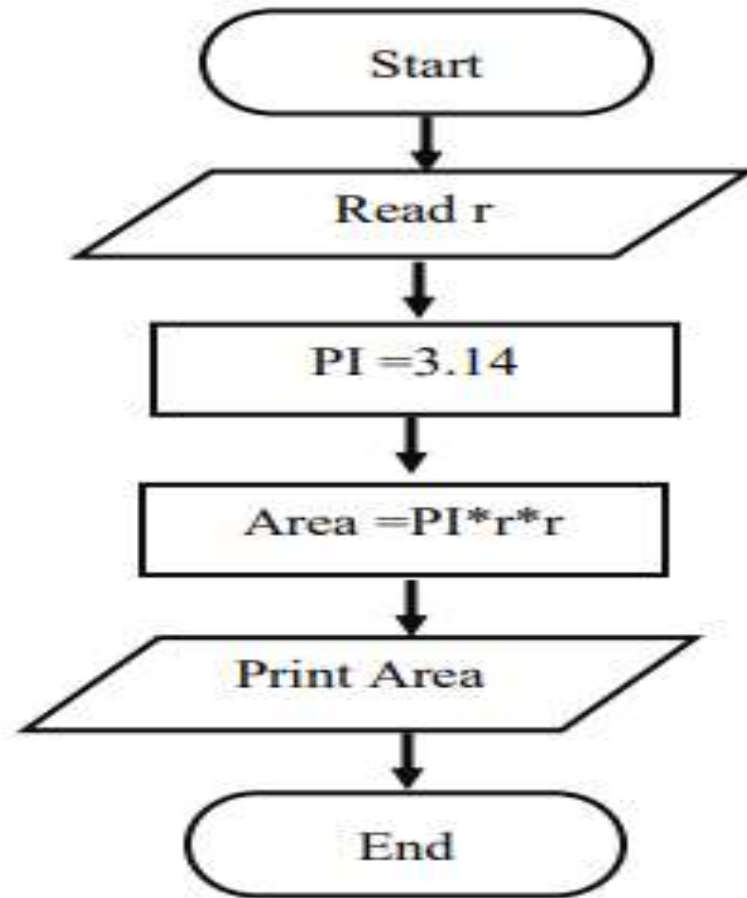
There are three types from flowcharts:

1. Simple Sequential Flowcharts
2. Branched Flowcharts
3. Loop Flowcharts

- Simple Sequential Flowcharts

The solution steps for this type of flowchart are arranged in a straight series from the beginning of the program to the end so that it is free of branches and loops

Example 1: Draw a flowchart to find the area of a circle of radius r .



Algorithm

Step1: Start

Step2: Read\input r

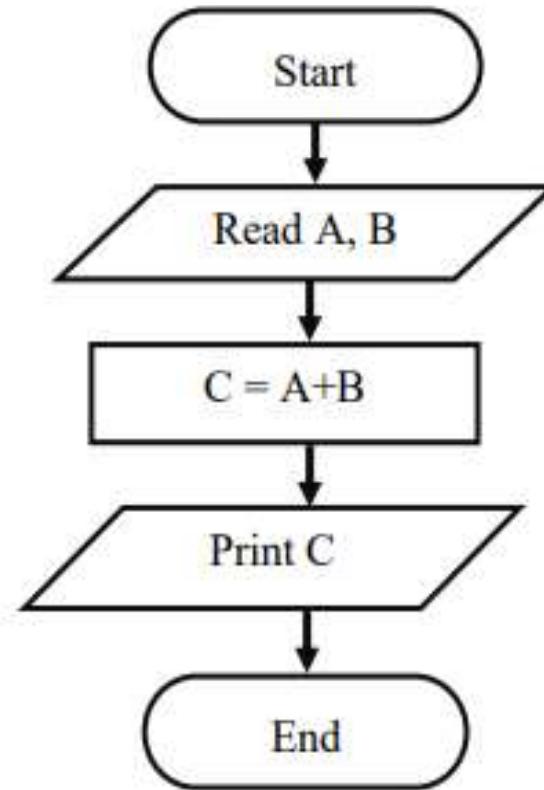
Step3: Input $\text{PI} \leftarrow 3.14$

Step4: $\text{Area} \leftarrow \text{PI} * r * r$

Step5: Print Area

Step6: End

Example 2: Draw a flowchart that gets two numbers and prints the sum of their value.



Algorithm

Step1: Start

Step2: Input A,B

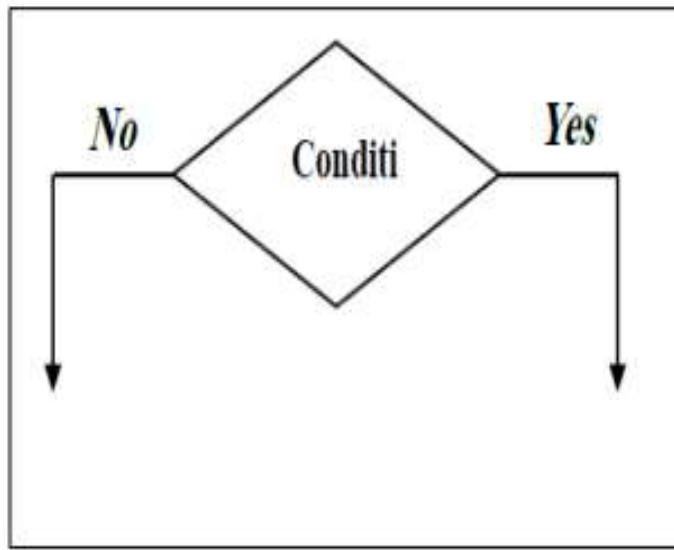
Step3: $C \leftarrow A+B$

Step4: Output C

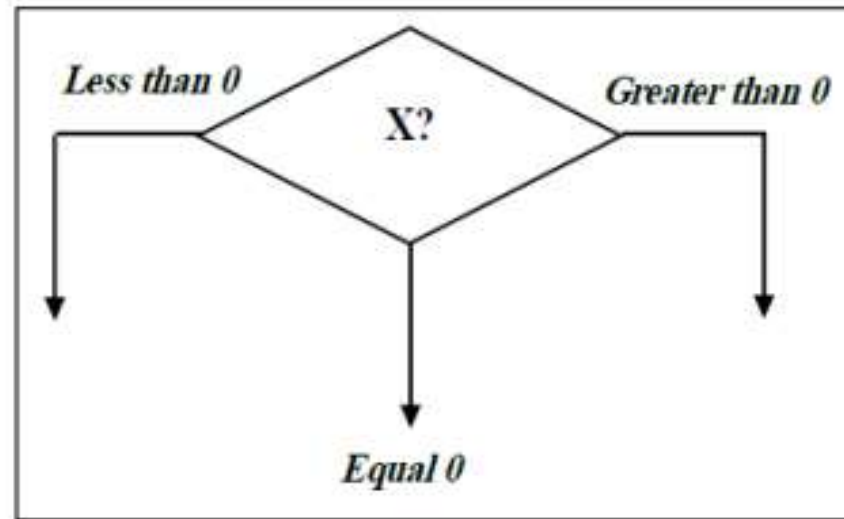
Step5: End

Types of flowchart

- **Branched Flowcharts**
- This type of flowchart is used when there is a need to make a decision or a tradeoff between two or more choices.
- There are two types of branched flowcharts as shown below:

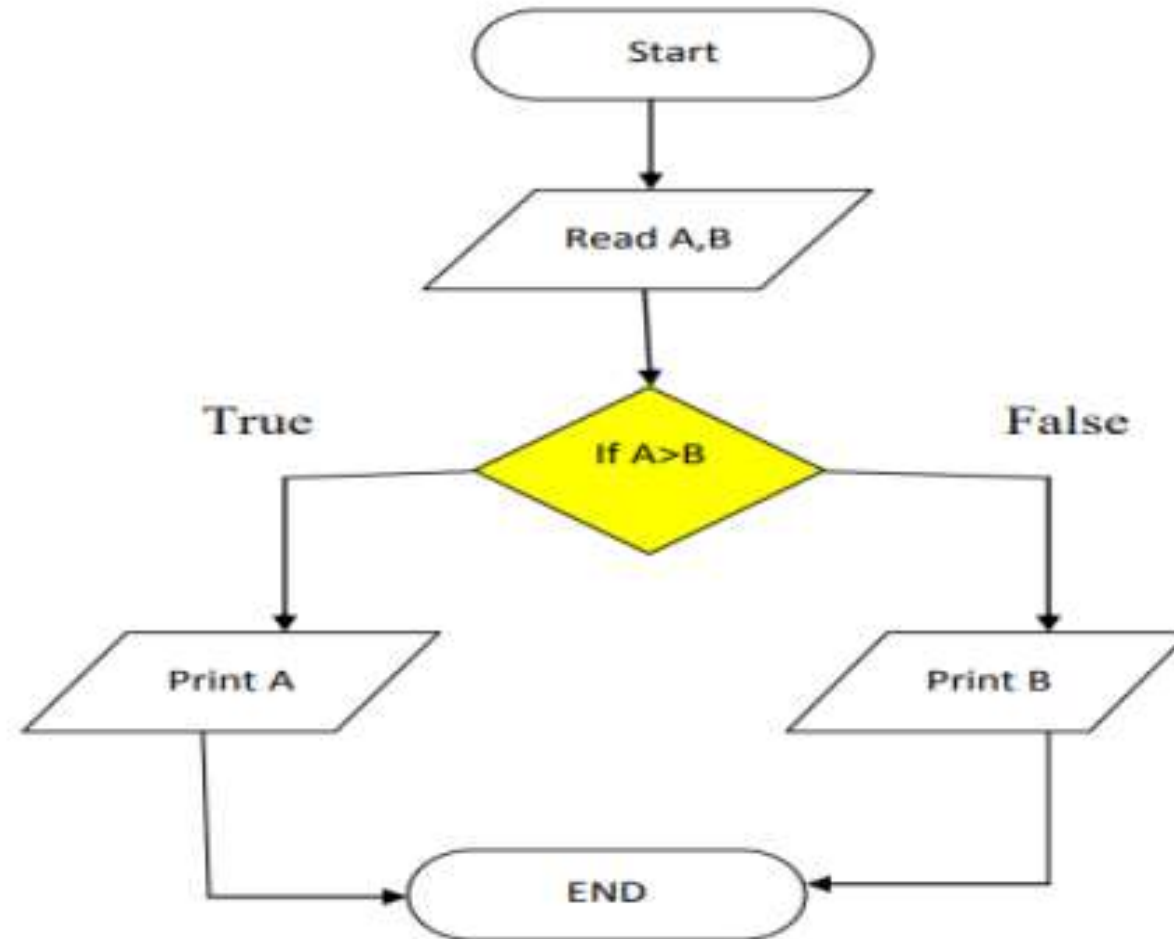


Two branches decision



Three branches decision

Example 3: Draw a flowchart to find the greater number between two numbers.



Algorithm

Step1: Start

Step2: Read/input A and B

Step3: If $A > B$ then

Step4: Print A

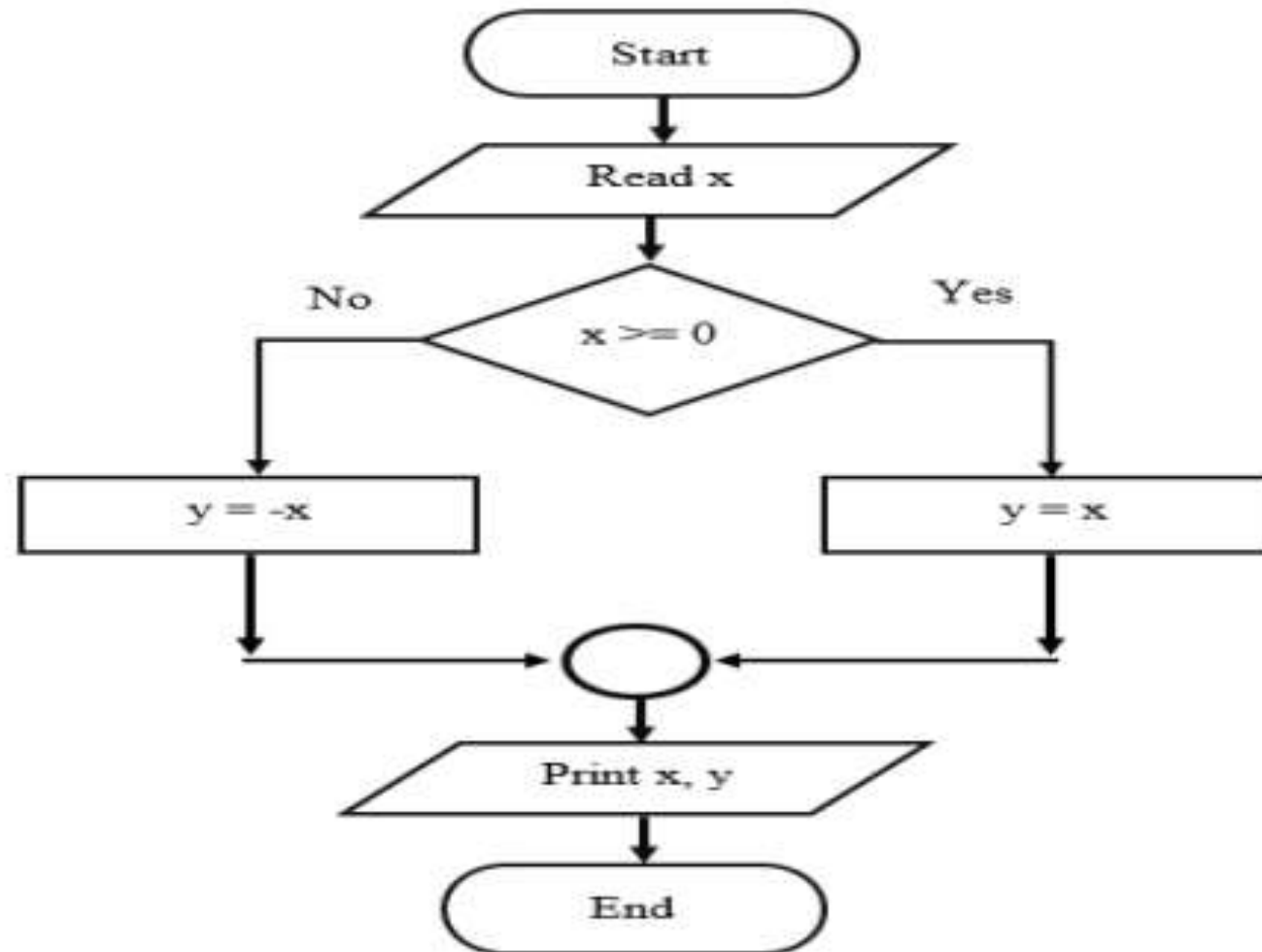
Step4: if $B > A$ then

Step5: Print B

Step6: End

Example 4: Draw a flowchart to find the value of y .

$$y = \begin{cases} -x, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases}$$



