2. Classes and Methods

Let's start with

Introduction to C++

22

Introduction to C++

- Object oriented programming language
- Developed by Bjarne Stroustrup in 1979 at Bell Laboratories, New Jersey
- In 1998 ANSI/ISO standards committee provide the final draft of C++
- Object oriented extension of C i.e, addition of classes to C
- Initially known as C with classes
- later the name changed to C++ i.e, incremented version of C
- Almost all features of C is also applicable to C++

23

Beginning with C++

- Any C++ compiler/IDE such as Turbo C++, Borland C++, Dev C++ can be used to write C++ program
  - **Steps to use C++ compiler/IDE:**
- Open C++ compiler window
- Open text editor
- **Write a program…**
- Save the file
- Then compile
- Run the program

  A new window with output of the program is shown

A simple C++ program comparing with C

```
/*  C program that prints
    Hello BEE on screen */
#include <stdio.h>
 main ( )
{
   printf("Hello BEE \n");
   return 0;
}
```

```
/*  C++ program that prints
    Hello BEE on screen */
#include <iostream>
using namespace std;
 int main ( )
{
   cout<<"Hello BEE \n";
   return 0;
}
```

25

Basic Elements of C++

Basic vocabulary/elements used in C++ programs:

- Tokens
- Statements
- Variables
- Data Types

26

Tokens

Smallest individual units in a program which may be:

- keywords (reserved words that can't use for anything else)
- identifiers (name given to variable, function etc. e.g. variable names, function names like "sum", "main" ...)
- constants (which does not change e.g. the number 5)
- strings (sequence of chracters e.g. "Hello\n")
- operators (symbol to indicate task e.g. +, -, =)
- punctuators (symbols e.g. ; , {})
- whitespace (Spaces of various types; ignored by the compiler e.g. space, newline etc)

27

Statements

–unit of code that that perform task

–a basic building block of a program ends with semicolon

–e.g. cout << "Hello, world!\n";

Variables

– Quantity which may vary (change)

– should declare at the beginning of the program before they are used

– Syntax for declaring variable is:

• *Type-name* variable name,…, variable name;

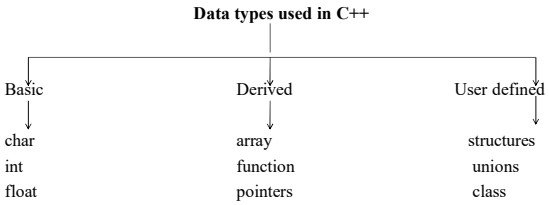
• Type- name refers data type

• Eg : int x, y, area;

28

Data Types

Refers to size and type of data that can be stored



29

```
/*Program to demonstrate structures*/
#include <stdio.h>
#include <conio.h>
struct book
{char name;
float price; int pages;};
void main()
{struct book b1={'A',23.5,5};
struct book b2;
b2.name='C';
b2.price=45.0;
b2.pages=23;
printf("\n%c %.2f %d", b1.name, b1.price, b1. pages);
printf("\n%c %.2f %d", b2.name, b2.price, b2. pages); getch();}
```

Limitations of Structures

- Does not allow the struct type to be treated like built-in (basic) data type
- Structures members are public

❖ To resolve these limitations, structures with OO approach called class was introduced in C++

31

…

Classes

- Collection of objects of similar type
- Objects are variables of the type class
- User defined data type
- Can have variables and functions as member
- Members may be private or public
- Class specification has two parts:
  - Class declaration (describe types & scope of the members)
  - Function definitions (describe how the class functions are implemented)

32

Classes and Methods…

Review of structures

- Collection of one or more than one variable, possibly of deferent data type, grouped together under a single name for convenient handling
- individual structure elements are referred to as members
- **Declaring Structure:**

```
Syntax:
        struct name
        {
        member 1;
        member 2;
        member 3;
        ……;
        member n;
        };
    struct name v1,v2,…vn;
```

