Minutes : 450
Hours : 6
Minutes : 590

Program 2.3.8 : WAP to declare a class 'Book' having data members as name, price and no_of_pages. Accept this data for two objects and display the name of book having greater price.

```
# include < iostream . h >
# include < conio . h >
Class Book {
    int no_pages, price ;
    char name [20] ;
public :
    void accept ( ) ;
    void display (Book, Book) ;
} ;
void book : : accept ( ) {
    cout <<"\n Enter book name, number
        of pages, price :" ;
    cin >> name >> no_pages >> price ;
}
void book : : display (Book b1, Book b2) {
    if (b1 . price > b2 . price)
    cout << b1 . name << "has greater price as"
        << b1 . price ;
else
    cout << b2 . name << "has greater price as"
        << b2 . price ;
}
void main ( ) ;
    Book b1, b2 ;
    b1 . accept ( ) ;
    b2 . accept ( ) ;
    b1 . display (b1, b2) ;
    getch ( );
}
```

Output

```
Enter book name, number of pages, price :
C
120
150
Enter book name, number of pages, price :
C++
150
200
C++ has greater price as 200.
```

2.4 Concept of Constructors

- Constructor is a special member function because its name is same as the class name.
- The constructor is invoked whenever an object of its associated class is created.
- A constructor is declared and defined as follows :

```
class integer
{
    int m, n;
public :
    integer (void) ;    //constructor declared
    ------
    ------
} ;
integer : : integer (void) ; //constructor defined
{
    m = 0 ; n = 0 ;
}
```

The declaration

```
integer int1 ; //object int1 created
```

not only creates the object int1 of type integer but also initializes its data members m and n to zero.

- There is not need to write any statement to invoke the constructor function.
- If a normal member function is defined for zero initialization, we would need to invoke this function for each of the objects seperately. This would be very inconvenient, if there are a large number of objects.

Types of Constructor

1) Default Constructor

2) Parameterized constructor

3) Copy constructor

2.4.1 Defualt Constructor

- A constructor that accepts no parameters is called the **default constructor.**
- The default constructor for class A is `A : : A ( ).`
- If no such constructor is defined, then the complier supplies a default constructor. Therefore a statement such as,

```
A a ;
```

invokes the default constructor or the complier to create the object a.

- The constructor function have some special characteristics.

1) They should be declared in public section.
2) They are invoked automatically when the objects are created.
3) They do not have return types, not even void and therefore, they cannot return values.
4) They cannot be inherited, though a derived class can call the base class constructor.
5) Like other C++ functions, they can have default arguments.
6) Constructors cannot be virtual.
7) We cannot refer to their addresses.
8) An object with a constructor (or destructor) cannot be used as a member of union.
9) They make implicit calls to the operators new and delete when memory allocation is required.

- Remember, when a construcor is declared for a class, initialization of the class objects become mandatory.

2.4.2 Parameterized Constructors

- It is possible to pass arguments to constructor functions. Typically these arguments help to initialize an object when it is created.
- To create parameterized constructor, simply add parameters to the constructor function as the same way you do with normal function.
- When you define constructor's body, use the parameters to initialize objects.
- The constructor that can take arguments are called parameterized constructors.
- Example

```
class student {
        int rollno, per ;
    public :
        student (int r, int p) ;
        ———
        ———
} ;
student : : student (int r, int p) {
        rollno = r ;
        per = p ;
}
```

- When the constructor has been parameterized the object declaration statement such as

```
student s ;
```

may not work

- We must pass the initial values as arguments to the constructor function when an object is declared. This can be done in two ways :

1) By calling the constructor explicity.
2) By calling the constructor implicitly.

```
student s = student (2, 75) ; //explicit call
student s (2, 75) ; //implicit call
```

This method (implicit call), sometimes called the shorthand method, is used very often as it is shorter, looks better and it is easy to implement.