Step1: Start

Step2: input X

Step3: if $x \geq 0$ then $y = x$

Step4: if $x < 0$ then $y = -x$

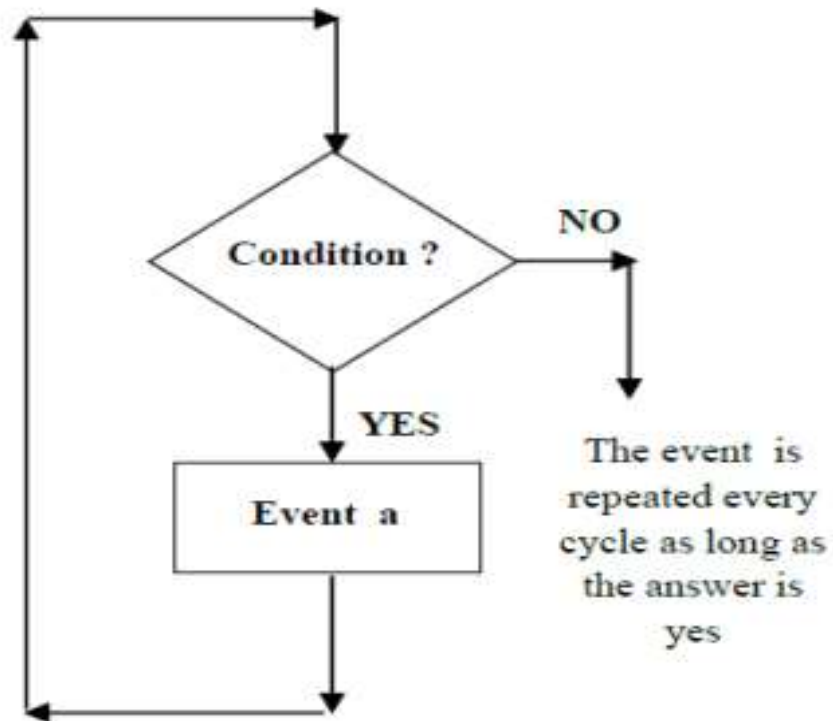
Step5: Print y

Step 6: End

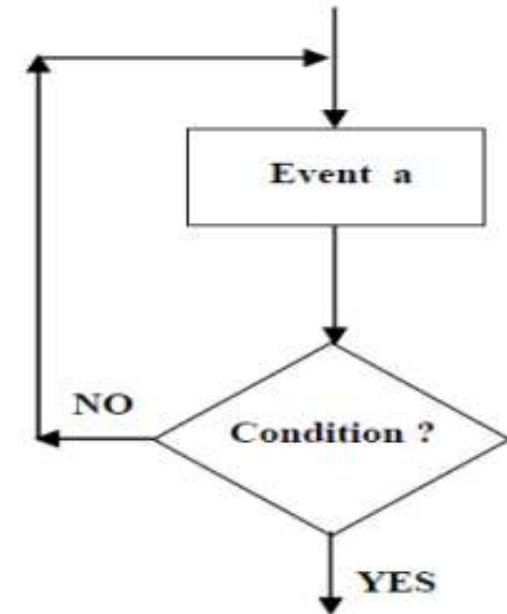
Types of flowchart

- Loop Flowcharts

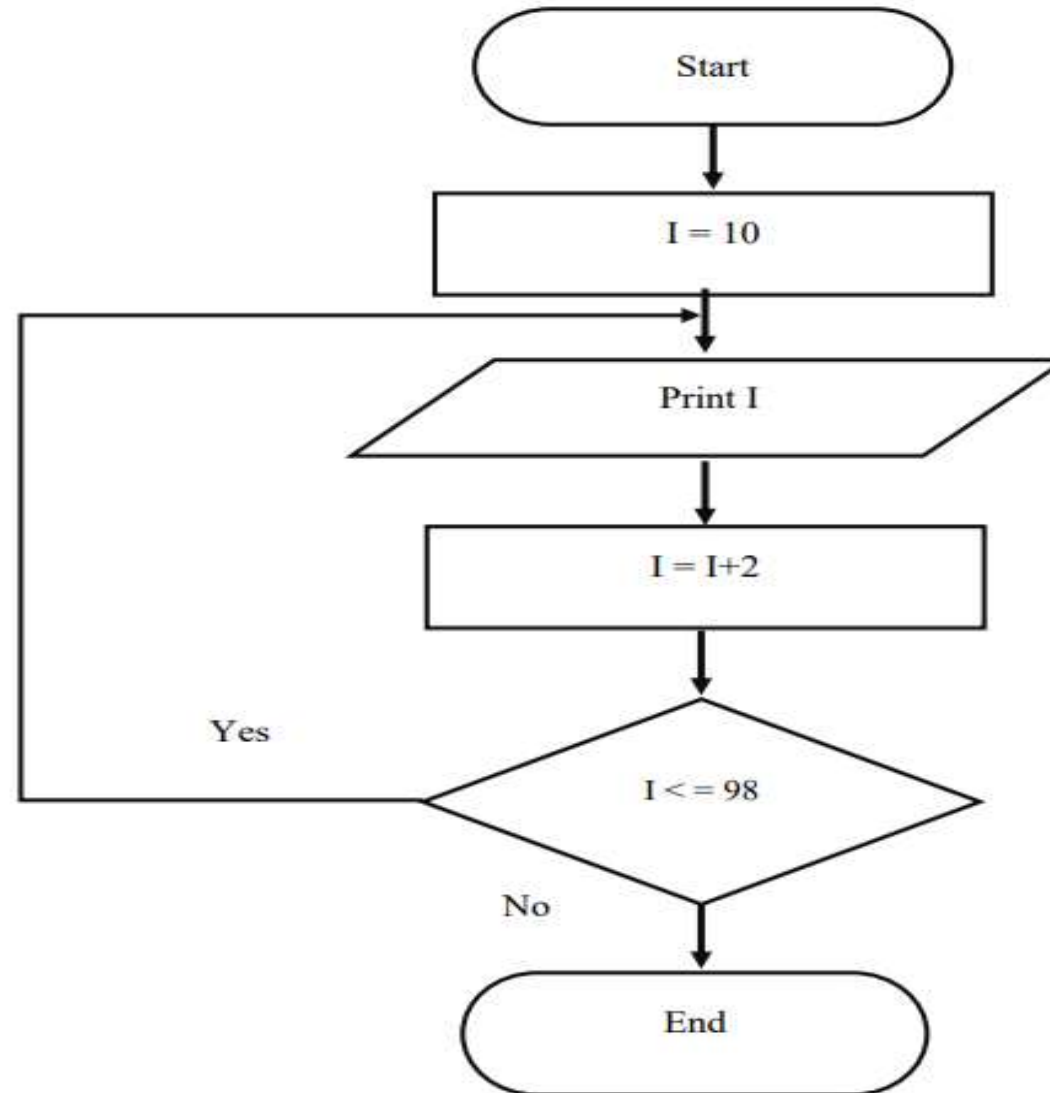
In this type of flowchart, we need to return a process or set of processes in the program a specific or unspecified number of times, and the overall form of these flowcharts is as follows:



The event is repeated every cycle until the answer is yes



Example 5: Draw a flowchart for printing even numbers between 9 and 99.



Algorithm

Step1: Start

Step2: $i = 10$

Step3: Print i

Step4: $i = i + 2$

Step5: If $(i \leq 98)$ then
go to line 3

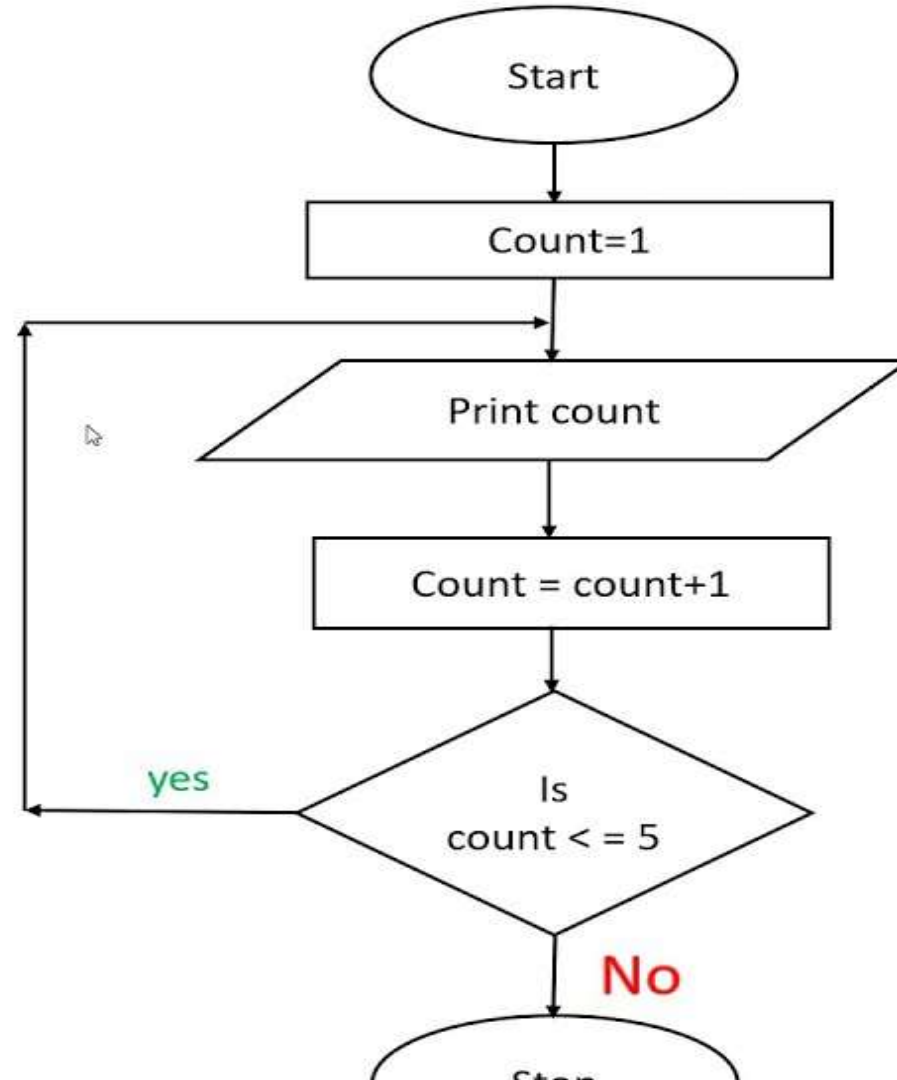
Step6: End

Example 6: Draw a flowchart that print the numbers between 1 to 5

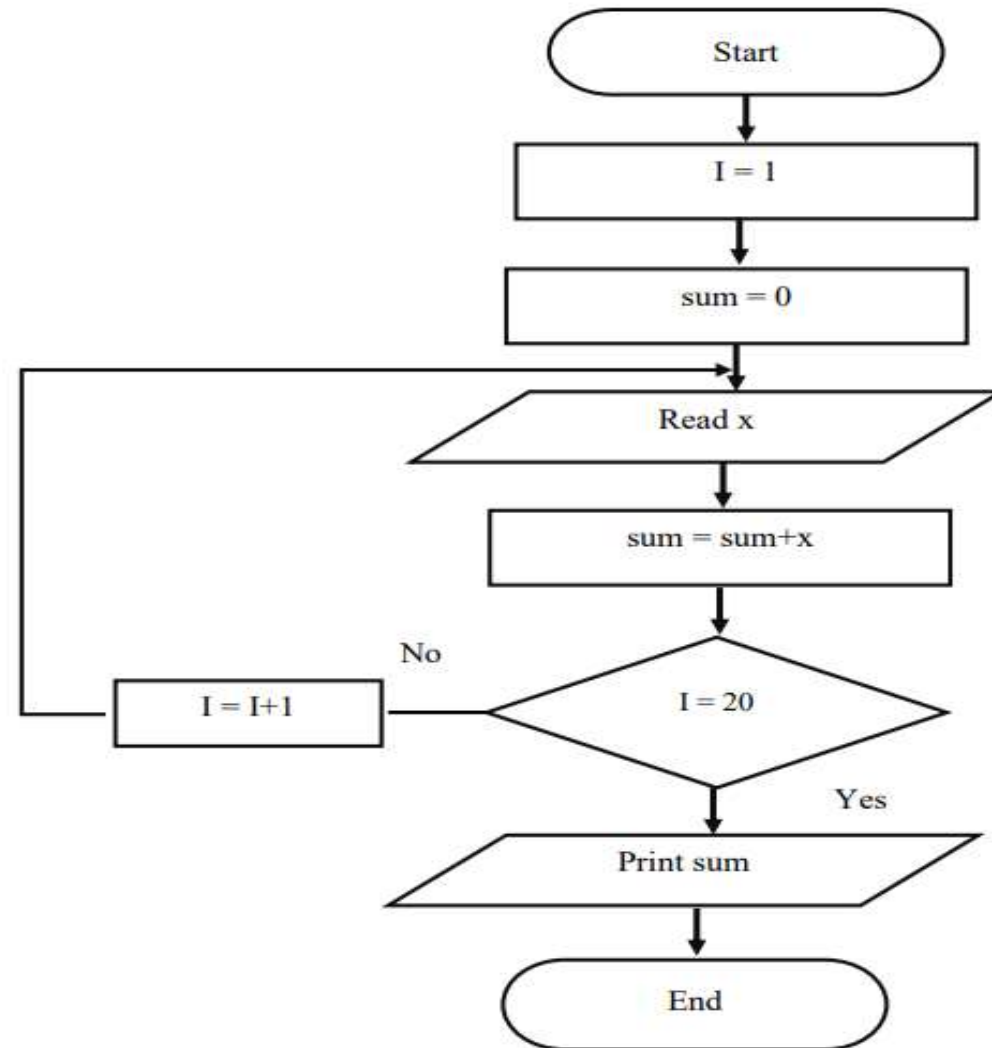
Flowchart to print 1 to 5 .

count
6 **+1**

1
2
3
4
5



Example 7: Draw a flowchart to find the sum of 20 numbers.



Exercises

1. Draw a flowchart to find the area and perimeter of circle have known radius (R).
2. Draw a flowchart to find the values of variables A, B, and C from the following equations

$$A = X^2 + 2Y$$

$$B = 2X - 3A$$

$$C = A^2 + XB$$

3. Draw a flowchart to print the series: 5,8,11,14,17.....50
4. Draw a flowchart to check the input number is odd or even.
5. Draw a flowchart to print the series: 2,4,8,16,32.....1024.

جامعة بغداد / كلية التربية للعلوم الصرفة ابن الهيثم / قسم علوم الحاسبات
المرحلة الاولى / نظري
السنة الدراسية 2024-2025

Structured Programming

Lecture3 – C++Programming Language

Introduction to C++

C++: is a programming language.

- Programming language: A set of rules, symbols, and special words.
- Source program: A program written in a high-level language.
- The following steps, as shown in figure 1, are necessary to process a C++ program.
 1. You use a text editor to create a C++ program following the rules, or syntax, of the high-level language. This program is called the source code, or source program. The program must be saved in a text file that has the extension .cpp.

Introduction to C++

2. In a C++ program, statements that begin with the symbol # are called preprocessor directives. These statements are processed by a program called preprocessor.
3. The next step is to verify that the program obeys the rules of the programming language—that is, the program is syntactically correct—and the compiler translates the program into the equivalent machine language which is called object program.

Introduction to C++

4. The programs that you write are developed using an integrated development environment (IDE). The IDE contains many programs that are useful in creating your program. Once the program is developed and successfully compiled, you must still bring the code for the resources used from the IDE into your program to produce a final program that the computer can execute.

This prewritten code (program) resides in a place called the library. A program called a linker combines the object program with the programs from libraries to create the executable code.