- For accessing structures element/member dot (.) operator is used as: v_name.m_name

30

…

Class declaration

- Similar to structure declaration
- Syntax:

```
class c_name
{
access specifier/visibility labels:
variable declaration(data member);
function declaration(member function);
};
```

eg:

```
class book
{private:
int pages;
float price;
public:
void getdata(int b,float c);
void display();
};
```

- Function need to be defined
- Access specifier: A keyword(public,private,protected) that controls the access to members of the class

33

…

Creating objects

- Class variables are called objects
- After declaring class objects are created as:
  - c_name o_name;
  - More than one object can also be created
  - Eg: book b1;
    book b1,b2; etc

Accessing class members

- Using dot (.) operator class member can be accessed as:
  - o_name.f_name(actual arguments);
  - Eg: b1.getdata(123,43.5);

…

```
/*  C++ program with class*/
#include <iostream.h>
class book
{private:
int pages;
float price;
public:
void getdata(int b,float c)
{pages=b;
price=c;}
void display()
{cout<<pages<<" "<<price;}};
```

```
int main()
{
book b1;
b1.getdata(123,34.5);
b1.display();
cin.get();
return 0;}
```

- Output:

123 34.5

…

- The variables declared inside the class are known as *data members* and the functions are known as *member functions*.
- Members may be *private* (access within the class only, by default all members are private), *public* (access from outside of the class also) and *protected* (accessible to the derived class of the given class i.e. needed only when inheritance is involved) .
- Binding of data and functions together into a single class-type variable is referred to as *encapsulation*
- Insulation of data from direct access is called *data/information hiding*
- One of the possible condition of an object may changes at any time is called *state*
- How an object acts and reacts in terms of message passing and state changing is called *behavior*
- *B*ehavior for which an object held accountable is called *responsibility*
- An operation upon an object such as message passing is called *method*

…

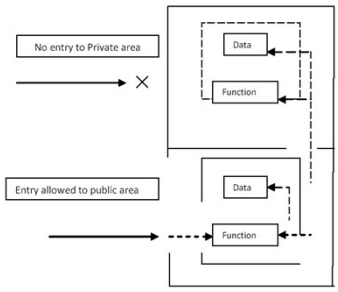


Fig: Data hiding and Encapsulation in class

Classes and Methods…

- /* This program display item number and cost */

```
# include <iostream>
//using namespace std;
class item
{
int num;
float cost;
 public:
void getdata(int a, float b)
{
num=a;
cost=b;
}
void display(void)
{
cout << num << "\n";
cout << cost << "\n";}};

int main ( )
{
        item x;
  x.getdata(101, 145.5);
  x.display();
return 0;
}
```


…

- Defining member functions (Function definition)

  Member functions need to be defined and it can be defined in two ways:

  a) Inside the class definition

  b) Outside the class definition

- **Inside the class definition**

  Method of defining a member function is to replace the function declaration by the actual function definition inside the class as:

class *class-name {*
access specifier (private/public):
// declare variables
// declare/define function as
return type function-name(argument declaration)
{ function body } };

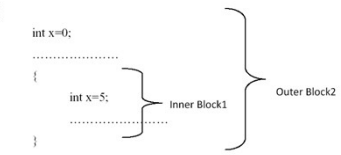
...

```
/*function definition inside class.
Program for calculating area*/
# include <iostream>
//using namespace std;
class areacal
{
int br, l,ar;
public:
void getdata(int a, int b)
{
br=a;
l=b;}
void display (void)
{
ar=br*l;
cout << ar;
}
};
void main ()
{
areacal area;
area.getdata(3,5);
area.display ();}
```


...

Scope Resolution Operator(::)

- The operator :: is called scope resolution operator. It is used to link a class name with a member name in order tell the compiler what class the member belongs to. However, it has another related use, it can allow access to a name in an enclosing scope that is hidden by a local declaration of the same name. As we know that the same variable can be used to have different meaning in different block. The scope of the variable extends from the point of its declaration till the end of the block containing the declaration. A variable declared inside block is said to be local to that block,
- eg:



...

