Dr. Neamah E. Kadhim

2.3 more examples for class Lecture Six

EX:C++ program to define the class of an employee.

```
#include <iostream>
using namespace std;
class employee
{       private:
        string name;
        int hr;
public:
        void setname(string);
        void setTime(int ,int ,int ,int );
        void getenfo(string&,int&);
        double salary();};
void employee::setname(string s)
 {      name=s; }
void employee::setTime(int f,int s,int t, int fo)
{       hr=f+s+t+fo;}
void employee::getenfo(string& s,int& t)
{       s=name;
        t=hr;}
double employee::salary()
{       return hr*15;}
int main()
```

```
{       employee emp1;
        string n;
        int x,y,z,w;
        emp1.setname("ahmad");
        emp1.setTime(4,5,6,8);
        emp1.getenfo(n,x);
        cout<<"name is: "<<n;
        cout<<"\nworking hours are: "<<x;
        cout<<"\nsalary is: "<<emp1.salary()<<endl;
        return 2;}
```

```
name is: ahmad
working hours are: 23
salary is: 345$
Press any key to continue . . .
```

EX:C++ program to define the class of an student.

```
#include <iostream>
#include<string>
using namespace std;
class student
{private:
        string f_name;
        string s_name;
```

```
        string i_d;
        int age;
        int birth;
        int exam[3];
public:
        void setname(string,string);
        void setbirth(int);
        void setexam(int,int,int);
        void makeid();
        void comage();
        double avg();
        void print();};
void student::setname(string s,string s2)
{       f_name=s;
        s_name=s2;}
void student::setbirth(int b)
{       birth=b;}
void student::setexam(int e1,int e2,int e3)
{       exam[0]=e1;
        exam[1]=e2;
        exam[2]=e3;}

void student::makeid()
```

```
{       i_d ="id";
         i_d[0]=f_name[1];
         i_d[1]=s_name[2];}
void student::comage()
{        age=2023-birth;}
double student::avg()
{   double d;
   d=(exam[0]+exam[1]+exam[2])/3;
   return d;}
void student::print()
{       cout<<" "<<f_name<<" "<<s_name;
         makeid();
         cout<<"\nid is: "<<i_d;
         comage();
         cout<<"\nage is: "<<age;
         cout<<"\nthe average of exam is: "<<avg()<<endl;}
int main()
{
         student st1;
         int ex1,ex2,ex3,b;
         string n1,n2;
         cout<<"enter first and second name";
         cin>>n1>>n2;
```

```
cout<<endl<<"enter three scores: ";
cin>>ex1>>ex2>>ex3;
cout<<endl<<"enter birth date: ";
cin>>b;
cout<<"****************************"<<endl;
st1.setname(n1,n2);
st1.setbirth(b);
st1.setexam(ex1,ex2,ex3);
st1.print();
return 2;}
```

```
enter first and second name amal ali
enter three scores: 30 40 50
enter birth date: 2005
****************************
 amal ali
id is: mi
age is: 18
the average of exam is: 40
Press any key to continue . . .
```

EX: c++ program to define a class to implement the time of day. Because a clock gives the time of day, let us call this class clockType . Furthermore, to represent time in computer memory, we use three int variables: one to represent the hours, one to represent the minutes, and one to represent the seconds. also want to perform the following operations on the time:

1. Set the time.

2. Retrieve the time.

3. Print the time.

4. Increment the time by one second.

5. Increment the time by one minute.

6. Increment the time by one hour.

7. Compare the two times for equality.

```
#include <iostream>
using namespace std;
class clockType
{public:
        void setTime(int, int, int);
        void getTime(int&, int&, int&) ;
        void printTime();
        void incrementSeconds();
        void incrementMinutes();
        void incrementHours();
        bool equalTime(const clockType&) ;
private:
        int hr;
        int min;
        int sec;};
int main()
{clockType myClock;
clockType yourClock;
int hours;
int minutes;
int seconds;
//Set the time of myClock
```

```
myClock.setTime(5, 4, 30); //Line 1
cout << "Line 2: myClock: "; //Line 2
myClock.printTime(); //print the time of myClock Line 3
cout << endl; //Line 4
cout << "Line 5: yourClock: "; //Line 5
yourClock.printTime(); //print the time of yourClock Line 6
cout << endl; //Line 7
//Set the time of yourClock
yourClock.setTime(5, 45, 16); //Line 8
cout << "Line 9: After setting, yourClock: "; //Line 9
yourClock.printTime(); //print the time of yourClock Line 10
cout << endl; //Line 11
//Compare myClock and yourClock
if (myClock.equalTime(yourClock)) //Line 12
cout << "Line 13: Both times are equal."
<< endl; //Line 13
else //Line 14
cout << "Line 15: The two times are not equal."
<< endl; //Line 15
cout << "Line 16: Enter the hours, minutes, and "
<< "seconds: "; //Line 16
cin >> hours >> minutes >> seconds; //Line 17
cout << endl; //Line 18
//Set the time of myClock using the value of the
```

```
//variables hours, minutes, and seconds
myClock.setTime(hours, minutes, seconds); //Line 19
cout << "Line 20: New myClock: "; //Line 20
myClock.printTime(); //print the time of myClock Line 21
cout << endl; //Line 22
//Increment the time of myClock by one second
myClock.incrementSeconds(); //Line 23
cout << "Line 24: After incrementing myClock by "
<< "one second, myClock: "; //Line 24
myClock.printTime(); //print the time of myClock Line 25
cout << endl; //Line 26
//Retrieve the hours, minutes, and seconds of the
//object myClock
myClock.getTime(hours, minutes, seconds); //Line 27
//Output the value of hours, minutes, and seconds
cout << "Line 28: hours = " << hours
<< ", minutes = " << minutes
<< ", seconds = " << seconds << endl; //Line 28
return 0;
}//end main
void clockType::setTime(int hours, int minutes, int seconds)
{       if (0 <= hours && hours < 24)
          hr = hours;
          else
```

```
        hr = 0;
        if (0 <= minutes && minutes < 60)
        min = minutes;
        else
        min = 0;
        if (0 <= seconds && seconds < 60)
        sec = seconds;
        else
        sec = 0;}
void clockType::getTime(int& hours, int& minutes, int& seconds)
{       hours = hr;
        minutes = min;
        seconds = sec;}
void clockType::printTime()
{       if (hr < 10)
        cout << "0";
        cout << hr << ":";
        if (min < 10)
        cout << "0";
        cout << min << ":";
        if (sec < 10)
        cout << "0";
        cout << sec;}
void clockType::incrementHours()
```

```
{       hr++;
         if (hr > 23)
         hr = 0;}
void clockType::incrementMinutes()
{       min++;
         if (min > 59)
         {
         min = 0;
         incrementHours(); //increment hours
         }}



void clockType::incrementSeconds()
{
         sec++;
         if (sec > 59)
         {
         sec = 0;
         incrementMinutes(); //increment minutes}}
bool clockType::equalTime(const clockType& otherClock)
{return (hr == otherClock.hr
         && min == otherClock.min
         && sec == otherClock.sec);}
```

Dr. Neamah E. Kadhim

2.3 constructors lecture seven

it is desirable to initialize all data members for a given instance. More so, it is crucial to ensure that data members for any instance have notified values, as we know that memory is not provided clean or zeroed-out by C++. Initializing each data member individually each time a class is instantiated can be tedious work. A **constructor** is a member function that is automatically invoked after the memory for an instance is made available. Constructors are used to initialize the data members that comprise a newly instantiated object. To guarantee that the member variables of a class are initialized, you use constructors. There are two types of constructors: with parameters and without parameters. The constructor without parameters is called the default constructor.

Constructors have the following properties:

• The **name** of a constructor is the same as the name of the class.

• A constructor, even though it is a function, **has no type**. That is, it is neither a value-returning function nor a void function.

• A class can have **more than one constructor**. However, all constructors of a class have the same name.

• If a class has more than one constructor, the constructors must have **different formal parameter lists**. That is, either they have a different number of formal parameters or, if the number of formal parameters is the same, then the data type of the formal parameters, in the order you list, must differ in at least one position.

• Constructors **execute automatically** when a class object enters its scope. Because they have no types, they cannot be called like other functions.

• Which constructor executes depends on **the types of values passed to the class object** when the class object is declared.

EX: c++ program illustrate two different constructors.

```
#include <iostream>
#include <string>
using namespace std;
class University
{
private:
string name;
int numStudents;
public:
// constructor prototypes
University(); // default constructor
University(string, int);
void Print();
};
University::University()
{
name = "";
numStudents = 0;
}
University::University(string n, int num)
{
```

```
name=n;
numStudents = num;
}
void University::Print()
{
cout << "University: " << name;
cout << " Enrollment: " << numStudents << endl;
}
int main()
{
University u1; // Implicit call to default constructor
University u2("University of Delaware", 23800);
u1.Print();
u2.Print();
return 0;
}
```