Introduction to C++

5. You must next load the executable program into the main memory for execution. A program called a loader accomplishes this task
6. The final step is to execute the program.

As a programmer, you need to be concerned only with Step 1. That is, you must learn, understand, and master the rules of the programming language to create source programs

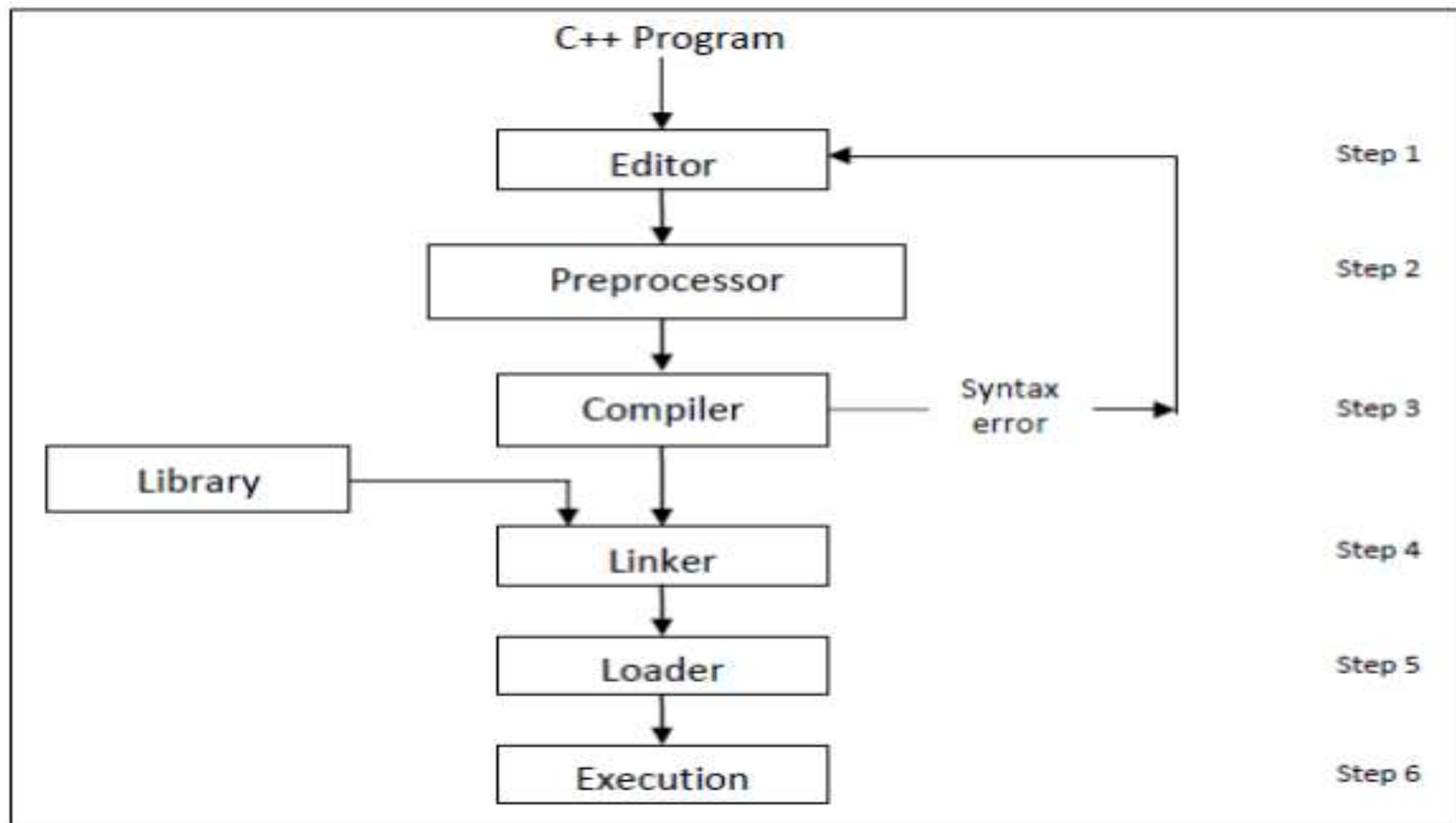


Figure 1: processing a C++ program

The Basics of a C++ Program

1. A C++ program is a collection of one or more subprograms, called functions.
2. Every C++ program has a function called main. Thus, if a C++ program has only one function, it must be the function main.
3. To write meaningful programs, you must learn the programming language's.

The Basics of a C++ Program

- Special symbols, words, syntax rules, and semantic rules.
- **Syntax rules** tell you which statements (instructions) are legal, or accepted by the programming language, and which are not.
- **Semantic rules** tell you which determine the meaning of the instructions.

4. Function: is a collection of statements, and when it is executed, it accomplishes something.

Comments in C++ Program

1. It is a notes which putting on your program to explain something.
2. It gives a brief explanation of the program, and explain the meaning of key statements in a program.
3. Comments are for the reader, not for the compiler. So when a compiler compiles a program to check for the syntax errors, it completely ignores comments.

Comments in C++ Program

4. There are two common types of comments in a C++ program.

- **Single-line comments** begin with `//` and can be placed anywhere in the line.

Ex: `cout << "7 + 8 = " << 7 + 8 << endl; //prints: 7 + 8 = 15`

- **Multiple-line comments** are enclosed between `/*` and `*/`. The compiler ignores anything that appears between `/*` and `*/`.
- For example, the following is an example of a multiple-line comment:
`/*`
- You can include comments that can occupy several lines.
- `*/`

Token in C++ Program

- The smallest individual unit of a program written in any language is called a token. C++'s tokens are divided into:
 1. Special symbols.
 2. Reserved word.
 3. Identifiers.

1. Special symbols

The following are some of special symbols:

- Mathematical symbols: include (+ - * /) for addition, subtraction, multiplication, and division.

Token in C++ Program

- **Punctuation marks:** include (. ; ? ,). These marks are taken from English grammar. Note that the comma is also a special symbol. In C++, commas are used to separate items in a list. Semicolons are used to end a C++ statement.
- **Comparison symbols:** include (< =, >, > =, !=, ==) for less than, less and equal, greater than, greater and equal, not equal, equal)
- **Character set:** C++ has the letters and digits, as show below:
 - Uppercase: A, B, C, . . . , Z
 - Lowercase: a, b, c, . . . , z
 - Digits: 0, 1, 2, . . . ,9

Reserved words

- Some words are reserved by C++ (are parts of the C++ language) such as main, for, while, if, int, float,... Reserved words can't be used as variable names or constant. Table 1 explains some examples of reserved words.

Table 1: Examples of reserved words

Some of C++ language Reserved words:				
break	case	char	cin	cout
delete	double	else	enum	false
float	for	goto	if	do
long	main	private	public	short
int	switch	true	while	void

Identifiers

- Identifiers are names of things that appear in programs, such as a variable, a constant, an array, a function, a structure, or a class. Table 2 shows some illegal identifier description.

Table 2: Examples of Illegal Identifiers

Illegal Identifier	Description
employee Salary	There can be no space between employee and Salary.
Hello!	The exclamation mark cannot be used in an identifier
one + two	The symbol + cannot be used in an identifier.
2nd	An identifier cannot begin with a digit.

- **Note:** C++ is case sensitive—uppercase and lowercase letters are considered different. Thus, the identifier NUMBER is not the same as the identifier number. Similarly, the identifiers X and x are different.
- **Whitespaces** In a C++ program, whitespaces are used to separate special symbols, reserved words, and identifiers. Whitespaces are nonprintable in the sense that when they are printed on a white sheet of paper, the space between special symbols, reserved words, and identifiers is white. Proper utilization of whitespaces in a program is important. They can be used to make the program readable.
- Example:
 - `int x;`
whitespace

Getting Started with C++

- To begin learning C++ let's examine the structure of C++ Program:

```
#include<iostream.h>
```

```
main ( )
```

```
{
```

```
// A statements for main program
```

```
}
```

- `#include<iostream.h>`
- This line is for pre-processor directive. Any begins with # is processed before the program is compiled. C++ programs must be start with #include.
- Every group of related functions is stored in a separate library called (header file). To use the cin and cout, must include the header file iostream.

main () is the name of C++ function.

- Every C++ program must have a function called main.
{
- indicates the start statements that define the function.
}
- indicates the end of the statements in the function.

- **cout <<**
- it is a C++ output statement. It causes the computer to evaluate the expression after the pair of symbols << and display the result on the screen.
- Note: << is an operator, called the stream insertion operator. (or send operator)
- **cin >>**
- The input stream object. It reads the input values from the keyboard.
- Note: >> is an operator, called the stream extraction operator (or get from operator)
- ; , semicolon, the terminator of every C++ statement.

- Note: The endl is used in C++ to represent a new line, also
- \n is a special escape code, also used in C++ to represent a new line, as shown in the following example:

Example 1: Write a C++ program to print any statement.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
cout << "welcome" << endl ;
cout << "students" ;
getch ( ) ;
}
```

Output:

welcome
students

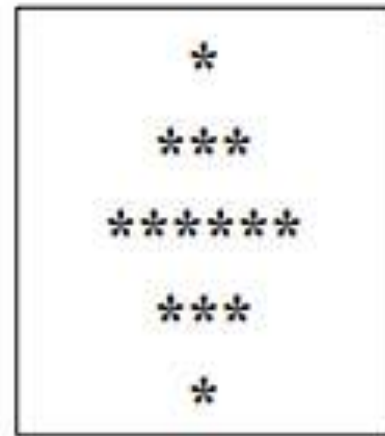
```
#include<iostream.h>
# include<conio.h>
main ( )
{
    cout << "welcome"<< "\n" ;
    cout << "students" ;
    getch ( ) ;
}
```

Output:

welcome
students

Example 2: Write a C++ program to print the figure.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
cout << "  * \n";
cout << " *** \n";
cout << " ***** \n";
cout << " *** \n";
cout << "  * \n";
getch ( );
}
```



C++ provides escape sequences for several usages. These escape sequences are listed below:

C++ provides escape sequences for several usages. These escape sequences are listed below:

Escape	Description
\n	newline
\r	carriage return. return the cursor to the beginning of the current line
\t	tab
\a	alert (beep) صوت او تنبيه
\'	single quote (')
\"	double quote (")
\?	question mark (?)
\\	backslash (\)

Example 3: Write a C++ program to represent all escape sequences in the above table.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
cout << "welcome students \n" ;
cout << "welcome students \r" << "\n" ;
cout << "welcome \t students " << "\n" ;
cout << "welcome students \a" << "\n" ;
cout << "welcome students \' " << "\n" ;
cout << "welcome students \'" << "\n" ;
cout << "welcome students \?" << "\n" ;
cout << "welcome students \\" << "\n" ;
getch ( ) ;
}
```

Output:

```
welcome students
welcome students
welcome      students
welcome students
welcome students '
welcome students "
welcome students ?
welcome students \
```

جامعة بغداد / كلية التربية للعلوم الصرفة ابن الهيثم / قسم علوم الحاسبات
المرحلة الاولى / نظري
السنة الدراسية 2024-2025

Structured Programming

Lecture4 – Variables Declaration in C++

Variables Declaration in C++

- In some programs, data needs to be modified during program execution. This type of data must be stored in those memory cells whose contents can be modified during program execution.
- In C++, memory cells whose contents can be modified during program execution are called variables.
- **Variable** is a memory location whose content may change during program execution.
- A variable defined by stating its type, followed by one or more spaces, followed by the one or more variable names separated by commas, then followed by semicolon. The syntax for declaring one variable or multiple variables is: **data type identifier, identifier,.....;**

- For example:

```
int x ;
```

```
unsigned short int X ;
```

```
float Y ;
```

```
char A, a, c ;
```

All variable must be declared with a name (an identifier) and a data type.

The types of variables used in C++ programs are described in the following table.

Note: C++ does distinguish between A and a variables (C++ is case-sensitive).

Table 3: Variables Types

Type	Size	Range and Value
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
short int	2bytes	-32768 to 32767
unsigned short int	2bytes	0 to 65,535
long int	4bytes	-2,147,483,647 to 2,147,483,647
unsigned long int	4bytes	0 to 4,294,967,295
float	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	8bytes	+/- 1.7e +/- 308 (~15 digits)
char	1byte	-127 to 127 or 0 to 255
bool	1byte	true or false

Example 1: Write a C++ to read three different inputs and print outputs it.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int n = 3 ; float f = 3.5 ; char ch = 'a' ;
cout << "number=" << n << "\n" ;
cout << "input decimal number=" << f << "\n" ;
cout << "input character=" << ch << "\n" ;
getch ( ) ;
}
```

Output:

number=3
input decimal number=3.5
input character=a

Example 2: Write a C++ to read four different inputs and print outputs it.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
unsigned int a, b ;
double c, d ;
cout << "input unsigned integer number:" ;
cin >> a >> b ;
cout << "a=" << a << "\n" ;
cout << "b=" << b << "\n" ;
cout << "input double number:" ;
cin >> c >> d;
cout << "c=" << c << "\n" ;
cout << "d=" << d << "\n" ;
getch ( ) ;
}
```

Output:

```
input unsigned integer number:4 8
a=4
b=8
input double number:3.8 9.12
c=3.8
d=9.12
```

Putting Data into Variables

- In C++, you can place data into a variable in two ways:

1. Use C++'s assignment statement.
2. Use input (read) statements.

Assignment Statement The assignment statement takes the following form:

variable = expression;

In an assignment statement, the value of the expression should match the data type of the variable. The expression on the right side is evaluated, and its value is assigned to the variable (and thus to a memory location) on the left side. A variable is said to be initialized the first time a value is placed in the variable. In C++, = is called the assignment operator.

Example 3: This program illustrates how data in the variables are manipulated.

```
#include<iostream.h>
#include<conio.h>
main ( )
{
int num1, num2 ;
double sale ;
char first ;
num1 = 4 ;
cout << "num1 = " << num1 << "\n" ;
num2 = 4 * 5 - 11 ;
cout << "num2 = " << num2 << "\n" ;
sale = 0.02 * 1000;
cout << "sale = " << sale << "\n" ;
first = 'D' ;
cout << "first = " << first << "\n" ;
getch ( ) ;
}
```

Output:

num1 = 4
num2 = 9
sale = 20
first = D

- **Notes:** This statement is valid in C++:

```
num = num + 2;
```

means “evaluate whatever is in num, add 2 to it, and assign the new value to the memory location num.” The expression on the right side must be evaluated first; that value is then assigned to the memory location specified by the variable on the left side. Thus, the sequence of C++ statements:

```
num = 6;
```

```
num = num + 2;
```

and the statement:

```
num = 8;
```

Input (Read) Statement

- In this section, you will learn how to put data into variables from the standard input device, using C++'s input (or read) statements.
- Putting data into variables from the standard input device is accomplished via the use of cin and the operator >>. The syntax of cin together with >> is:

```
cin >> variable >> variable ...;
```


Example 4: This program illustrates how input statements work.