Program 2.4.1 : WAP to declare class time having data members as hrs, min, sec. Write a constructor to accept data and display for two objects

```
# include < iostream . h >
# include < conio . h >
class Time {
    int hr, min, sec ;
public :
    Time (int a, int b, int c) ;
    void display ( ) ;
} ;
Time : : Time (int a, int b, int c) {
    hr = a ; min = b ; sec = c ;
}
void time : : display ( ) {
    cout << "\n\n Hours :" << hr ;
    cout << "\n Minutes :" << min ;
    cout << "\n Seconds :" << sec ;
}
void main ( ) {
    Time t1 (1, 60, 120) ;
    Time t2 (2, 120, 240) ;
    t1 . display ( ) ;
    t2 . display ( ) ;
    getch ( ) ;
}
```

Output

```
Hours : 1
Minutes : 60
Seconds : 120
```

Hours : 2
Minutes : 120
Seconds : 240

Program 2.4.2 : WAP to declare a class 'distance' having data members dist 1, dist 2 and dist 3. Initialize the two data members using constructor and store their addition in third data member using function and display the addition.

```
# include < iostream . h >
# include < conio . h >
class Distance
{
    int dist 1, dist 2, dist 3 ;
public :
    Distance (int a, int b) ;
    void display ( ) ;
} ;
Distance : Distance (int a, int b) {
    dist 1 = a ; dist 2 = b ;
}
Void Distance : : display ( ) {
    dist 3 = dist 1 + dist 2 ;
    cout << "in Addition :" << dist 3 ;
}
void main ( ) {
    Distance d1 (60, 120) ;
    d1 . display ( ) ;
    getch ( ) ;
}
```

OR

```
# include < iostream . h >
# include < conio . h >
Class Distance {
    int dist 1, dist 2, dist 3 ;
    public :
Distance (int a, int b) {
    dist 1 = a ; dist 2 = b ;
}
void display ( ) ;
} ;
Void Distance : : display ( ) {
    dist 3 = dist 1 + dist 2 ;
    cout << "Addition :" << dist 3 ;
}
void main ( ) {
    Distance d1 (60, 120) ; d1 . display ( ) ;
    getch ( ) ;
}
```

Program 2.4.3 : WAP to declare a class student having data members as name and percentage write constructor to intilialize these data members. Accept and display data for one object.

Using Default constructor -

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
class student{
    char name[20];
    int per;
    public :
    student(){
    strcpy (name, " ");
    per = 0;
    }
    void accept( ){
    cout <<"\n Enter name and percentage : ";
    cin >> name >> per;
    }
    void display( ){
    cout <<"\n Name : "<< name;
    cout <<"\n Percentage : "<< per;
    }
};
void main( ){
    student s1;
    s1 . accept ( ); s1 . display( ); getch( );
}
```

Using Parameterized constructor -

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
class student{
    char name[20];
    int per;
    public:
        student(char n[20], int p){
        strcpy(name, n);
        per = p;
    }
    void display( ){
        cout<<"\n Name : " <<name;
        cout<<"\n Percentage "<<per
    }
};
    void main( ){
    student s1 ("ABC" , 88);
```

```
    s1 . display( );
    getch( );
}
```

Note :

- The constructor function can also be defined as inline function.

eg.-

```
class student {
    int rollno, per ;
public ;
    student (int x, int y)
    {
        rollno = X ;
        per = Y ;
    }
        _____
        _____
        _____
} ;
```

The parameters of a constructor can be of any type except that of the class to which it belongs.

eg. -

```
class A
{
    _____
    _____
public :
    A (A) ;
} ;
```

is illegal.

Program 2.4.4 : Define a class fraction, which has numerator and denominator as data members. Write the constructor and display fraction

```
# include < iostream . h >
# include < conio . h >
class Fraction {
    int n, d ;
public :
    Fraction (int a, int b) ;
    void display ( ) ;
} ;
Fraction : : Fraction (int a, int b) {
    n = a ; d = b ;
}
void Fraction : : display ( ) {
    cout << "Fraction :" << n/d ;
}
void main ( ) {
    Fraction f (5, 2) ;
    f . display ( ) ;
    getch ( ) ;
}
```

Program 2.4.5 : Define a class to represent a bank account. Include the following members. Data members : Name, account number, account type, balance amount and constructor to assign initial value and member function :

1) To deposit an amount.