EX: c++ program illustrate two different constructurs.

```
University:  Enrollment: 0
University: University of Delaware Enrollment: 23800
Press any key to continue . . .
```

```
#include <iostream>
#include <string>
using namespace std;
class department
name=s;
numofs=i1;
numofstu=i2;
numoftech=i3;
```

```
{
private:
string name;
int numofs;
int numofstu;
int numoftech;
int numofemp;
int numoflechall;
int numboflabs;
public:
// constructor prototypes
Department();
department(string,
int,int,int,int,int,int);
department(string,int);
department(string,int,int);
department(string,int,int,int);
void Print();
};
department::department()
{
name="";
numofs=4;
numofstu=0;
numoftech=0;
numofemp=0;
numoflechall=0;
numboflabs=0;
}
department::department(stri
ng s, int i1,int i2,int i3,int i4,int
i5,int i6){
numofemp=i4;
numoflechall=i5;
numboflabs=i6;
}
department::department(string s, int i1)
{
name=s;
numofs=i1;}
department::department(string s, int i1,int i2)
{
name=s;
numoftech=i1;
numofemp=i2;}
department::department(string s, int i1,int i2,int
i3)
{
name=s;
numofs=i1;
numoflechall=i2;
numboflabs=i3;}
void department::Print()
{
cout << "Department: " << name;
cout << " Enrollment: " << numofstu<< endl;
cout<<"stages: "<<numofs<<endl;
cout<<"techers: "<<numoftech<<endl;
cout<<"employees: "<<numofemp<<endl;
cout<<"lectures hall: "<<numoflechall<<endl;
cout<<"labs: "<<numboflabs<<endl;
cout<<"*****************************"<<e
ndl;
}
int main()
{
department d1; // Implicit call to default constructor
department d2("computer science", 4,200,20,5,7,10);
```

```
department d3("biology science", 4);
department d4("mathematics science",40,10);
department d5("chemical science",4,10,20);
d1.Print();
d2.Print();
d3.Print();
d4.Print();
d5.Print();
```

```
return 0;
}
```

```
Department:  Enrollment: 0
stages: 4
techers: 0
employees: 0
lectures hall: 0
labs: 0
*****************************
Department: computer science Enrollment: 200
stages: 4
techers: 20
employees: 5
lectures hall: 7
labs: 10
*****************************
Department: biology science Enrollment: 1984070207
stages: 4
techers: 15535208
employees: 15537756
lectures hall: 1984137077
labs: 1984467740
*****************************
Department: mathematics science Enrollment: 6749872
stages: 1
techers: 40
employees: 10
lectures hall: 15535208
labs: 6749876
*****************************
Department: chemical science Enrollment: 15535200
stages: 4
techers: -782334202
employees: 15541244
lectures hall: 10
labs: 20
*****************************
Press any key to continue . . .
```

A constructor can also have default parameters. In such cases, the rules for declaring formal

parameters are the same as those for declaring default formal parameters in a function.

```
#include <iostream>
#include <string>
using namespace std;
class car
{private:
        string name;
        string size;
        int cost;
        string number;
public:
        // constructor prototypes
        car(string="obama", string="mid-size",int=90000,string="000AA");
        car(int,string);
        void Print();};
car::car(string s, string s2,int i,string s3)
        {name=s;
        size=s2;
```

```
Type: obama Size: mid-size
Price: 90000
Number: 000AA
*****************************
Type:  Size:
Price: 50000
Number: aa1234
*****************************
Type: Toyota Size: small
Price: 40000
Number: a789
*****************************
Press any key to continue . . .
```

```
        cost=i;

        number=s3;}

        car::car(int i,string s)

        {cost=i;

        number=s;}

void car::Print()

        {cout << "Type: " << name;

        cout << " Size: " << size<< endl;

        cout<<"Price: "<<cost<<endl;

        cout<<"Number: "<<number<<endl;

        cout<<"*****************************"<<endl;}

int main()

{car c2;

car c3(50000,"aa1234");

car c4("Toyota","small",40000,"a789");

c2.Print();

c3.Print();

c4.Print();

return 0;}
```

2.4 destructors

A **destructor** is a function that is called automatically each time an object is destroyed. An object is destroyed when it goes out of scope. For example, variables are destroyed at the end of the block in which they are declared. This means a local variable always is destroyed by the end of the function in which it is declared.

The rules for creating destructor prototypes are similar to the rules for constructor prototypes:

- As with constructors, you must give a destructor the same name as its class (and therefore the same name as any constructor for that class). Unlike constructors, you must precede the destructor name with a tilde (~).
- As with constructors, you cannot give a destructor a return type (it's not necessary because destructors never return anything).
- Unlike constructors, you *cannot* pass any values to a destructor.

EX: c++ program illustrate that how use destructor.

```
#include<iostream>
using namespace std;
class House
{
private:
int squareFeet;
public:
House();
~House();
```

```
int getSquareFeet();};
House::House()
{
squareFeet = 1000;
cout << "House created." << endl;
}
House::~House()
{
cout << "House destroyed!" << endl;
}
int House::getSquareFeet()
{
return squareFeet;
}
int main()
{
House aHouse;
cout << "Square feet in house object = " <<
aHouse.getSquareFeet() << endl;
return 0;
}
```

```
House created.
Square feet in house object = 1000
House destroyed!
Press any key to continue . . .
```