```
#include<iostream.h>
#include<conio.h>
main ( )
{
int feet, inches ;
cout << "Enter two integers separated by spaces: ";
cin >> feet >> inches ;
cout << "\n" ;
    cout << "Feet = " << feet << "\n" ;
    cout << "Inches = " << inches << "\n" ;
    getch ( ) ;
}
```

Output:

Enter two integers separated by
spaces: 44 77
Feet = 44
Inches = 77

Increment and Decrement Operators

- You will learn about two more operators: the increment and decrement operators. Suppose `count` is an `int` variable. The statement: `count = count + 1;` or `count++;` increments the value of `count` by 1. To execute this assignment statement, the computer first evaluates the expression on the right, which is `count + 1`. It then assigns this value to the variable on the left, which is `count`.
- Increment and decrement operators each have two forms, pre and post. The syntax of the increment operator is:
 - Pre-increment: `++variable`
 - Post-increment: `variable++`

Increment and Decrement Operators

The syntax of the decrement operator is:

- Pre-decrement: `--variable`
- Post-decrement: `variable--`

Let's look at some examples.

- The statement: `++count;` or: `count++;` increments the value of `count` by 1.
- Similarly, the statement: `--count;` or: `count--;` decrements the value of `count` by 1.

- **Notes:** What is the difference between the pre and post forms of these operators?
- Suppose that `x` is an int variable. If `++x` is used in an expression, first the value of `x` is incremented by 1, and then the new value of `x` is used to evaluate the expression.
- On the other hand, if `x++` is used in an expression, first the current value of `x` is used in the expression, and then the value of `x` is incremented by 1.
- The following example clarifies the difference between the pre- and post-increment operators.

Example 5: Suppose that x and y are int variables. Consider the following statements:

`x = 5 ;`

`y = ++x ;`

- The first statement assigns the value 5 to x.
- To evaluate the second statement, which uses the pre-increment operator, first the value of x is incremented to 6, and then this value, 6, is assigned to y.
- After the second statement executes, both x and y have the value 6.

- Now, consider the following statements:
- `x = 5;`
- `y = x++;`
- As before, the first statement assigns 5 to x. In the second statement, the post increment operator is applied to x.
- To execute the second statement, first the value of x, which is 5, is used to evaluate the expression, and then the value of x is incremented to 6.
- Finally, the value of the expression, which is 5, is stored in y. After the second statement executes, the value of x is 6, and the value of y is 5

Example 6: Suppose a and b are int variables and:

`a = 5; b = 2 + (++a);`

- The first statement assigns 5 to a.
- To execute the second statement, first the expression `2 + (++a)` is evaluated. Because the pre-increment operator is applied to a, first the value of a is incremented to 6.
- Then 2 is added to 6 to get 8, which is then assigned to b. Therefore, after the second statement executes, a is 6 and b is 8.
- On the other hand, after the execution of the following statements:

`a = 5;`

`b = 2 + (a++);`

The value of a is 6 while the value of b is 7.

Example 7: This program illustrates how pre and post increments statements work

pre-increment

```
#include<iostream.h>
#include<conio.h>
main ( )
{
    int y ;
    int x = 7 ;
    cout << ++x << "\n" ;
    y = x ;
    cout << y ;
    getch ( ) ;
}
```

Output:

8
8

post-increment

```
#include<iostream.h>
#include<conio.h>
main ( )
{
    int y ;
    int x = 7 ;
    cout << x++ << "\n" ;
    y = x ;
    cout << y ;
    getch ( ) ;
}
```

Output:

7
8

Operational Assignment Operators

The operational assignment operator has the form:

Variable = variable operator expression;

For example: $X = X + 5;$

$Y = Y * 10;$

The operational assignment operator can be written in the following form:

Variable operator = expression;

For example: $X += 5 ;$

$Y *= 10 ;$

Example 8: Rewrite the equivalent statements for the following examples, and find its results. Assume: $X = 2$, $Y = 3$, $Z = 4$, $V = 12$.

Example	Equivalent Statement	Result
$X += 5$	$X = X + 5$	$X = 7$
$Y -= 8$	$Y = Y - 8$	$Y = -5$
$Z *= 5$	$Z = Z * 5$	$Z = 20$
$V /= 4$	$V = V / 4$	$V = 3$

Example 9: What is the output of the program?

```
#include<iostream.h>
#include<conio.h>
main ( )
{
int x, a = 5, y = 7, z ;
x = a ++ ; or   x = a -- ;
z = ++ y ; or   z = -- y ;
cout << "x= " << x << "\t" << "a=" << a << "\n" ;
cout << "z= " << z << "\t" << "y=" << y ;
getch ( ) ;
}
```

Output: ++

x= 5 a=6
z = 8 y=8

Output: --

x= 5 a=4
z = 6 y=6

Exercises

- **Q1:** Write a C++ program that receives two integer values from the user. The program then should print the (addition), (subtraction), (multiplication), (division), and remainder after division (modulus). Your program must use only integers.
- **Q2:** Write a C++ program that receives two double values from the user. The program then should print the (addition), (subtraction), (multiplication), and (division).
- **Q3:** What is printed by the following statement:

```
cout << /* 5 */ 3 << '\n';
```

Exercises

- **Q4:** Write a C++ program to find the area of square.
- **Q5:** Given the following declaration: `int x = 2;` Indicate what each of the following C++ statements would print.
 - (a) `cout << "x"<< "\n";`
 - (b) `cout << 'x'<< "\n";`
 - (c) `cout << x << "\n";`
 - (d) `cout << "x + 1"<< "\n";`
 - (e) `cout << "x=" <<x<< "\n";`
 - (f) `cout << x + 1 << "\n";`

جامعة بغداد / كلية التربية للعلوم الصرفة ابن الهيثم / قسم علوم الحاسبات
المرحلة الاولى / نظري
السنة الدراسية 2024-2025

Structured Programming

Lecture5 – Constant Declaration in C++ and Arithmetic Operators

Constant Declaration in C++

- Some data must stay the same throughout a program. In C++, you can use a named constant to instruct a program to mark those memory locations in which data is fixed throughout program execution.
- **Named constant:** A memory location whose content is not allowed to change during program execution.
- The syntax to declare a named constant is:

```
const dataType identifier = value;
```

For example:

- `const int INCHES_IN_FOOT = 12;`
- `const int NO_OF_STUDENTS = 20;`
- `const int CENTIMETER_IN_METER = 100;`
- `const float PI = 3.14 ;`
- `Const double SALARY=20.5`

Example 1: Write a C++ program that read the radius of a circle, then computes and outputs its area.

```
#include<iostream.h>
#include<conio.h>
main ( )
{
const float pi = 3.14 ;
int r ; float c ;

cout << "enter the radius of circle:" ;

cin >> r ;

cout << "\n" ;

c = r * r * pi ;

cout << "the area of circle=" << c;
getch ( ) ;
}
```

Output:

enter the radius of circle: 5
the area of circle=78.5

Arithmetic Operators

- To perform arithmetic operations, C++ language provides five arithmetic operators:
 - (subtraction operator),
 - + (addition operator),
 - * (multiplication operator),
 - / (division operator) and
 - % (modulus or remainder operator).
- You can use the operators +, -, *, and / with both integral and floating-point data types.
- You use % with only the integral data type to find the remainder in ordinary division.

Example 2: This program illustrates the arithmetic operators.

```
#include<iostream.h>
#include<conio.h>
main ( )
{
int x , y, z , r ;
x = 7 / 2 ;
cout << "x=" << x << "\n" ;
y = 17 / ( - 3 ) ;
cout << "y=" << y << "\n" ;
z = - 17 / 3 ;
cout << "z=" << z << "\n" ;
r = - 17 / ( - 3 ) ;
cout << "r=" << r << "\n" ;
getch ( ) ;
}
```

Output:

x=3
y=-5
z=-5
r=5

Example 3: This program illustrates the modulus arithmetic operators (i.e., %).

```
#include<iostream.h>
#include<conio.h>
main ( )
{
int y1, y2 ;
y1 = 8 % 3 ;
y2 = - 17 % 3 ;
cout << " y1=" << y1 << "\n" ;
cout << " y2=" << y2 << "\n" ;
getch ( ) ;
}
```

Output:

y1=2
y2=-2

Operator Precedence and Associativity

- When different operators are used in the same expression, the normal rules of arithmetic apply. All C++ operators have a precedence and associativity:
- **Precedence** —when an expression contains two different kinds of operators.
- **Associativity** —when an expression contains two operators with the same precedence.

C++ evaluates arithmetic expressions by the following rules of operator precedence, which are generally the same in algebra.

1. Parentheses evaluation, which can be used to force the order of evaluation to occur in any desired sequence.

- $Y1 = 2 + (3 * 5)$ evaluates to 17.
- $Y2 = (2 + 3) * 5$ evaluates to 25

2. Multiplication (*), division (/), and modulus (%) operations are evaluated next. For several operations in the same expression, the evaluation is from left to right.

3. Addition (+), and subtraction (-) operations are evaluated last. For several operations in the same expression, the evaluation is from left to right. To avoid confusion, you can use parentheses to group arithmetic expressions.

4. (<=) less than or equal, (>=) greater than or equal, (>) greater than.

5. (==) equal, and (!=) not equal.

6. (&&) AND.

7. (||) OR.

8. (=) assignment operator.

For example, using the order of precedence rules,

$$3 * 7 - 6 + 2 * 5 / 4 + 6$$

means the following:

$$(((3 * 7) - 6) + ((2 * 5) / 4)) + 6$$

$$= ((21 - 6) + (10 / 4)) + 6 \quad (\text{Evaluate } *)$$

$$= ((21 - 6) + 2) + 6 \quad (\text{Evaluate } /. \text{ Note that this is an integer division.})$$

$$= (15 + 2) + 6 \quad (\text{Evaluate } -)$$

$$= 17 + 6 \quad (\text{Evaluate first } +)$$

$$= 23 \quad (\text{Evaluate } +)$$

- For example, using the order of precedence rules,

$$2 + 3 * 4$$

$$2 + (3 * 4)$$

$$2 + 12 = 14$$

- For example, using the order of associativity rules,

$$2 - 3 - 4$$

$$(2 - 3) - 4$$

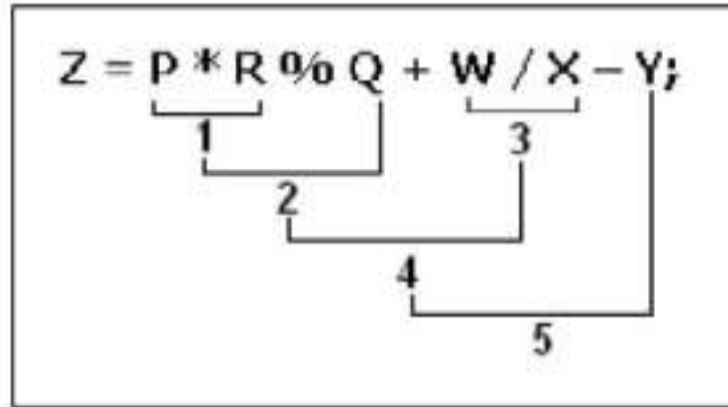
$$-1 - 4 = -5$$

Example 4: Write a C++ program to perform the equation:
 $Z = P * R \% Q + W / X - Y;$

```
#include<iostream.h>
#include<conio.h>
main ()
{
int P, R, Q, W, X, Y ;
float Z ;
cin >> P >> R >> Q >> W >> X >> Y ;

Z = P * R % Q + W / X - Y;

cout << "the result=" << Z;
getch ( ) ;
}
```



Output:

4 5 6 8 2 8
the result=-2

جامعة بغداد / كلية التربية للعلوم الصرفة ابن الهيثم / قسم علوم الحاسبات
المرحلة الاولى / نظري
السنة الدراسية 2024-2025

Structured Programming

Lecture 6 – Relational , Logical and Bitwise Operators

Relational and Equality Operators

- The relationships can be expressed in C++ by using the relational operators. These operators are listed in the following table and assume variable A holds 10 and variable B holds 20. then:

Table 4: Relational and Equality Operators

Operator	Description	Example
==	Checks if the values of two operands are equal, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then	(A <= B) is true.

Here there are some examples:

- `(7 == 5)` // evaluates to **false**.
- `(5 > 4)` // evaluates to **true**.
- `(3 != 2)` // evaluates to **true**.
- `(6 >= 6)` // evaluates to **true**.
- `(5 < 5)` // evaluates to **false**.

Also, instead of using only numeric constants, we can use any valid expression, including variables.

Suppose that `a=2`, `b=3` and `c=6`,

- `(a == 5)` // evaluates to **false** since `a` is not equal to 5.
- `(a * b >= c)` // evaluates to **true** since `(2*3 >= 6)` is true.
- `(b + 4 > a * c)` // evaluates to **false** since `(3+4 > 2*6)` is false.
- `((b = 2) == a)` // evaluates to **true**.

Logical Operators

- The logical expression is constructed from relational expressions by the use of the logical operators not(!), and(&&), or(| |)

AND (&&) Table:		
A	B	A && B
T	T	T
T	F	F
F	T	F
F	F	F

AND (&&) Table:		
A	B	A && B
1	1	1
1	0	0
0	1	0
0	0	0

OR () Table:		
A	B	A B
T	T	T
T	F	T
F	T	T
F	F	F

OR () Table:		
A	B	A B
1	1	1
1	0	1
0	1	1
0	0	0

NOT (!) Table:	
A	!A
T	F
F	T

NOT (!) Table:	
A	!A
1	0
0	1

Example 1: Assume: $a = 4$, $b = 5$, $c = 6$. find the following expression:

$(a < b) \&\& (b < c)$ T && T T	$(a < b) (b > c)$ T F T	$!(a < b) (c > b)$!(T) T F T T	$(a < b) (b > c) \&\& (a > b) (a > c)$ T F && F F T F F T F T
---------------------------------------	-------------------------------------	---	--

Example 2: Assume $X = 0$, $Y = 1$, $Z = 1$; find the following expression:

$$\begin{aligned} M &= ++X || ++Y \&\& ++Z \\ &= 1 || (2 \&\& 2) \\ &= T || (T \&\& T) \\ &= T || T \\ &= T = 1 \end{aligned}$$

Example 3: Suppose you have the following declarations:

- `bool found = true ;`
- `int age = 20 ;`
- `double hours = 45.30 ;`
- `double overTime = 15.00`
- `int count = 20 ;`
- `char ch = 'B' ;`

Consider the following
Expressions :

Expression	Value / Explanation
<code>! found</code>	False Because found is true, ! found is false.
<code>hours > 40.00</code>	true Because hours is 45.30 and 45.30 > 40.00 is true, the expression <code>hours > 40.00</code> evaluates to true.
<code>! age</code>	false age is 20, which is nonzero, so age is true. Therefore, ! age is false.
<code>! found && (age >= 18)</code> <div><code>F && T</code> <code> F</code></div>	False ! found is false; age > 18 is 20 > 18 is true. Therefore, ! found && (age >= 18) is false && true, which evaluates to false.
<code>!(found && (age >= 18))</code> <div><code>! (T && T)</code> <code> ! (T)</code> <code> F</code></div>	False Now, found && (age >= 18) is true && true, which evaluates to true. Therefore, !(found &&(age >= 18)) is !true, which evaluates to false.

Bitwise Operators

- The bitwise operators are listed in the following table as:

Operator	Name	Description
&	AND	Bitwise AND
	OR	Bitwise Inclusive OR
^	XOR	Bitwise Exclusive OR
~	NOT	complement(bit inversion)
<<	SHL	Shift Left
>>	SHR	Shift Right

Let's look at the truth table of the bitwise operators.

X	Y	X & Y	X Y	X ^ Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

- **Example:** We will perform a Bitwise operation between two numbers 7 and 4. In binary 7 will be represented as 111 and 4 will be represented as 100.

1. Bitwise AND (&)

$$\begin{array}{r} 1\ 1\ 1 \\ \& \\ 1\ 0\ 0 \\ \hline 1\ 0\ 0 = 4 \end{array}$$

As we can see in the above example only those bits are set bits whose corresponding bits (both) are set. Therefore $7\&4=4$

2. Bitwise OR (|)

$$\begin{array}{r} 1 \ 1 \ 1 \\ | \\ 1 \ 0 \ 0 \\ \hline 1 \ 1 \ 1 = 7 \end{array}$$

As we can see in above example those bits are set bits whose at least any one corresponding bit are set. Therefore $7|4=7$.

3. Bitwise XOR (^)

$$\begin{array}{r} 1 \ 1 \ 1 \\ ^ \\ 1 \ 0 \ 0 \\ \hline 0 \ 1 \ 1 = 3 \end{array}$$

As we can see in above example those bits are set bits whose corresponding bits are different . Therefore $7^4=3$.

4.Bitwise NOT (~)

- The Bitwise NOT operation is performed on a single number. It change the current bit to it's complement , i.e. if current bit is 0 then in result it will be 1 and if current bit is 1 then it will become 0. It is denoted by the symbol ~ .
- **Example:** We will perform bitwise NOT Operation on number 4. The number 4 is represented as 100 in binary.

$$\begin{array}{r} \sim 1\ 0\ 0 \\ \hline 0\ 1\ 1 \end{array}$$

- As we can see in result the bits whose initial value was 1 are 0 in result and vice-versa. Therefore Bitwise NOT of number 4 will be 3.

5.Left Shift (<<)

- **Example:** Consider we have an integer 5, and we will left-shift its bits by 2 positions. The operation will be represented as $x \ll 2$.
- The number 5 is represented as 101 in binary. We will add some zeros at the beginning to left shift the bits. Therefore it will be represented as 00000101.
- we will move all the bits two positions to left and we will fill the empty positions with 0. Therefore it will become 00010100 which is 20 .
- As mentioned earlier left shifting the number by two bits means multiplying it by 2 raised to 2 which is 4 . $5 * 4 = 20$ shows the above mentioned statement.

6.Right Shift (>>)

- **Example:** Consider we have an integer 16, and we will right-shift its bits by 2 positions. The operation will be represented as $x \gg 2$.
- The number 16 is represented as 10000 in binary. We will add some zeros at the beginning to right shift the bits. Therefore it will be represented as 00010000.
- We will move all the bits two positions to right and we will fill the empty positions with 0. Therefore it will become 00000100 which is 4.
- As mentioned earlier right shifting the number by two bits means dividing it by 2 raised to 2 which is 4 . $16 \div 4 = 4$ shows the above mentioned statement.

Example 4: Write a C++ program to read two numbers and compute bitwise operators between them.

