**Classes and Object**

1. **C Structures:**

Structures provide a method for packing together data of different types. A structure is a convenient tool for handling a group of logically related data items. It is a user-defined data type with a *template* that serves to define its data properties. Once the structure type has been defined, we can create variables of that type using declarations that are similar to the built-in type declarations. **For example,**

|  |
| --- |
| struct student  {  int roll\_number;  float total;  } ; |

**The keyword struct declares student as a new data type that can hold three fields of different data types. These fields are known as *structure members* or *elements.***The identifier student, which is referred to as ***structure name* or *structure tag****,* can be used to create variables of type student.

Example: **struct student A; *//* C *declaration***

A is a variable of type student and has three member variables as defined by the template. Member variables can be accessed using the *dot* or *period operator* as follows:

|  |
| --- |
| A.roll number = 999;  A.total marks = 595.5;  final\_total = A.total + 5; |

**Limitations of C Structure**

The standard C does not allow the struct data type to be treated like built-in types. For example,

struct complex

{

float x;

float y;

} ;

struct complex c1, c2, c3;

The complex numbers c1, c2, and c3 can easily be assigned values using the dot operator, but we cannot add two complex numbers or subtract one from the other. For example,

c3 = cl + c2; // is illegal in C.

Another important limitation of C structures is that they do not permit *data hiding.*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

1. **Specifying a Class**

A class is a new way of creating and implementing a user defined data type.

A class is a way to bind the data and its associated functions together**. It allows the data and functions to be hidden, if necessary, from external use. When defining a class, we are creating a new *abstract data type* that can be treated like any other built-in data type.** A class specification has two parts:

**1. Class declaration**

**2. Class function definitions**

The class declaration describes the type and scope of its members. The class function definitions describe how the class functions are implemented.

**The general form of a class declaration is:**

|  |
| --- |
| class class-name  {  private:  variable declarations;  function declarations;  public:  variable declarations;  function declaration;  }; |

The keyword class specifies that what follows is an abstract data of type ***class\_name.***The body of a class is enclosed within braces and terminated by a semicolon. The class body contains the declaration of variables and functions. These functions and variables are collectively called ***class members****.* They are usually grouped under two sections, namely, ***private***and ***public*** to denote which of the members are *private* and which of them are *public.* The keywords private and public are known as visibility labels.

The class members that have been declared as private can be accessed only from within the class. Public members can be accessed from outside the class also. By default, the members of a class are private. If both the labels are missing, then, by default, all the members are private.

**The variables declared inside the class are known as *data members* and the functions are known as *member functions.***Only the member functions can have access to the private data members and private functions. However, the public members (both functions and data) can be accessed from outside the class. **The binding of data and functions together into a single class-type variable is referred to as *encapsulation.***

**Class Example :**

class item

{

int number *//* variables declaration

float cost; //private by default

public:

void getdata(int a , float b);

void putdata(void);

}



The class item contains two data members and two function members. The data members are private by default while both the functions are public by declaration. The function *getdata()*can be used to assign values to the member variables number and cost, and *putdata()* for displaying their values. These functions provide the only access to the data members from outside the class.



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**3) Creating Objects**

The declaration of item as shown above does not define any objects of itembut only specifies what they will contain. Once a class has been declared, we can create variables of that type by using the class name.

**Example. item x,y,z;**

**Creates a variable x,y,z are variables of type item. The class variables are known as *objects****.* The necessary memory space is allocated to an object at this stage

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**4) Accessing Class Members**

The private data of a class can be accessed only through the member functions of that class. The main() cannot contain statements that access number and cost directly.

Format for calling a member function:

|  |
| --- |
| **object-name.function-name (actual-arguments);** |

**For example:**  **x.getdata(10,75.5) ;**

is valid and assigns the value 10 to number and 75.5 to cost of the object x by implementing the getdata() function.

A variable declared as public can be accessed by the objects directly.

Example: class **xyz**

{

int x;

int y;

public:

int z;

} ;

void main()

{

xyz p;

p.x= 10; // error is private

p. z = 20 //OK, z is public

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**5) Defining Member Functions**

Member functions can be defined in two places:

1. **Outside the class definition.**
2. **Inside the class definition.**
3. **Outside the Class Definition**

Member functions that are declared inside a class have to be defined separately outside the class. An important difference between a member function and a normal function is that a member function incorporates a membership 'identity label' in the header. This 'label' tells the compiler which class the function belongs to. The general form of a member function definition is:

|  |
| --- |
| **return-type class-name :: function-name (argument declaration)**  **{**  **function body**  **}** |

**The membership label class-name :: tells the compiler that the function function-namebelongs to the class class-name. That is, the scope of the function is restricted to the class-namespecified in the header line. The symbol :: is called the scope resolution operator.**

Ex: the member functions getdata() and putdata()as discussed above. They may be coded as follows:

void **item::** getdata(int a, float b)

{

number = a;

cost = b;

}

void **item::** putdata(void)

{

cout « "Number " «number";

cout «"Cost '''«cost”;

}

**Characteristics of member function:**

1. Several different classes can use the same function name. **The 'membership label'** will resolve their scope.
2. Member functions can access the private data of the class. A non-member function (main) cannot do .
3. A member function can call another member function directly, without using the dot operator.