Outside the class Definition

- Member functions that are declared inside a class have to be defined separately outside the class. Their definitions are very much like the normal function. They should have a function header and a function body.
- However a member function incorporates a membership identity label in the header. This label tells the compiler which class the function belongs to.
- Syntax:

```
return type class_name::function-name(argument declaration)
{
function body
}
```

- Here the symbol :: is called scope resolution operator.

...

:: …

- Here block1 is contained in block2 and hides the declaration of same variable in outer block. This can be receive by using a scope operator as:

  ::variable name;

- Example:

```
#include<iostream>
int m=10;
void main()
{
int m=20;
cout<<"M="<<m<<"\n";
cout<<"\nM="<<::m;
}
```

•output
M=20
M=10

...

```
/*function definition outside class.
Program for calculating area*/
# include <iostream>
//using namespace std;
class areacal
{int br, l,ar;
public:
void getdata(int a, int b);
void display (void);
};
void areacal::getdata(int a, int b)
{
br=a;
l=b;}
void areacal::display (void)
{
ar=br*l;
cout << ar;
}
void main ()
{
areacal area;
area.getdata(3,5);
area.display ();
cin.get();
}
```


...

Varieties of Classes:

- On the basis of task to be performed there are four types of classes:
  a) Data Manager
  b) Data sink or data source
  c) View or observer class
  d) Facilitates or helper class
- **Data manager:** It is main class and its responsibility is to maintain the data or state information. This type of class can hold data and state information which may used by another class. Data manager class is also known as Data or state class. Suppose, in a software system, a class maintain database and arrange the data values which facilitates the other classes to easily access the database. Here, this class is known as data manager class.

- **Data sink or data source :** The class which generates data, or that accept data, and process them for further is known as data sink or data source class. This class does not hold data for certain period of time but it generates data and supply to another class. Data sink accept data and data source means supply data, therefore, this type of class can supply data and accept supplied data and process further. For example, a class generates random number and supply the generated number and another class will accept that number and process further. Here, both the classes are data sink or data source.
- **View or observer classes:** This type of class is concern with displaying information on an output devices such as monitor. To display information in attractive form the code may be complex and frequently modified and independent of actual data therefore there type of class only handle the display behavior.
- **Facilitator or helper class:** This class is mainly helper class of Data manager class and other classes which facilitates the proper maintaining data, proper display information etc.

Functions:

- A function is a self-contained program segment that carried out some specific and well defined task
- Information is passed (value send) to the function via special identifiers called arguments (input data to the function) and return a value from return statement
- A function consists of three parts:
  - Function prototype/function declaration
  - Function definition
  - Function call/Procedure call (in OOP this can be defined as message passing)
- C++ support all most all concepts of functions used in C
- Contain other various new features of functions such as inline function, friend function, static function etc.

Inline Function:

- Functions save memory space because same code can be called from different portion
- Takes extra time to execute when function is called due to tasks such as jumping to the function and returning to the calling function
- It is possible to define functions that are not actually called but, rather, are expanded in line, at the point of each call i.e. when the function is called, the actual code from a function is inserted, instead of a jump to the function
- Applicable for short functions, inline function is defined using keyword inline as:

```
inline return type function_name(arguments declare)
{
function body
}
```

- Member function defined inside the class will automatically become an in-line function


```
// Example of inline function
#include<iostream.h>
inline int max(int a, int b)
{
return a > b ? a : b;
}
void main()
{
cout<<max(10,20);
cout<<" "<<max(99,88);
cin.get();
}
```


Friend Function:

- Any data function is declared private inside a class is not accessible from outside the class. A function which is not a member or an external class can never access such private data. But there may be some cases, where a programmer will need access to the private data from non-member functions and external classes; C++ offers some exception in such cases.
- A class can allow non-member functions and other classes to access its own private, by making them as friends. A friend function posses certain special characteristics:

a) It is not the scope of the class to which it has been declared as friend.