```
#include<iostream.h>
```

```
#include<conio.h>
```

```
main ( )
```

```
{
```

```
int a, b, c, d, e, f, g, h ;
```

```
cin >> a >> b ;
```

```
c = a & b ;
```

```
cout << "c=" << c << "\n" ;
```

```
d = a | b;
```

```
cout << "d=" << d << "\n" ;
```

```
e = a ^ b ;
```

```
cout << "e=" << e << "\n" ;
```

```
f = ~ a ;
```

```
cout << "f=" << f << "\n" ;
```

```
g = a << 3 ;
```

```
cout << "g=" << g << "\n" ;
```

```
h = a >> 3;
```

```
cout << "h=" << h << "\n" ;
```

```
getch ( ) ; }
```

c = a & b	d = a b	e = a ^ b	f = ~ a	g = a << 3	h = a >> 3
a=3 11	a=5 101	a=5 101	a=23	a=25	a=25
b=3 11	b=7 111	b=4 100	00010111	00011001	00011001
c= 3 11	d=7 111	e=1 001	f= 132	g=200	h=3
			11101000	11001000	00000011

Exercises

Q1: Write C++ program to find the value of B (true or false) for the following: `i= 5; j = 9; B= ! ((i > 0) && (i >= j));`

Q2: Write C++ program to illustrate the relational operators :
(a > b), (a < b), (a == b), (a != b). Assume that a=10 , b=6 .

Q3: Write C++ program to read two numbers and compute bitwise operators on them. Assume that a= 5, b= 9

جامعة بغداد / كلية التربية للعلوم الصرفة ابن الهيثم / قسم علوم الحاسبات
المرحلة الاولى / نظري
السنة الدراسية 2024-2025

Structured Programming

Lecture 7 – Selection Statements

Selection Statements

- Selection Statements Conditional expressions are mainly used for decision making.
- C++ provides multiple selection structures: if, if/else, else if, nested if, and switch

- **The Single If Statement Structure**

The if statement allows you to put some decision making into your programs. If the expression is true, the statement will be executed. If the expression is not true, the statement will not be executed.

General Form of Single –Selection If Statement

if (expression or condition)
statement 1;

Example 1:

if (avg >= 50)
cout << "good";

Example 2:

if (x > 0.0)
sum += x ;

Example 3:

cin >> num;
if (num == 0)
count = count + 1 ;

Example 4:

if (x == 100)
cout << "x is 100";

Example 5: Write a C++ program to find the average of three numbers and print “good” if the average greater than or equal to 50.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int x = 50, y = 60, z = 100, sum ;
float avg ;
sum = x + y + z ;
avg = sum / 3 ;
cout << " avg ="<< avg << "\n" ;
    if ( avg >= 50 )
        cout << "good";
getch ( ) ;
}
```

Output:

avg =70
good

The Single If Statement Structure (Blocks)

Multiple statements after the if may be grouped by putting them inside curly braces ({ })

General Form of Single –Selection If Statement (Blocks)	
if (expression or condition)	
{	
	<i>statement 1;</i>
	<i>statement 2;</i>
	<i>statement 3;</i>
	<i>statement 4;</i>
}	

Example 6:

```
if (x == 100)
{
    cout << "x is " ;
    cout << x ;
}
```

Example 7: Write a C++ program to read a number and print “the number is positive” if it positive, and decrement it by 2.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int x ;
cin >> x ;
if ( x >= 0 )
{
    cout << x << "x is a positive \n" ;
    x -= 2 ;
    cout << "x=" << x ;
}
getch ( ) ;
}
```

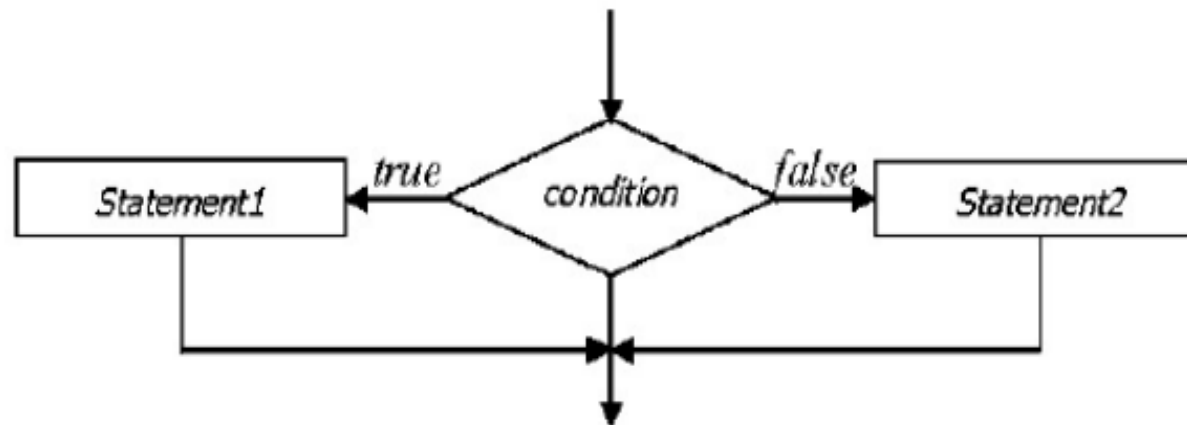
Output:

```
32
32 x is a positive
x=30
```

The If / else Statement Structure

The if-else form is used to decide between two statements referred to as:

General Form of If / else Statement
<pre>if (expression or condition) statement 1; else statement 2;</pre>



Example 8: Write a C++ program to read a number and print “the number is positive” if it positive, otherwise print negative.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int x ;
cin >> x ;
if ( x >= 0 )
    cout << "positive" ;
else
    cout << "negative" ;
getch ( ) ;
}
```

Output:

5
positive

-3
negative

Example 9: Write a C++ program to read two numbers and print the largest number.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int x, y ;
cin >> x >> y ;
if ( x > y )
    cout << x ;
else
    cout << y ;
getch ( ) ;
}
```

Output:

```
6 8
8
10 2
10
```

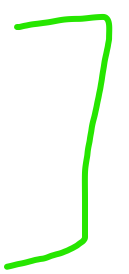

The If / else Statement Structure (Blocks)

- The if and if. .else structures control only one statement at a time. Suppose, however, that you want to execute more than one statement if the expression in an if or if. .else statement evaluates to true. To permit more complex statements, C++ provides a structure called a compound statement or a block of statements. A compound statement takes the following form:

General Form of If / else Statement (Blocks)
<pre>if (expression or condition) { <i>statement 1;</i> <i>statement 2;</i> <i>statement 3;</i> } else { <i>statement 1;</i> <i>statement 2;</i> <i>statement 3;</i> }</pre>

Example 11: Write a C++ program to read a number and print “the number is even” if it even, and decrement it by 2, otherwise print odd and increment it by 10.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int x;
cin >> x ;
if ( x % 2 == 0 )
{
cout << x << "x is even \n" ;
x -= 2 ;
cout << "x=" << x ;
}
else
{
cout << x << "x is odd \n" ;
x += 10 ;
cout << "x=" << x ;
}
getch ( ) ;
}
```



Output:

3
3 x is odd
x=13

4
4 x is even
x= 2

else If Statements

General Form of else If statement as:

General Form of else If Statement

```
if ( expression or condition 1 ) statement1 ;  
else if ( expression or condition 2 ) statement2 ;  
else if ( expression or condition 3 ) statement3 ;  
:  
else if ( expression or condition n ) statement-n ;  
else statement-e ;
```

```
— if (time < 10)  
    cout<<("Good morning.\n");  
— else if (time < 20)  
    cout<<("Good day.\n");  
— else  
    cout<<("Good evening.\n");
```

Example 13: Write a C++ program to read a number and print the day of the week.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int day ;
cin >> day ;
if ( day == 1 ) cout << "Sunday" ;
else if (day == 2 ) cout << "Monday" ;
else if (day == 3 ) cout << "Tuesday" ;
else if (day == 4 ) cout << "Wednesday" ;
else if (day == 5 ) cout << "Thursday" ;
else if (day == 6 ) cout << "Friday" ;
else if (day == 7 ) cout << "Saturday" ;
else cout << "Invalid day number" ;
    getch( ) ;
}
```

Output:

1
Sunday

4
Wednesday

8
Invalid day number

Example 14: Write a C++ program to compute the value of Z according to the following equations:

$$Z = \begin{cases} x + 5 & : x < 0 \\ \cos(x) + 4 & : x = 0 \\ \sqrt{x} & : x > 0 \end{cases}$$

```
#include<iostream.h>
# include<math.h>
# include<conio.h>
main ( )
{
int Z, x ;
cout << "Enter X value \n" ;
cin >> x ;
if ( x < 0 ) Z = x + 5 ;
else if ( x == 0 ) Z = cos(x) + 4 ;
else Z = sqrt ( x ) ;
cout << "Z is " << Z ;
    getch ( ) ;
}
```

Output:

Enter X value
25
Z is 5

Enter X value
-4
Z is 1

Enter X value
0
Z is 5

Nested If Statement

- General form of nested if statement as:

General Form of Nested If Statement
<pre>if (expression or condition 1) { if (expression or condition 2) {statements } else {statements} } else {statements}</pre>

Example 15: Write a C++ program to find a largest value among three numbers:

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int x, y, z ;
    cout << "Enter any three numbers \n" ;
    cin >> x >> y >> z ;
    if ( x > y ) {
        if ( x > z )
            cout << "the largest value is" << x << endl ;
        else
            cout << "the largest value is" << z << endl ;
    }
    else if ( y > z )
        cout << "the largest value is" << y << endl ;
    else
        cout << "the largest value is" << z << endl ;
    getch ( ) ;
}
```

Output:

Enter any three numbers

12 4 7

the largest value is **12**

Enter any three numbers

4 55 8

the largest value is **55**

Enter any three numbers

3 5 88

the largest value is **88**

Conditional Statement (?)

- Returns a certain value if that expression is true, and it returns a different value if that expression is false. This operator is an abbreviation of the if / else operator. General Form of Conditional statement

General Form of Conditional Statement ?
<i>condition ? result1 : result2</i>

If the condition is true, then the expression will return the value result1, but if it is false, it will return the value result2.

For examples:

- `7 == 5 ? 4 : 3` // return 3
- `7 == 5 + 2 ? 4 : 3` // return 4
- `5 > 3 ? a : b` // return a
- `a > b ? a : b` // return a or b

Example 10: Write a C++ program to read integer number and print if it is positive or negative.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int value ;

cin >> value ;
cout << ( value >= 0 ? "positive" : "negative" );
getch ( );
}
```

<p><u>Output:</u> 5 positive -4 negative</p>

Switch-Case Selection Statement

- Its objective is to check several possible constant values for an expression. General Form of Switch-Case Selection statement as:

General Form of Switch-Case Selection
<pre>switch (variable) { case value One: statement1; break; case value Two: statement2; break; : : case value N: statement N; break; default: statement; }</pre>

Example

```
#include <iostream.h>
#include <conio.h>
main( ) {
    int num;
    cout<<"Enter a number:";
    cin>>num;
    switch (num)
    {
        case 20:
            cout<<"It is 20";
            break;
        case 30:
            cout<<"It is 30";
            break;
        case 40:
            cout<<"It is 40";
            break;
        default:
            cout<<"Not 20, 30 or 40";
    }
    getch( ); }
```

Output

Enter a number:30
It is 30

Enter a number:20
It is 20

Enter a number:40
It is 40

Enter a number:10
Not 20, 30 or 40

Example 16: Write a C++ program to read two integer numbers, and read the operations to perform on these numbers.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int a, b ;
char x ;
cout << "Enter two numbers \n" ;
cin >> a >> b ;
cout<< " enter your choice \n" ;
cin >> x;
switch ( x )
{
case '+' : cout << ( a + b ) << endl ; break ;
case '-' : cout << ( a - b ) << endl ; break ;
case '*' : cout << ( a * b )<< endl ; break ;
case '/' : cout << ( a / b ) << endl ; break ;
default: cout << "not correct character \n" ;
}
getch ( ) ;
}
```

Output:

```
Enter two numbers
3 5
enter your choice
*
15
```

Exercises

Q1: Write a C++ program to find the value of Z.

$$Z = \begin{cases} 5x^2 + 3x/y & \text{when } x = y \\ Y^2 - 3x & \text{when } y > x \end{cases}$$

Q2: Write a C++ program to find the smallest value among three numbers using (else ... if).

Q3: Write a C++ program to enter a value less than (10) or greater than (100) and print it using (if ... else)

Q4: Write a C++ program to find the largest number between five numbers (using the if statement).

Exercises

Q5: Write a C++ program to read a number and print the day of the week. (Using Switch Statement)

Q6: Write a C++ program to read integer number and print if it is even or odd . (Using Conditional Statement (??))