2) To withdrawl an amount.

3) To display name and balance.

```
# include < iostream . h >
# include < conio . h >
# include < string . h >
class Bank - Account {
    int acc_no, bal_amt ;
    char name [30],acc_type [20] ;
public :
Bank_Accout (char n [30], int an, char at [20], int ba) ;
void deposit ( ) ;
void withdrawl ( ) ;
void display ( ) ;
} ;
Bank_Account : : Bank_Account (char n [30],
    int an, char at [20], int ba) {
    strcpy (name, n) ;
    acc_no = an ;
    strcpy (acc_type, at) ;
    bal_amt = ba ;
}
void Bank_Account : : deposit ( ) {
    int dep_amt ;
    cout << "\n Enter deposit amount :" ;
    cin >> dep_amt ;
    bal_amt = bal_amt + dep_amt ;
    cout << "\n Amount " << dep_amt << " is
    deposited to an account" << acc_no
    << "\n Available balance :" << bal_amt ;
}
void Bank_Account : : withdrawl ( ) {
    int wd_amt ;
    cout << "\n Enter withdrawl amount :" ;
    cin >> wd_amt ;
```

```
        bal_amt = bal_amt – wd_amt ;
        cout << "\n Amount :" << wd_amt << "is
        withdrawl from an account" << acc_no <<
        "\n Available balance :" << bal_amt ;
    }
    void Bank_Account : : display ( ) {
        cout << "\n Name :" << name << "\n
        Available Balance :" << bal_amt ;
    }
    void main ( ) {
        Bank_Account X ("ABC", 765, "FIX", 10000) ;
        clrscr ( ) ;
        X · deposit ( ) ;
        X · withdrawl ( ) ;
        X · display ( ) ;
        getch ( ) ;
    }
```

Program 2.4.6 : WAP to create class linear having data members var 1, var 2 to represent a linear equation z = (var 1) X + (var 2) Y. Use constructor to initialize the data members. Write function for addition of two linear equations. Display the result.

```
# include < iostream · h >
# include < conio · h >
class Linear {
    int var 1, var 2 ;
public :
    Linear (int, int) ;
    void display ( ) ;
} ;
Linear : : Linear (int V1, int V2) {
    var 1 = V1 ; var 2 = V2 ;
}
void Linear : : display ( ) {
    int z = var 1 + var 2 ;
    cout << "\n" << var 1 << "x +" << var 2
    << "y = " << z ;
}
void main ( ) {
    Linear l (3, 6) ;
    clrscr ( ) ;
    l · display ( ) ;
    getch ( ) ;
}
```

- In default argument initialization of default value to variable is done only in function declaration or function definition not in both.

2.5 Constructors with Default Argment

For example

- The constructor student ( ) can be decalred as follows :

```
student (int rn, float per = 80) ;
```

The default value of argument per is 80. Then, the statement,

```
student S (2) ;
```

assigns the value 2 to rn and 80 to per (by default).

However, the statement

```
student S (3, 75) ;
```

assigns 3 to rn and 75 to per. The actual parameter override the default value.

- It is important to distinguish between the default constructor,

```
student : : student ( ) ;
```

and the defualt argument constructor

```
student : : student (per = 81)
```

- The default argument constructor can be called with either one argument or no argument when called with no argument, it becomes a default consturctor.
- When both these forms are used in a class, it causes ambiguity for a statement such as

```
student S ;
```

The ambiguity is whether to 'call' student : : student ( ) or student : : student (per = 81).