b) Since it is not in the scope of the class, it cannot be called using the object of that class.

c) It can be invoked like normal function without the help of any object.

d) Unlike member function, it cannot access the member names directly and has to use an object and dot operator with each member name.

e) It can be declared either in the public or private part of a class without affecting its meaning

f) Usually it has the object as arguments.


```
//Example of friend function
#include<iostream.h>
class myclass
{
int a,b;
public:
friend int sum(myclass x);
void set_ab(int i, int j);
};
void myclass::set_ab(int i,int j)
{
a=i;
b=j;
}
int sum(myclass x)
{
return x.a+x.b;
}
void main()
{
myclass n;
n.set_ab(3,4);
cout<<sum(n);
cin.get();
}
```

3. Message, Instance and Initialization

Message:

- Message for an object is a request for execution of procedure
- An operation that one object performs upon another
- The terms message, method and operation are usually interchangeable

Message Passing Formalization:

- Message passing means process of asking an object to perform specific function. Message passing is like invoke the member function. To perform certain action, member function is called by object

52

…

- A message is always given to some object called receiver.
- Different object may receive the same message but perform different action.
- Three different parts of message passing are:

  Receiver : It is object to which the message is sent

  Message selector: It indicates the particular message.

  Arguments/agents: It is the information to responding message.

53

…

Message passing syntax in C++

- message passing means invoking a member function
- To invoke a member function dot operator is used after the object or receiver as in shown below:

  Object_name.function_name(actual arguments);

  e.g: x.display() ;
- Here x is object and display is member function.

Message Passing (MP) vs. Procedure Calling (PC)

- PC: there is no designated receiver
- MP: Interpretation of the message is dependent on the receiver and can vary with different receivers
- Usually, the specific receiver for any given message will not be known until run time (is dynamic/late binding) , so the determination of which method to invoke cannot be made until then.
  - late binding – between a message and the code fragment (method) used to respond to the message in MP
  - vs. compile-time or link-time binding in PC.

54

…

Creation and Initialization in C++

- Here creation means allocation of memory and initialization means initialization of value in memory

Mechanism for creation and Initialization

- Like variable, an object can be created and initialized
  - Initialize the object that have already been created using member functions such as input(), getdata() etc., which we have seen so far in few examples of c++ till now.
  - Initialization of object at the time of creation, with the help of **constructor**.

55

…

Constructor:

- Constructor is a member function with the same name as the name of a class. it is automatically called when an object of the class is created. It is used to initialize the object of the class. They do not have any return type. It is called constructor because it constructs the values of data member.
- Syntax:

```
class class_name
{
private:
data member declare;
public:
class_name() //creating constructor
{ body of constructor; }};
```

56

…

```
//Example constructor
#include <iostream>
class counter
{
private:
int count;
public:
counter()
{
count=0;
}
void inc_count()
{
count++;
}
void display()
{
cout<<"Count="<<count<<endl;
}
};
void main()
{
counter C; //automatically run counter constructor and set count value to 0
C.display();
C.inc_count();
C.inc_count();
C.display();
cin.get();
}
```

57

...

- When class contains a constructor, it is guaranteed that an object C created by the class will be initialized automatically.
- For example:

  counter C;
- Not only creates the object C of type counter but also initialize its data member count to 0. There is no need to write any statement to invoke the constructor function.
- If a normal member function is defined for zero initialization we would need invoke the function for each of the objects separately.

58

...

Characteristics of Constructor:

a) They should be declared in the public section.

b) They are invoked automatically when the objects are created.

c) They do not have return types, not even a void, and therefore, they cannot return values.

d) They cannot inherit.

e) Constructor cannot be virtual.

f) They make implicit call to the operator new and delete when memory allocation is required.

g) We cannot refer to their address.

59

...