جامعة بغداد / كلية التربية للعلوم الصرفة ابن الهيثم / قسم علوم الحاسبات
المرحلة الاولى / نظري
السنة الدراسية 2024-2025

Structured Programming

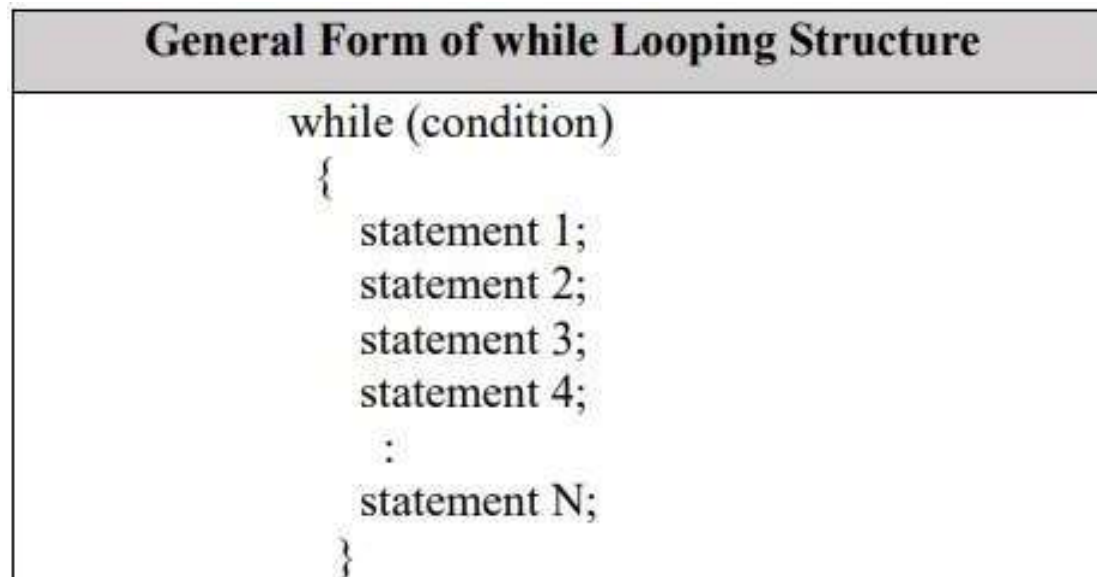
Lecture 8 – Looping Statements

Looping (Repetition) Statements

- Sometimes it is necessary to repeat a set of statements several times. If you want to repeat a set of statements 100 times, you type the set of statements 100 times in the program.
- However, this solution of repeating a set of statements is impractical, if not impossible. Fortunately, there is a better way to repeat a set of statements. C++ has three repetitions, or looping, structures that allow you to repeat a set of statements until certain conditions are met as while, do-while, and for loop.

while Looping (Repetition) Structure

- In C++, while is a reserved word. Of course, the statement can be either a simple or compound statement.
- The expression acts as a decision maker and is usually a logical expression.
- The statement is called the body of the loop. Note that the parentheses around the expression are part of the syntax.



Example 1:

```
i = 0 ;  
while ( i < 10 )  
{  
    cout << i ;  
    i ++ ;  
}
```

Output:

0 1 2 3 4 5 6 7 8 9

Example 2:

```
i = 0 ;  
while ( i < 10 )  
{  
    cout << i ;  
    i += 2 ;  
}
```

Output:

0 2 4 6 8

Example 3: Write a C++ program to read 10 numbers and print only even numbers.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int x, i ;
    i = 0 ;
    while ( i < 10 )
    {
        cin >> x ;
        if ( x % 2 == 0 )
            cout << x << " is even \n" ;
        i = i + 1 ;
    }
    getch ( ) ;
}
```

Output:

1 3 5 6 8 8 2 9 0 6

6 is even

8 is even

8 is even

2 is even

0 is even

6 is even

Example 4: Write a C++ program to find the summation of the following series.

$$\text{sum} = 1 + 3 + 5 + 7 + \dots + 99.$$

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int i, sum = 0 ;
    i = 1 ;
    while ( i <= 99 )
    {
        sum = sum + i ;
        i = i + 2 ;
    }
    cout << "sum=" << sum ;
getch ( ) ;
}
```

<p><u>Output:</u> sum=2500</p>

Example 5: Write a C++ program to find the summation of the following series.

$$\sum_{i=1}^n i^2 = 1^2 + 2^2 + 3^2 + \dots + n^2$$

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int i=1, n, sum = 0 ;
cout << "enter positive number" ;
    cin >> n ;
    while ( i <= n )
    {
        sum += i * i ;
        i ++ ;
    }
    cout << "sum is: " << sum << endl ;
    getch ( ) ; }
```

Output:

enter positive number 3
sum is:14

Example 6: Write a C++ program to read a sequence of numbers and print them, the program stops when the number equal to zero.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int x ;
cout << " Enter number" ;
cin >> x ;
while ( x != 0 )
{
cout << "x is" << "\t" << x << " \n" ;
cin >> x ;
}
getch ( ) ;
}
```

Output:

Enter number

4

x is 4

9

x is 9

7

x is 7

3

x is 3

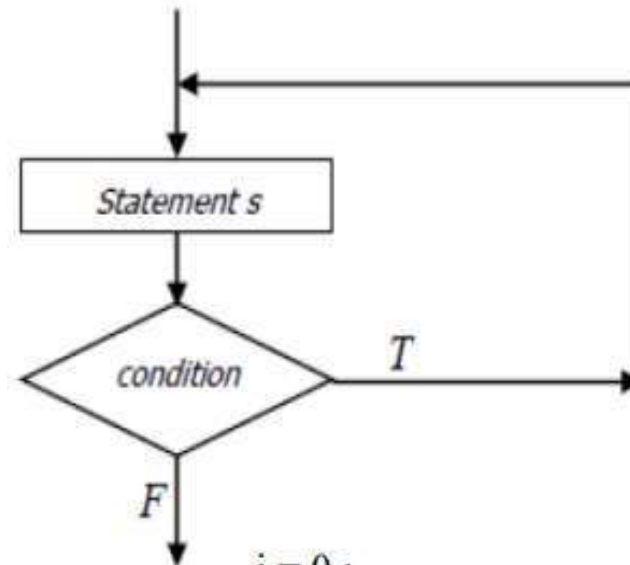
0

Do / While Looping (Repetition) Structure

- do-while loop is evaluated after the execution of statement instead of before, granting at least one execution of statement even if condition is never fulfilled

General Form of do / while Looping Structure

```
do
{
    statement 1;
    statement 2;
    statement 3;
    :
    statement N;
}
while (condition);
```



```
i = 0 ;
do
{
    cout << i ;
    i += 2 ;
}
while ( i < 10 );
```

Output:
0 2 4 6 8

Example 9: Write a C++ program to find the summation of student marks, and its average, assume the student have 8 marks.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int mark, sum = 0, i ;
float avg;
i = 1 ;
do
{
    cin >> mark ;
    sum += mark ;
    i ++ ;
}
while ( i <= 8 );
cout << "sum= " << sum << "\n " ;
avg = sum / 8 ;
cout << " the average is: " << avg ;
getch ( ) ;
}
```

Output:

```
6 7 3 1 2 2 2 1
sum= 24
the average is: 3
```

Example 10: Write a C++ program to read a list of integer numbers ended by (-100) and count the number of list and the product and the average.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int x, k, p ;
float ave, s ;
cin >> x ;
s = 0 ; k = 0 ; p = 1 ;
do
{
    s = s + x ;
    k = k + 1 ;
    p = p * x ;
    cin >> x ;
}
while ( x != -100 );
ave = s / k ;
cout << s << endl << k << endl << p << endl << ave ;
getch ( ) ;
}
```

Output:

1 2 3 4 2 -100

S=12

K=5

P=48

Ave=2.4

Example 11: Write C++ program to find the sum of positive numbers if the user enters the negative numbers the loop ends.

```
#include<iostream.h>
```

```
#include<conio.h>
```

```
main( ) {
```

```
    int number = 0;
```

```
    int sum = 0;
```

```
    do
```

```
    {
```

```
        cout << "Enter a number: ";
```

```
        cin >> number;
```

```
        sum += number;
```

```
    }
```

```
    while (number >= 0);
```

```
    cout << "The sum is " << sum << endl;
```

```
    getch ( ) ;
```

```
}
```

Output:

Enter a number: 6

Enter a number: 5

Enter a number: 9

Enter a number: 1

Enter a number: 4

Enter a number: -1

The sum is 25

Different between while and Do while loop

while	do-while
Condition is checked first then statement(s) is executed.	Statement(s) is executed atleast once, thereafter condition is checked.
It might occur statement(s) is executed zero times, If condition is false.	At least once the statement(s) is executed.
No semicolon at the end of while. while(condition)	Semicolon at the end of while. while(condition);
If there is a single statement, brackets are not required.	Brackets are always required.
<pre>while(condition) { statement(s); }</pre>	<pre>do { statement(s); } while(condition);</pre>

Different between while and Do while loop

```
// while loop example
int i = 1; // initialize a variable
while (i <= 5) { // check the
condition
    cout << i << " "; // print the
variable
    i++; // increment the variable
}
// output: 1 2 3 4 5
```

```
// do-while loop example
int j = 1; // initialize a variable
do
{
    cout << j << " "; // print the
variable
    j++; // increment the variable
}
while (j <= 5); // check the
condition
// output: 1 2 3 4 5
```

Different between while and Do while loop

```
// while loop example
int i = 6;
while (i <= 5)
{
    cout << i << " ";
    i++;
}
// Output: nothing
```

```
// do-while loop example
int i = 6;
do
{
    cout << i << " ";
    i++;
}
while (i <= 5);
// Output: 6
```

جامعة بغداد/ كلية التربية للعلوم الصرفة ابن الهيثم/قسم علوم الحاسبات
المرحلة الاولى/ نظري
السنة الدراسية 2024-2025

Structured Programming

Lecture 9 – for loop Statement

for loop Statement Structure

- The for statement allows you to execute a block of code a specified number of times.
- The for loop is designed to allow a counter variable that is initialized at the beginning of the loop and incremented (or decremented) on each iteration of the loop. The general form of the for statement is:

General Form of for loop statement Structure		
for (initialization; condition; incrementation)		
statement1;	➡	One Statement
for (initialization; condition; incrementation)		
{		
statement1;		
statement2;		
:		
statement N;	➡	Multiple Statements
}		

Example 12:

```
for ( i = 0 ; i < 10 ; i ++ )  
cout << i ;
```

Output:

0 1 2 3 4 5 6 7 8 9

Example 13:

```
for ( i = 0 ; i < 10 ; i += 2 )  
cout << i ;
```

Output:

even numbers only

0 2 4 6 8

Example 14:

```
for ( i = 1 ; i < 10 ; i += 2 )  
cout << i ;
```

Output:

odd numbers only

1 3 5 7 9

Example 15:

```
for ( int i = 1; i <= 5 ; i++ )  
cout << "Hello!" << endl ;  
cout << "*" << endl ;
```

Output:

```
Hello!  
Hello!  
Hello!  
Hello!  
Hello!  
*
```

Example 16:

```
for ( int i = 1; i <= 5 ; i++ )  
{  
cout << "Hello!" << endl ;  
cout << "*" << endl ;  
}
```

Output:

```
Hello!  
*  
Hello!  
*  
Hello!  
*  
Hello!  
*  
Hello!  
*
```

Example 17: Write a C++ program to sum the numbers between 1 and 100.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int sum = 0 ;
for ( int i = 1 ; i <= 100 ; i++ )
sum = sum + i ;
cout << "sum= " << sum ;
getch ( ) ;
}
```

Output:

sum= 5050

Example 18: Write a C++ program to find the result of the following:

$$\sum_{i=1}^{20} i^2$$

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int sum = 0 ;
for ( int i = 1 ; i <= 20 ; i++ )
sum = sum + ( i * i ) ;
cout<< "sum= " << sum ;
getch ( ) ;
}
```

Output:

sum= 2870

Example 20: Write a C++ program to read 10 integer numbers, and find the sum of these numbers and check if its positive, negative and zero.

```
#include<iostream.h>
# include<conio.h>
main ( )
{

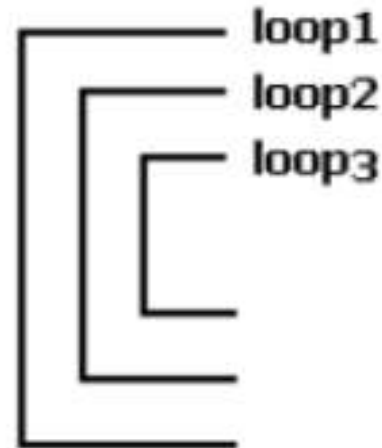
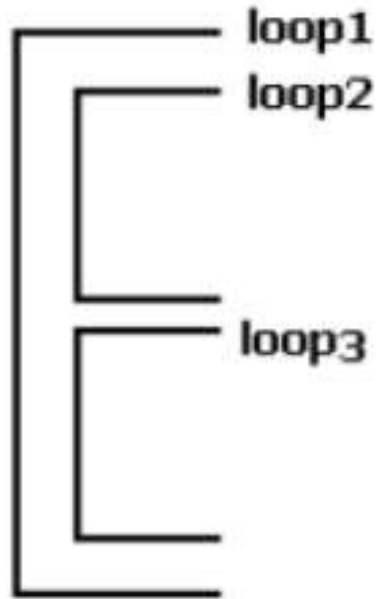
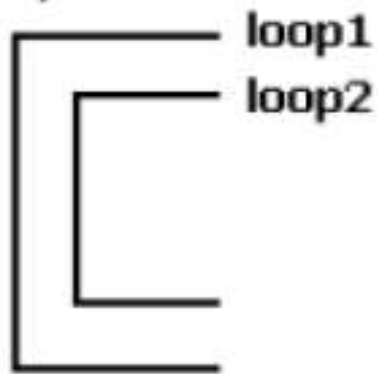
int sum = 0, x ;
for ( int i = 1 ; i <= 10 ; i++ )
{
cin >> x ;
sum = sum + x ;
if ( x > 0 )
    cout << x << "x is a positive \n" ;
else
if ( x < 0 )
    cout << x << "x is a negative \n" ;
else
    cout << x << "x is a zero \n" ;
}
cout << "sum= " << sum ;
getch ( ) ;
}
```

Output:

```
1 2 3 -4 0 5 3 -1 0 -7
1 x is a positive
2 x is a positive
3 x is a positive
-4 x is a negative
0 x is a zero
5 x is a positive
3 x is a positive
-1 x is a negative
0 x is a zero
-7 x is a negative
sum= 2
```

Nested for loop

- loops can be put one inside another to solve certain programming problems. Loops may be nested as follows:



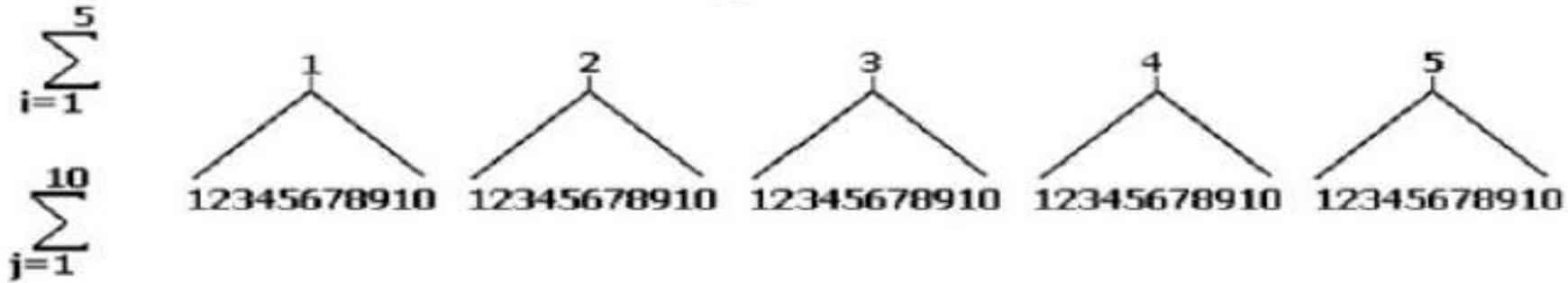
Example 21: Write a C++ program to print the following form.

```
*  
* *  
* * *  
* * * *  
* * * * *
```

```
#include<iostream.h>  
# include<conio.h>  
main ( )  
{  
for ( int i = 1; i <= 5 ; i++ )  
{  
    for (int j = 1 ; j <= i ; j++ )  
        cout << "*" ;  
    cout << endl ;  
}  
getch ( ) ;  
}
```


Example 22: Write a C++ program to find the result of the following equation.

$$\sum_{i=1}^5 \sum_{j=1}^{10} i + 2j$$



```
#include<iostream.h>
#include<conio.h>
main ( )
{
    int i, j, sum = 0;
    for ( i = 1; i <= 5 ; i++ )
        for ( j = 1 ; j <= 10 ; j++ )
            sum = sum + ( i + 2 * j ) ;
    cout << "sum is:" << sum ;
    getch ( ) ;
}
```

Output:
S=700

Example 23: Write a C++ program to find the maximum number from 10 numbers.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
    int max = 0, x ;
    for ( int i =1; i <= 10 ; i++ )
    {
        cin >> x ;
        if ( x > max )
            max = x ;
    }

    cout << "max = " << max ;
    getch ( ) ;
}
```

Output:

```
100 2 3 4 5 6 77 88 999 0
max = 999
```

Break Control Statements

- Break statement causes enclosing loop or switch to be terminated immediately. It can be used to end an infinite loop, or to force it to end before its natural end. The general form of the break statement is:

break;

Example 24:

```
for ( int i = 1 ; i < 100 ; i++ )  
{  
    cout << i ;  
    if ( i == 10 )  
        break ;  
}
```

Output:

1 2 3 4 5 6 7 8 9 10

Example 25: Write a C++ program to read 10 integer numbers and the program will be stopped when the number is negative otherwise compute the square root of the number.

```
#include<iostream.h>
# include<conio.h>
# include<math.h>
main ( )
{
int i, x ;
for ( i=1 ; i <= 10 ; i++ )
{
    cin >> x ;
    if ( x < 0 )
        break ;
    else
        cout << sqrt ( x ) << endl ;
}
getch ( ) ;
}
```

Output:

```
2
sqrt=1.41421
4
sqrt=2
9
sqrt=3
16
sqrt=4
-4
```

Example 26: Write a C++ program to read 10 integer numbers and the number of (-100) must not be counted and print the count of even number.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int i, x, m = 0 ;
for ( i=1 ; i <= 10 ; i++)
{
cin >> x;
if ( x == -100 )
continue ;
else
if ( x % 2 == 0 )
m ++ ;
}
cout << "m=" << m ;
getch ( ) ;
}
```

Output:

2 3 4 4 6 7 -100 8 9 2
m=6

Example 27: Write a C++ program which input a sequences of integer numbers and compute their sum when the value is -19 must not be counted and the program finished when the number is 9.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int x, sum = 0 ;
for ( ; ; )
{
    cin >> x ;
    if ( x == 9 )
        break ;
    if ( x == -19 )
        continue ;
    else
        sum = sum + x ;
}
cout << "sum=" << sum ;
getch ( ) ;
}
```

Output:

```
5
6
2
8
-19
8
9
sum=29
```

جامعة بغداد/ كلية التربية للعلوم الصرفة ابن الهيثم/قسم علوم الحاسبات
المرحلة الاولى/ نظري
السنة الدراسية 2024-2025

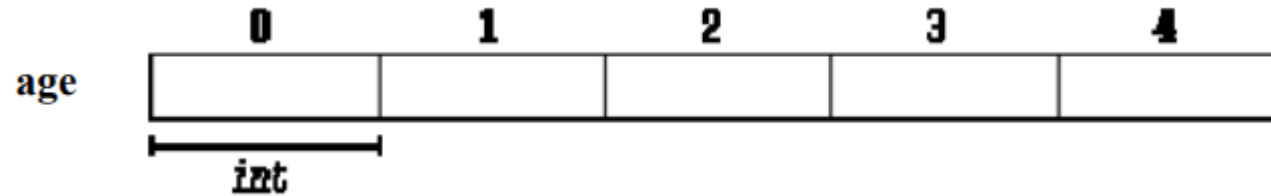
Structured Programming

Lecture 10 – Arrays

Arrays

- An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier.
- That means that, for example, we can store 5 values of type `int` in an array without having to declare 5 different variables, each one with a different identifier. Instead of that, using an array to store 5 different values of the same type, `int` for example, with a unique identifier.

For example: an array to contain 5 integer values of type `int` called `age` could be represented like this:



where each blank panel represents an element of the array, that in this case are integer values of type `int`. These elements are numbered from 0 to 4 since in arrays the first index is always 0, independently of its length. Like a regular variable, an array must be declared before it is used.

Array of One Dimensional

A typical declaration for one dimensional array (1-D) in C++ is:

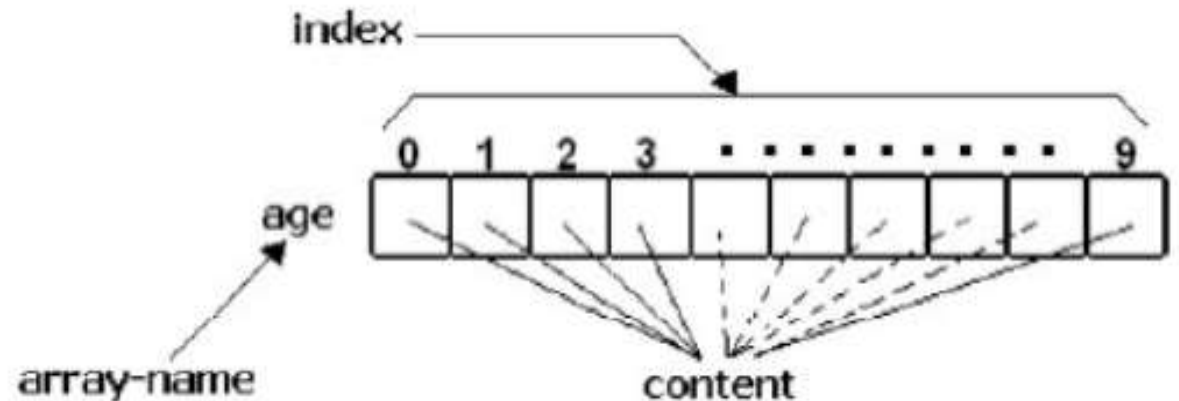
General Form of 1-D Array
Data-type Array-name [Size];

where data-type is a valid type (like int, float...), array-name is a valid identifier and the size field (which is always enclosed in square brackets []), specifies how many of these elements the array has to contain

For example:

```
int age[10];
```

```
float degree[5];
```



Initializing Array of One Dimensional

To declare an array, we have the possibility to assign initial values to each one of its elements by enclosing the values in braces { }. For examples: `int billy [5] = { 16, 2, 77, 40, 12071 };` This declaration would have created an array like this:

	0	1	2	3	4
billy	16	2	77	40	12071

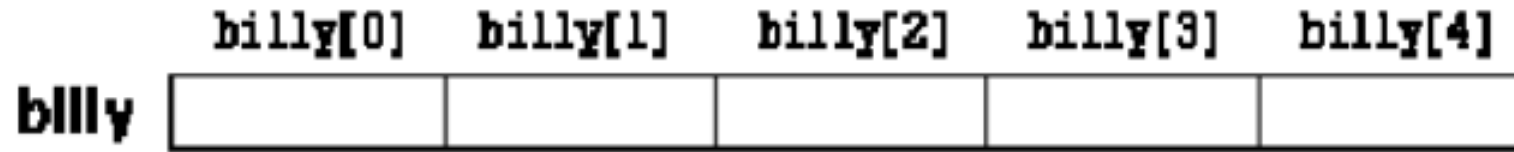
The amount of values between braces { } must not be larger than the number of elements that we declare for the array between square brackets []. For example in the example of array billy we have declared that it has 5 elements and in the list of initial values within braces { } we have specified 5 values, one for each element.

Note: When an initialization of values is provided for an array, C++ allows the possibility of leaving the square brackets empty []. In this case, the compiler will assume a size for the array that matches the number of values included between braces { }

For example: `int billy [] = {16, 2, 77, 40, 12071};`

Accessing the Values of an Array

Access to each array element is achieved by written name of array, followed by brackets delimiting a variable in the brackets which are called the array index. Following the previous examples in which billy had 5 elements and each of those elements was of type int, the name which we can use to refer to each element is the following:



- For example, to store the value 75 in the third element of billy, we could write the following statement:

```
billy[2] = 75
```

and, for example, to pass the value of the third element of billy to a variable called a, we could write:

```
a = billy[2];
```

```
int billy[5]; // declaration of a new array
```

```
billy[2] = 75; // access to an element of the array.
```

Read and Write Array Elements

- `cin>>num[i];` \\ read elements
- `cout<<num[i];` \\ print elements

Example 1: Write a C++ program to initialize array num to read 10 integers and print them.

Solution 1

```
#include <iostream.h>
# include <conio.h>
main ( )
{
int num [10] ={1, 4, 8, 66, 99, 66, 22, 54, 87, 77 } ;
for ( int i = 0 ; i < 10 ; i++ )
cout << num [i] << "\\n" ; getch ( ) ;
}
```

Output:

```
1
4
8
66
99
66
22
54
87
77
```

Solution 2

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int num [10] ;
for ( int i = 0 ; i < 10 ; i++ )
cin >> num [i] ;
for ( int i = 0 ; i < 10 ; i++ )
cout << num [i] << "\n" ;
getch ( ) ;
}
```

Output:

```
2 3 5 6 7 33 66 55 88 99
2
3
5
6
7
33
66
55
88
99
```

Array of Two Dimensional

- Two dimensional array are often used to represent tables of values consisting of information arranged in rows and columns. To identify a particular table element, we must specify two subscripts.
- By convention, the first identifies the element's row and the second identifies the element's column.
- Arrays that require two subscripts to identify a particular element are called two-dimensional arrays or 2-D arrays.
- Figure below illustrates a two-dimensional array, a. The array contains three rows and four columns, so it is said to be a 3-by-4 array. In general, an array with m rows and n columns is called an m -by- n array.

Two-dimensional array with three rows and four columns.

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

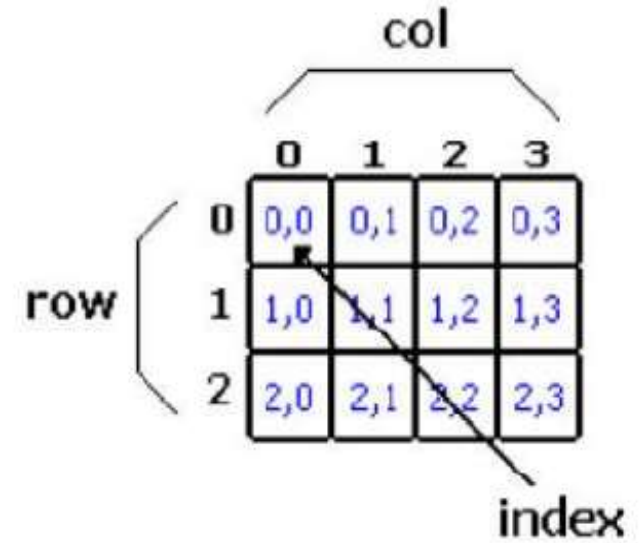
Diagram illustrating the structure of a 2D array with three rows and four columns. The array is represented as a grid of elements. The first row is labeled 'Row 0' and the first column is labeled 'Column 0'. The elements are shown as `a[row][column]`. Arrows point from the labels to the corresponding parts of the array notation:

- Column subscript: Points to the second bracketed value (column index).
- Row subscript: Points to the first bracketed value (row index).
- Array name: Points to the variable name 'a'.

A typical declaration for two dimensional array (2-D) in C++ is:

General Form of 2-D Array
Data-type Array-name [Row-Size][Column-Size];

Example: `int a [10] [10];`
 `int num [3] [4];`



Initializing Array of Two Dimensiona

Example:

`a [2] [3] = { {1, 2, 3} , {4, 5, 6} };`

1	2	3
4	5	6

Read and Write Array Elements

Example 3: Write a C++ program to read 6 numbers, 3 numbers per row, and print them.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int a [ 2 ] [ 3 ];
int i , j ;
for ( i = 0 ; i < 2 ; i++ )
    for ( j = 0 ; j < 3 ; j++ )
        cin >> a [ i ] [ j ];
    cout << "+++++++\n" ;
for ( i = 0 ; i < 2 ; i++ )
{
for ( j = 0 ; j < 3 ; j++ )
cout << a [ i ] [ j ];
cout << endl ;
}
getch ( ) ;
}
```

Output:

1 4 5 6 7 8

+++++++

145

678

Example 4: Write a C++ program to read 3*3 2D-array, then find the summation of the array elements, finally print these elements.