- Default argument write after other normal argument.

```
SALARY (float ta, float basic, float da = 200 float hra =
    400); //correct
SALARY (float da = 200, float hra = 400, float ta, float
basic) ; //wrong
```

Program 2.5.1 : Define a class 'salary' which will contain member variable Basic, TA, DA, HRA : Write a program using constuctor with default values for DA and HRA calculate salary of employee.

```
# include < iostream · h >
# include < conio · h >
class SALARY {
    float BASIC, TA, DA, HRA ;
public :
    SALARY (float, float, float, float) ;
void display ( ) ;
} ;
```

```
SALARY : : SALARY (float ta, float basic,
        float da = 200, float hra = 400) {
   BASIC = basic ;
   DA = (BASIC * da) / 100 ;
   HRA = (BASIC * hra) / 100 ;
   TA = (BASIC * ta) / 100 ;
}
void SALARY : : display ( ) {
   cout << "\n Salary of employee :" <<
      (DA + HRA + TA + BASIC) ;
}
void main ( ) {
   SALARY S (110, 6000) ;
   S . display ( ) ;
   getch ( ) ;
}
```

Output

```
Salary of employee : 48600
```

SALARY (float ta, float hra = 400, float basic, float da = 200) ; // wrong

- Write default arguments after normal variable.

Program 2.5.2 : Program to create a class having data members as principle, rate_of_interest and no_of_years. Assign rate_of_interest = 11.5 as default value. Calculate simple interest and display it.

```
# include < iostream . h >
# include < conio . h >
class SI {
   float r ;
   int p, y ;
public :
   SI (int a, int b, float c) ;
void display ( ) ;
} ;
SI : : SI (int a, int b, float c = 11.5) {
   p = a ; y = b ; r = c ;
}
void SI : : display ( ) {
   int SI = (p * r * y) / 100 ;
   cout << "\n Simple Interest :" << si ;
}
void main ( ) {
   SI S (1000, 4) ;
   S . display ( ) ;
   getch ( ) ;
}
```

Output

```
Simple Interest : 460
```

Note : You cannot assign default value to the variable in constructor in both constructor declaration and constructor definition i.e.

```
SI (int a, int b, float c) ;      //correct
SI : : SI (int a, int b, float c = 11.5) {_}    // correct
OR
SI (int a, int D, float c = 11.5) ;    // correct
SI : : SI (int a, int b, float c) {_}    //correct
OR
```

NOT OK : SI (int a, int b, float c = 11.5) ; //Not correct

NOT OK : SI (int a, int b, float c = 11.5) [---]//Not correct

Program 2.5.3 : Define a class student which will contain member variables as roll_no, name and course. Write a program using constructor with default values as "Computer Engineering" for course. Accept this data for two objects of class and display the data

```
# include < iostream . h >
# include < conio . h >
# include < string . h >
Class student {
   int roll no ;
   char name [30], course [30] ;
public :
   student (char str [30] = "Computer Engineering");
   void accept ( ) ;
   void display ( ) ;
} ;
student : : student (char str [30]) {
   strcpy (course, str) ; }
void student : : accept ( ) {
   cout << \n Enter rollno and name :" ;
   cin >> rollno >> name ; }
void student : : display ( ) {
   cout << "\n\n Rollno is :" << roll no ;
   cout << "\n Name is :" << name ;
   cout << "\n Course is :" << course ;
}
void main ( ) ;
   student S [2] ;
   for (int i = 0; i < 2 ; i + +) {
      S [i] . accept ( ) ; }
   for (int j = 0 ; j < 2 ; j + +) {
      S [j] . display ( ) ;
}
   getch ( ) ;
}
```

2.5.1 Multiple Constructors in a Class

So far, we have used two kinds of constructors. They are :

```
integer ( ) ;          // No argument
integer (int, int) ;// Two argument
```

- In the first case, the constructor itself supplies the data values and no values are passed by the calling program.
- In the second case, the function all passes the appropriate values from main( ).
- C ++ permits us to use both these constructors in the same class.

For example :

```
class student {
    int rollno, per ;
public :
    student ( ) { rollno = 0; per = 0; }
    student (int a, int b) {
        rollno = a ;
        per = b ;
    }
    student (student & S)
        // Note Argument is passed by reference
    {
        rollno = S . rollno ;
        per = S . per ;
    }
} ;
```

This declares three constructors :

- The first constructor receives no argument.
- The second receives two argument.
- The third receives object as an argument.