Types of Constructor

a) Default constructor:

- A constructor that accepts no parameter is called the default constructor.

b) Parameterized constructor:

- A parameterized constructor can take values as argument into the constructor function at the time of object declaration. A parameterized constructor is defined in the same way as that of other constructor, the only difference between a default constructor is that, the function headers of these constructors will have parameters and hence different initial values can be given to the object of the same class on the use of constructor.

60

...

- **Syntax:**

```
class class_name
{
data member declare;
public:
class_name(argument list)
{
initialize the data members;
}
};
```

values as arguments to the constructors function can be pass when an object is declared. This can be done in two ways:

i) By calling the constructor explicitly:

```
class_name
object_Name=constructor_name(parameters);
```

ii) By calling the constructor implicitly

```
class_name object_name(parameters);
```

61

...

```
#include<iostream>
class Area
{int length,width;
public:
Area()
{length=0;
width=0;}
Area(int a, int b);
void display();};
Area::Area(int a, int b)
{length=a;
width=b;}
void Area::display()
{int A;
A=length*width;
cout<<"\nThe Area of
    Rectangel="<<A;}
void main()
{
Area obj; //Run default constructor
Area obj1(5,15); /* Run
    parameterize constructor
    calling Implicitly*/
Area obj2=Area(20,25); /* Run
    parameterize constructor
    calling Explicitly*/
obj.display();
obj1.display();
obj2.display();
cin.get();
}
```

62

...

c) Copy constructor

- A copy constructor is a type of constructor which construct an object by copying the state from another object of the same class. Copy constructor is used to declare and initialize an object from another object. When an object is copied, another object is created in this process so it is called copy constructor. It takes a reference to an object of the same class as an argument.
- for example the statement

  integer I2(I1);
- would define the object I2 and at the same time initialize it to the value of I1. This statement would invoke the copy constructor.
- And another form of this statement is

  integer I2=I1;
- here I2 and I1 are object, this statement is legal and simply assigns the value of I1 to I2, member to member.

d) Dynamic constructor

63

...

```
//Example copy constructor
#include<iostream>
class code
{
int id;
public:
code(int a)
{
id=a;
}
code (code &x)
{
id=x.id;
}
void display()
{cout<<id;}};
void main()
{
code A(100); //Object A is created and initialize.
code B(A); // Copy constructor is called.
code C=A; // Copy constructor is called again
cout<<"\n ID of A:";
A.display();
cout<<"\n ID of B:";
B.display();
cout<<"\n ID of C:";
C.display();
cin.get();}
```

64

...

Destructor:

- If allocated memory of constructor is not needed then C++ provides a member function to destroy that memory. The member function which destroys or de-allocates the memory object allocated created by the constructor is called destructor.
- Some properties of destructor are as follows:

a) It has the same name as class name and constructor but preceded by tilde(~)

b) Like constructor, destructors do not return any value.

c) Destructor does not contain any argument.

d) Only one destructor can be declared in a class.

- Due to the use of destructor, memory space is released which can be used in future. Therefore, destructor is used to save memory space in the program.

65

...

```
//Example of Destructor
#include<iostream>
class des
{
int a;
public:
des(int x)
{
a=x;
cout<<"Object is created
    "<<a<<endl;
cout<<"Press any key to end
    the program";}
~des()
{
cout<<"Object destroyed";
}
};
void main()
{
des obj(10);
}
```

66

...

Issues in Creation and Initialization

- Memory allocation and management is most important because large program may use classes with large memory space which should be reduced as far as possible. Following are the important issue in creation and initialization.

1) Stack vs Heap storage allocation.

2) Memory Recovery

3) Pointer

67

...

Stack Vs Heap Memory storage Allocation

- There are mainly two ways to allocate and release memory. They are automatic and dynamic where the automatic and dynamic variables plays key role. The storage for automatic variable is created when procedure or block containing variable declaration is entered or started and released automatically when the procedure is exit. The memory allocation of automatic variable is like stack i.e. last in first out (LIFO). The user cannot release allocated memory.
- Heap storage allocation is used by dynamic variable. In dynamic memory allocation, newly created memory allocated for variable & value can be changed and release at runtime. Here memory is created when request.