```
#include<iostream.h>
# include<conio.h>
main ( )
{
int a [ 3 ] [ 3 ] ;
int i, j, sum = 0 ;
for ( i = 0 ; i < 3 ; i++ )
    for ( j = 0 ; j < 3 ; j++ )
        cin >> a [ i ] [ j ] ;
for ( i = 0 ; i < 3 ; i++ )
    for ( j = 0 ; j < 3 ; j++ )
        sum += a [ i ] [ j ] ;
cout << "sum is: " << sum << endl ;
for ( i = 0 ; i < 3 ; i++ )
{
    for ( j = 0 ; j < 3 ; j++ )
        cout << a [ i ] [ j ] ;
    cout << endl ;
}
getch ( ) ;
}
```

Output:

```
1
2
3
4
5
6
7
8
9
sum is: 45
123
456
789
```

Exercises

- Q1: Write C++ program to find the maximum value in array of 8 numbers. Q2: Write C++ program to split the odd numbers and even numbers of one array contains 20 numbers into two arrays.
- Q3: Write C++ program to read two arrays one dimensional a[5],b[5] and print the sum of elements of two arrays in array c.
- Q4: Write C++ program to read one dimensional array 1-D a[10] and compute the following: 1- count the numbers divisible by 5. 2- multiply the even number by 2 and multiply the odd number by 3 and print array. 3- print the sum of first number and last number.
- Q5: Write a C++ program to read 3*4 2D-array, then find the summation of each row.

Exercises

- Q6: Write a C++ program to read 3*4 2D-array, then replace each value equal 5 with 0.
- Q7: Write a C++ program to read 4*4 2D-array, and replace each element in the main diagonal with one.
- Q8: Write a C++ Program to read array two dimensional a(3×3) and find the following:
 - 1. find the sum of element in the last row.
 - 2. replace each element of the third column by value 10.
 - 3. replace each element of the first row by the value 5.

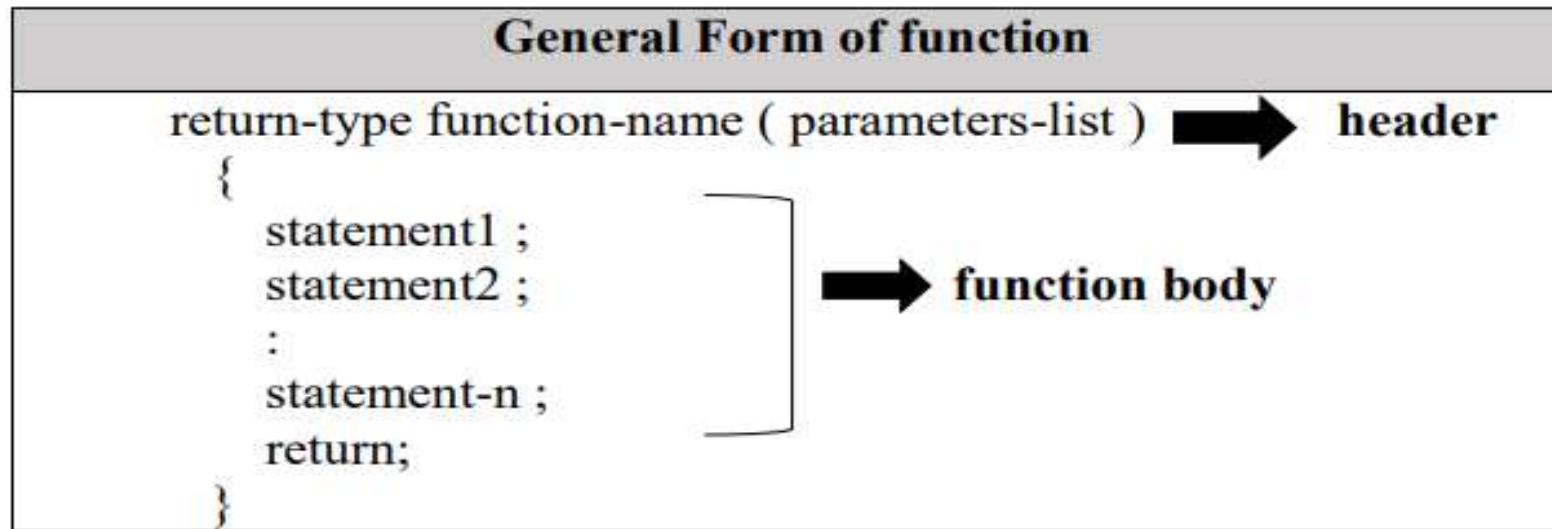
جامعة بغداد/ كلية التربية للعلوم الصرفة ابن الهيثم/قسم علوم الحاسبات
المرحلة الاولى/ نظري
السنة الدراسية 2024-2025

Structured Programming

Lecture 11– Functions

Functions

- A function is a set of statements designed to accomplish a particular task. Experience has shown that the best way to develop and maintain a large program is to construct it from smaller pieces or (modules).
- Modules in C++ are called functions. A function definition has a name, parentheses pair containing zero or more parameters and a body.
- For each parameter, there should be a corresponding is taken to be an integer by default. The general form of the function definition is:



- The type of the function may be int, float char, etc. it may be declared as type (void), which informs the compiler not to the calling program.

void function_name (...)

int function_name (...)

- Any variable declared in the body of function is said to be local to that function. Other variables which are not declared either as arguments or in the function body are considered “global” to the function and must be defined externally. For example
- `void square (int a, int b)` \\ a,b are the formal parameters
- `float output (void)` \\ function without formal parameters
- The keyword `return` is used to terminate function and return a value to its caller. The return statement may or may not include an expression. Its general syntax is:
 - `return;`
 - `return (expression) ;`
- The return statements terminate the execution of the function and pass the control back to the calling environment.

Function Definition

- Definition before main function
- The functions are defined before the first appearance of calls to them in the source code. These calls were generally in function main which we have always left at the end of the source code.
- **Example 1** : Write a C++ program using function to print any text.

```
#include<iostream.h>
# include<conio.h>
void print ( )
{
cout << "Computer Science" ;
}
main ( )
{
print ( ) ; // call for function
getch ( ) ;
}
```

<p><u>Output:</u> Computer Science</p>

Example 2 : Write a C++ program to read two numbers and return the maximum number between them using function.

```
#include<iostream.h>
# include<conio.h>
int max ( int a , int b )
{
int c ;
if ( a > b ) c = a ;
else c = b ;
return (c) ;
}
main ( )
{
cout << max (5, 6) ; // call for function, 5 and 6 are actual Arguments
getch ( ) ;
}
```

Output:

6

Example 3 : Write a C++ program to read two numbers and return the summation of these numbers using function.

```
#include<iostream.h>
# include<conio.h>
int sum ( int a , int b )
{
int r ;
r = a + b ;
return ( r ) ;
}
main ( )
{
int x, y, z ;
cin >> x >> y ;
z = sum ( x , y ) ;    // call for function
cout << "The result is " << z ;
getch ( ) ;
}
```

Output:

99

10

The result is 109

Note:

```
int sum ( int a , int b )  
      ↑ 99 ↑ 10  
z = sum (x , y);
```

At the point at which the function is called from within main, the control is lost by main and passed to function sum. The value of both arguments passed in the call (99 and 10) are copied to the local variables int a and int b within the function. Function sum declares another local variable (int r), and by means of the expression $r=a+b$, it assigns to r the result of a plus b. Because the actual parameters passed for a and b are 99 and 10 respectively, the result is 109. The following line of code: `return (r);`

```
int sum (int a, int b)  
      ↓ 109  
z = sum (x , y);
```

Definition after main function (prototype)

- There is an alternative way to avoid writing the whole code of a function before it can be used in main function. This can be achieved by declaring just a prototype of the function before it is used, instead of the entire definition.
- This declaration is shorter than the entire definition, but significant enough for the compiler to determine its return type and the types of its parameters.
- type name (type1 argument1, type2 argument2, ...);
- It is identical to a function definition, except that it does not include the body of the function itself (i.e., the function statements that in normal definitions are enclosed in braces { }) and instead of that we end the prototype declaration with a semicolon (;).

Example 4 : Write a C++ program using function prototype to calculate the average of two numbers entered by the user in the main program.

```
#include<iostream.h>
# include<conio.h>
float average ( int a , int b ) ; // prototype
main ( )
{
float z ;
int x, y ;
cin >> x >> y ;
z = average ( x , y ) ;    // call for function
cout << "The result is " << z ;
getch ( ) ;
}

float average ( int a , int b )
{
float c ;
c = ( a + b ) / 2 ;
return ( c ) ;
}
```

// **function definition**

Output:

10 6

The result is 8

جامعة بغداد/ كلية التربية للعلوم الصرفة ابن الهيثم/قسم علوم الحاسبات
المرحلة الاولى/ نظري
السنة الدراسية 2024-2025

Structured Programming

Lecture 12– Passing Parameters

Local and Global Variables

Variables can be classified into local or global variables as:

1. Local variables are declared within a function or any other inner block and cannot be used outside of them.
2. Global variables are declared outside the main function block.

For example:

```
#include<iostream.h>
```

```
# include<conio.h>
```

```
int a ;
```

```
float m, n ;
```

```
Char s ;
```

Global Variables

```
main ()
```

```
{
```

```
int num, age ;
```

```
float n2, n3 ;
```

Local Variables

```
cout << "enter your age" ;
```

```
cin >> age ;
```

Instructions

```
getch () ;
```

```
}
```

Example 9: Write a C++ program using function to subtract two both passed parameters and to return the result.

```
#include<iostream.h>
# include<conio.h>
int subtraction ( int a , int b )
{
int r ;
r = a - b ;

return ( r ) ;
}
main ()
{
int x = 5, y = 3 , z ;
z = subtraction ( 7 , 2 ) ;
cout << "The first result is " << z << "\n" ;
cout << "The second result is " << subtraction (7,2) << "\n" ;
cout << "The third result is " << subtraction (x,y) << "\n" ;
z = 4 + subtraction ( x , y ) ;
cout << "The fourth result is " << z << "\n" ;
getch () ;
}
```

Output:

The first result is 5
The second result is 5
The third result is 2
The fourth result is 6

Passing Parameters

- There are two main methods for passing parameters to a program:
- (1) passing by value, and (2) passing by reference.
- 1. Passing by Value When parameters are passed by value, a copy of the parameters value is taken from the calling function and passed to the called function. The original variables inside the calling function, regardless of changes made by the function to it are parameters will not change. All the pervious examples used this method.

Passing Parameters

- 2. Passing by Reference When parameters are passed by reference their addresses are copied to the corresponding arguments in the called function, instead of copying their values. Thus pointers & are usually used in function parameters and any modification that do to the local variables will have an effect in their variables passed as arguments in the call to the function.
- This method is more efficient and provides higher execution speed than the call by value method, but call by value is more direct and easy to use

Example 1 : Write a C++ program using function and passing by reference to read three integer numbers and multiplied by 2.

```
#include<iostream.h>
# include<conio.h>
void multiplied ( int & a , int & b , int & c )
{
a *= 2 ;
b *= 2 ;
c *= 2 ;
}
main ( )
{
int x = 1, y = 3, z = 7 ;
multiplied ( x , y , z ) ;
cout << "x=" << x << ", y=" << y << ", z=" << z ;
getch ( ) ;
}
```

Output:

x=2, y=6, z=14

Note: Passing by reference is also an effective way to allow a function to return more than one value. For example, here

- **Example 2 :** Write a C++ program using function and passing by reference to returns the previous and next numbers of the first parameter passed.

```
#include<iostream.h>
# include<conio.h>
void prevnext ( int x , int & prev , int & next )
{
    prev = x - 1 ;
    next = x + 1 ;
}
main ( )
{
    int x1 = 100, y, z ;
    prevnext ( x1 , y , z ) ;
    cout << "Previous=" << y << ", Next=" << z ;
    getch ( ) ;
}
```

Output:

Previous=99, Next=101

Arrays as parameters for Function

- At some moment may need to pass an array to a function as a parameter. In order to accept arrays as parameters the only thing that have to do when declaring the function is to specify in its parameters **the element type of the array, an identifier and a pair of void brackets []** For example, the following function:
 - void procedure (int arg [])
 - accepts a parameter of type "array of int" called arg. In order to pass to this function an array declared as:
 - int myarray [40] ;

Example 3 : Write a C++ program to read array 1-D contains 10 elements and print the array elements using function print.

```
#include<iostream.h>
# include<conio.h>
void print ( int arg [ 10 ]) //array with size
{
for ( int n = 0 ; n < 10 ; n++ )
cout << arg [ n ] << " ";
}
main ( )
{
int arr [10];
for ( int i = 0 ; i < 10 ; i++ )
cin >> arr [ i ];
print ( arr );
getch ( ) ;
}
```

Output:

```
1
2
3
4
5
6
7
8
9
0
1 2 3 4 5 6 7 8 9 0
```


Note: Also function parameters can be defined as array without size as:

```
void print ( int arg [ ] )  
{  
    for ( int n = 0 ; n < 10 ; n++ )  
        cout << arg [ n ] << " ";  
}
```

Example 4 : Write a C++ program using functions to read, print, and find the summation of array elements a [10].

```
#include<iostream.h>  
#include<conio.h>  
int a [10], s = 0, i ;  
void read ( )  
{  
    for ( i = 0 ; i < 10 ; i++ )  
        cin >> a [ i ] ;  
}  
void print ( )  
{
```

```
    for ( i = 0 ; i < 10 ; i++ )  
        cout << a [ i ] << " ";  
}  
void sum ( )  
{  
    for ( i = 0 ; i < 10 ; i++ )  
        s = s + a [ i ] ;  
    cout << "\n" << s ;  
}  
main ( )  
{  
    clrscr ( ) ; // clean screen  
    read ( ) ;  
    print ( ) ;  
    sum ( ) ;  
    getch ( ) ;  
}
```

Output:

```
1  
2  
4  
5  
4  
6  
7  
8  
9  
1  
1 2 4 5 4 6 7 8 9 1  
47
```

هذا البرنامج نفس السؤال السابق لكن ال function تستلم برامتر من نوع مصفوفة ذات بعد واحد فيكون حل السؤال بالشكل التالي:

```
#include<iostream.h>
#include<conio.h>
void read ( int a [ 10 ] )
{ int i ;
  for ( i = 0 ; i < 10 ; i++ )
    cin >> a [ i ] ;
}
void print ( int a [ 10 ] )
{ int i ;
  for ( i = 0 ; i < 10 ; i++ )
    cout << a [ i ] ;
}
void sum ( int a [ 10 ] )
{ int i, s = 0 ;
  for ( i = 0 ; i < 10 ; i++ )
    s = s + a [ i ] ;
  cout << "\n" << s ;
}
main ( )
{
  clrscr ( ) ;
  int b [ 10 ] ;
  read ( b ) ;
  print ( b ) ;
  sum ( b ) ;
  getch ( ) ;
}
```

تعريف الدالة بكتابة حجم المصفوفة

وايضا يمكن تعريف الدالة بدون كتابة حجم المصفوفة

استدعاء الدالة بكتابة اسم المصفوفة فقط بدون ابعادها

Output:

```
1
2
3
2
5
4
1
6
4
3
1 2 3 2 5 4 1 6 4 3
31
```

Example 5 : Write a C++ program to find the following.

- 1) to read array (3×3) by using function.
- 2) to print array (3×3) by using function.
- 3) find the sum of elements on triangle upper main diagonal by using function.
- 4) find the sum of elements on secondary diagonal using function

```
#include<iostream.h>
#include<conio.h>
void read ( int a [3] [3] );
int print ( int a [3] [3] );
int sumuper ( int a [3] [3] );
int sumse ( int a [3] [3] );
main ( )
{ int b [3] [3] ;
read ( b );
print ( b );
sumuper ( b );
sumse ( b );
getch ( ) ;

}
void read ( int a [3] [3] )
{ int i, j ;
```

Output:

1 2 3 4 5 6 7 8 9

1 2 3

4 5 6

7 8 9

11

15

```

for ( i = 0 ; i < 3 ; i++ )
for ( j = 0 ; j < 3 ; j++ )
cin >> a [ i ] [ j ] ;
cout << "\n" ;
}
int print ( int a [3] [3] )
{ int i, j ;
for ( i = 0 ; i < 3 ; i++ )
{
for ( j = 0 ; j < 3 ; j++ )
cout << a [i] [j] << " " ;
cout << "\n" ;
}
}

```

```

int sumuper ( int a [3] [3] )
{
int s = 0, i, j ;
for ( i = 0 ; i < 3 ; i++ )
{
for ( j = 0 ; j < 3 ; j++ )
if ( i < j )
s = s + a [ i ] [ j ] ;
}
cout << s << "\n" ;
}
int sumse ( int a [3] [3] )
{ int s1 = 0, i, j ;
for ( i = 0 ; i < 3 ; i++ )
{
for ( j = 0 ; j < 3 ; j++ )
if ( i + j == 2 )
s1 = s1 + a [ i ] [ j ] ;
}
cout << s1 << "\n" ;
}

```