**b.) Inside the Class Definition**

Another method of defining a member function is to replace the function declaration by the actual function definition inside the class.

**example, we could define the item class as follows:**

class item

{

int number;

float cost;

public:

void getdata (int a, float b);  *// declaration*

void putdata(void) *II definition inside the class*

{

cout « number « "\n";

cout «cost «"\n";

};

When a function is defined inside a class, it is treated as an inline function. Therefore, all the restrictions and limitations that apply to an **inline** function are also applicable here. Normally, only small functions are defined inside the class definition.

**A C++ Program with Class Implementation**

#include <iostream>

using namespace std;

class item

{

int number; *// private by default*

float cost;  *// private by default*

public :

void getdata(int a, float b); *// prototype declaration,*

*//Function defined inside class*

void putdata(void)

{

cout « "number :" « number « "\n";

cout « "cost :" « cost « "\n";

}

};

*//*Member Function Definition

void item :: getdata(int a, float b)  *//use membership label*

{

number = a;  *// private variables*

cost = b; *// directly used*

}

// Main Program

int main()

{

item x; *// create object x*

cout « "\nobject x " « "\n";

x.getdata(100, 299.95);  *// cal member function*

x.putdata (); *// call member function*

item y; *// create another object*

cout « "\nobject y" « "\n";

y.getdata(200, 175.50);

y.putdata() ;

return 0;

}

This program features the class item. This class contains two private variables and two public functions. The member function getdata() which has been defined outside the class supplies values to both the variables. Note the use of statements such as

**number = a;**

in the function definition of getdata(). This shows that the member functions can have direct access to private data items. The member function putdata() has been defined inside the class and therefore behaves like an inline function. This function displays the values of the private variables number and cost.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**6) Making an Outside Function Inline**

One of the objectives of OOP is to separate the details of implementation ( execution) from the class definition. It is therefore good practice to define the member functions outside the class.

We can define a member function outside the class definition and still make it inline by just using the qualifier **inline** in the header line of function definition.

Example: class item

{

public: void getdata(int a, float b); *// dec1aration*

};

**inline** void item :: getdata(int a, float b) */ / definition*

{

number = a;

cost = b;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**7) Nesting of Member Functions**

A member function can be *called* by using its name inside another member function of the same class. This is known as nesting of member function*s*

class nest  
{  
         int a,b;  
         public:  
             void get();  
             void put();  
             int large();  
};  
void nest::get()  
{  
        cout<<"enter a and b:";  
        cin>>a>>b;  
}  
void nest::put()  
{  
            cout<<"largest number:"<<large();  
}  
int nest::large()  
{  
       if(a>b)  
        return a;  
     else  
       return b;  
}  
void main()  
{  
 clrscr();  
        nest x;  
       x.get();  
       x.put();  
 getch();  
}

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

**8) Private Member Functions**

Some situations may require certain functions to be hidden (like private data) from the outside calls. A private member function can only be called by another function that is a member of its class. Even an object cannot invoke a private function using the dot operator.

class sample

{

int m;

void read (void) ; *// private member function*

public:

void update (void);

void write(void);

};

If s1 is an object of sample, then

sl.read(); *// won't work; objects cannot access*

*// private members*

is illegal. However, the function read() can be called by the function update() to update the value of m.

void sample:: update (void)

{

read (); // simple call; no object used

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**9) Arrays within a Class**

The arrays can be used as member variables in a class.

class array

{

int a [10];

public:

void setval(void);

void display (void);

} ;

The array variable **a[ ]** declared as a private member of the class **array** can be used In the member functions, like any other array variable. We can perform any operations on it. For instance, in the above class definition, the member function **setval()** sets the values of elements of the array **a[ ],** and **display()** function displays the values. Similarly, we may use other member functions to perform any other operations on the array values.

* Arrays can be declared as the members of a class.
* The arrays can be declared as private, public or protected members of the class.
* To understand the concept of arrays as members of a class, consider this example.

A program to demonstrate the concept of arrays as class members

**#include<iostream>**

const int size=5;

class student

{

int roll\_no;

int marks[size];

public:

void getdata ();

void tot\_marks ();

} ;

void student :: getdata ()

{

cout<<"\nEnter roll no: ";

Cin>>roll\_no;

for(int i=0; i<size; i++)

{

cout<<"Enter marks in subject"<<(i+1)<<": ";

cin>>marks[i] ;

}

void student :: tot\_marks() //calculating total marks

{

int total=0;

for(int i=0; i<size; i++)

total+ = marks[i];

cout<<"\n\nTotal marks "<<total;

}

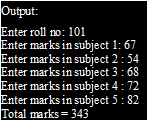
void main()

student stu;

stu.getdata() ;

stu.tot\_marks() ;

getch();}



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*