68

...

Memory Recovery:

- If allocated memory is no longer use heap based allocation can recover it. When the allocated memory is not used then C++ provides library routine delete to free the memory. Other language like java automatically detects this unused memory and free automatically and reuse. This process is known as garbage collection. In large program large number of variables or data used large memory area. So we need to free up unnecessary memory space.

Pointer:

- Pointer is the most efficient and effective way of dynamic information. Pointer is address of variable. Different types of calculation can also be performed from pointer variables. The OOP language like Java represent pointer internally but in C++ we can explicitly declare the pointer.
- To declare a pointer variable
  int *x;
- where x is a pointer variable.

69

...

Dynamic Memory Allocation (the new and delete operator)

- We use dynamic memory allocating technique when it is not known in advance how much memory space is needed. C++ supports dynamic memory allocation by using two unary operator 'new' to create dynamic memory and 'delete' to release the memory. The allocated memory will not release inside the block until the use of delete operator. The lifetime of an object or data is directly under the controls of users.
- The general syntax of new is

pointer variable=new datatype;

- Here pointer variable is pointer of datatype. The new operator sufficiently allocate memory to hold data.

p=new int;

- We can directly declare variable by using new as

int *x=new int;

- Here x is a pointer variable.

...

- We can also initialize the memory using new operator.

int *x=new int(25);

- Similarly we can release memory by using delete operator.
- syntax:

delete pointer variable;

Eg: delete(x);

- Similarly we can use new and delete operator to allocate and release memory dynamically for objects. This enables the system to allocate the enough amount of memory to each object. So allocation of memory to object at the time of their construction is known as dynamic construction of object.

...

```
//Program to calculate the length of string by allocated memory
  dynamically
#include<iostream.h>
#include<string.h>
#include<conio.h>
void main()
{char *str;
str=new char[25];
int len;
cout<<"Enter string:";
cin>>str;
len=strlen(str);
cout<<"The length of string:"<<len;
delete str; getch();}
```


...

```
// Allocating dynamic objects
#include<iostream>
class samp {
int i, j;
public:
void set_ij(int a, int b) { i=a;
   j=b; }
int get_product( ) { return i*j;
   }};
int main( ) { samp *p;
p = new samp; //allocate object
if (!p) {
cout << "Allocation error\n";
return 1;}
p->set_ij(4, 5);
cout<< "product is: "<<
   p->get_product( ) << "\n";
delete p; // release memory
cin.get();
return 0;
}
```

- **Output:**

```
product is: 20
```


...

Dynamic constructor

we can use new and delete operator to allocate and release memory dynamically for objects. This enables the system to allocate the enough amount of memory to each object. So allocation of memory to object at the time of their construction is known as dynamic construction of object. The memory is allocated with the help of new operator and is also used in constructors also.

```
#include <iostream.h>
#include <conio.h>
class dyncons
{ int * p;
 public:
 dyncons()
 { p=new int;  *p=10; }
 dyncons(int v)
  { p=new int;
   *p=v; }
  int dis()
  { return(*p); }};
void main()
{
clrscr();
dyncons obj, obj1(9);
cout<<"The value of object obj's p
   is:";
cout<<obj.dis();
cout<<"\nThe value of object obj1's p
   is:"<<obj1.dis();
getch();
}
```


Assignments

4) What do you mean by stack versus heap? Explain about memory recovery. Explain the use of new and delete operator with example.
5) Differentiate class and structure with example.
6) What are the advantages of using a friend function? List different types of classes and explain any two.
7) What are constructors and destructors? Explain their types and uses, with good illustrative example?
8) What is access specifier? Describe all access specifier available in c++.
9) Compare and contrast message passing with procedure call.
10) Write short notes on:
    a)scope resolution operator b) data member and member function
    c)inline function