For example - the declaration

```
student SI ;
```

(1) Would automatically invoke the first constructor and set both rollno and per of SI to zero.

(2) The statement

```
student S1 (2, 80) ;
```

Would call the second constructor which will initialize the data members rollno and per of S2 to 2 and 80 respectively.

(3) Finally, the statement

```
student S3 (S2) ;
```

Would invoke the third constructor which copies the values of S2 into S3. In other words, it sets the value of every data element of S3 to the value of the corresponding data element of S2. Such constructor is called **copy constructor.**

- The process of sharing the same name by two or more functions is referred to as function overloading.
- Similarly, when more than one constructor function is defined in a class, we say that the **constructor is overloaded.**

Example

```
# include < iostream . h >
# include < conio . h >
class Addition {
    int num 1, num 2, num 3 ;
Public :
    Addition ( ) ;
    Addition (int a, int b) ;
    Addition (Addition & ad) ;
    void display ( ) ;
} ;
Addition : : Addition ( ) {
    num 1 = 0 ;
    num 2 = 0 ;
}
Addition : : Addition (int a, int b) {
    num 1 = a ;
    num 2 = b ;
}
Addition : : Addition (Addition & Ad) {
    num 1 = ad . num 1 ;
    num 2 = ad . num 2 ;
}
void Addition : : display ( ) {
    num 3 = num 1 + num 2 ;
    cout << "\n\n Addition is :" << num 3 ;
}
void main ( ) {
    Addition a1, a2 (5, 6), a3 (a2) ;
    clrscr ( ) ;
    a1 . display ( ) ;
    a2 . display ( ) ;
    a3 . display ( ) ;
    getch ( ) ;
}
```

Output

```
Addition is : 0
Addition is : 11
Addition is : 11
```

2.6 Destructors

- A **destructor**, as the name implies, is used to destroy the objects that have been created by a constructor.
- Like a constructor, the destructor is a member function whose name is the same as the class name but is preceded by a tide (~).
- For example, the destructor for the class student can be declared as below :

```
~ student ( ) ;
```

- A destructor never takes any argument nor does it return any value.
- It will be invoked implicitly by the complier upon exit from the program, to clean up storage that is no longer accessible.
- It is good practice to declare destructors in a program since it release memory space for future use.
- New is used to allocate memory in the constructor, we should use delete to free that memory.

Example

```
# include < iostream . h >
# include < conio . h >
int count = 0 ;
class Destructor {
public :
Destructor ( ) {
   cout + + ;
   cout << "\n No. of object created :"
      << count ; .
}
~ Destructor ( ) {
   cout << "in No. of object destroyed :"
      << count ;
   count - - ;
}
void main ( ) {
   cout << "\n\n" ;
   Destructor d1, d2 ;
   cout << "\n\n" ;
   getch ( ) ; }
```

Output

```
No. of object created : 1
No. of object created : 2
```
First Time excecution / Run

```
No. of object destroyed : 2
No. of object destroyed : 1

No. of object created : 1
No. of object created : 2
```
Second Time excecution / Run

```
No. of object destroyed : 2
No. of object destroyed : 1

No. of object created : 1
No. of object created : 2
```
Thid Time excecution / Run

2.6.1 Difference between Constructor and Destructor

	Constructor	Destructor
Purpose	Constructor is used to initialize the instance of a class.	Destructor destroys the objects when they are no longer needed.
When Called	Constructor is Called when new instance of a class is created.	Destructor is called when instance of a class is deleted or released.
Memory Management	Constructor allocates the memory.	Destructor releases the memory.
Arguments	Constructors can have arguments.	Destructor can not have any arguments.
Overloading	Overloading of constructor is possible.	Overloading of Destructor is not possible.
Name	Constructor has the same name as class name.	Destructor also has the same name as class name but with (~) tiled operator.
Syntex	ClassName (Arguments) { //Body of Constructor }	~ ClassName() { }