



BEGINNING WITH C++



UNIT – 2

BEGINNING WITH C++

2

2.1 The Main Function
2.2 Function prototyping
2.3 Call by Reference and Return by Reference
2.4 Inline functions
2.5 Default Arguments
2.6 Constant Arguments
2.7 Function Overloading
2.8 Classes and Objects :
2.9 Overview of C structure
2.10 Defining Class and Creating Objects
2.11 Memory Allocation for Objects
2.12 Defining Member function
2.13 Making an outside function Inline
2.14 Nesting of Member functions
2.15 Private Member functions
2.16 Arrays within a Class
2.17 Static Data member and Static Member functions,
2.18 Array of Objects,
2.19 Passing Objects as an Argument, Returning Object,
2.20 Friend function, Pointer to members

2.1 THE MAIN FUNCTION

The main function is the starting point for the execution of the program.

Definition of the main function:

In C++, the main () function returns a value of type int to the operating system.

Prototype of the main() function

```
int main( );
```

```
int main( int argc, char * argv[ ] );
```

The functions that have a return value should use the return statement for termination.

```
int main()
```

```
{
```

```
-
```

```
-
```

```
return 0;
```

```
}
```

If we doesn't specifies the return type of the main () function then C++ takes by default the return type as integer.



2.2 FUNCTION PROTOTYPING

A function prototype tells the compiler

- Name of the function,

- Type of data returned by the function (Return type),

- Number of parameters,

- Types of the parameters, and

- Order in which these parameters are expected.

The compiler use function prototypes to validate function calls.

Syntax:

Return-type function-name(parameters);

Example:

int maximum(int,int);

Here, return type is integer, function name is maximum, no of arguments are 2 of type integer.



Example:

```
int sum(int x, int y);
```

In the function prototype, all the argument variable must be declared independently inside the parenthesis.

In the function declaration, the names of the arguments are dummy variables and therefore they are optional.

Example:

```
int sum(int a, int b)
{
    int total=a+b;
    . . . . .
    . . . . .
}
```



ADVANTAGES OF FUNCTIONS

- It helps in reducing both the physical and executable file sizes in large programs.
- To save memory, because all the calls to a function tell the compiler to execute the same block of code.
- Improve the program's readability, understandability and maintainability.
- Reusability of the code.
- Dividing a program into functions is one of the major principles of top-down, structure programming.



2.3 CALL BY REFERENCE AND RETURN BY REFERENCE

- **There are 3 types of parameter passing techniques:**
- Passing by value (Call by value method)
- Passing by address
- Passing by reference (Call by reference method)



Call By Value:

- When a function is called using the value of variables then it is known as call by value.
- The values of the actual parameter do not be modified by formal parameter.
- In this method, all the process done on the duplicate variables rather than actual variables.

- **Example:**

```
void swap(int,int);           // function declaration
swap(a,b);                   // function call
void swap(inta, int b)       // function definition
{
    int temp;
    temp=a;
    a=b;
    b= temp;
    cout<<"After swapping value of a<<a;
    cout<<"After swapping value of b<<b;
}
```



○ Passing by address (which is call by reference in C):

- When a function is called using the address of variable then it is known as passing by address.
- The values of the actual parameter can be modified by formal parameter.
- In this method, all the process done on the actual variables.

○ Example:

```
void swap(int *,int *);
```

```
// function declaration
```

```
swap(&a,&b);
```

```
// function call
```

```
void swap(int *x,int *y)
```

```
// function definition
```

```
{ int temp;
```

```
    temp = *x;
```

```
    *x = *y;
```

```
    *y = temp;
```

```
}
```



○ Call by Reference:

- When a function is called using the Reference variable then it is known as Call by Reference method.
- Reference variable creates alias for variables.
- So no waste of memory.

- **Example:**

```
void swap(int ,int ) // function declaration
swap(a, b);          // function call
void swap(int &x,int &y) // function definition use Reference variable
{
    int temp;
    temp = x;
    x =y;
    y = temp;
}
```

- Here, variable x and y are reference variable of variable a and b.



- **Return by Reference:**

- A function can also return a reference.

- Example:

```
int & max(int & , int &);  
void main()  
{  
    int a=10, b=20,m;  
    m = max(a, b);  
    cout<<"Maximum nos:"<<m;  
}  
int & swap (int &x, int &y)  
{  
    if( x > y )  
        return x;  
    else  
        return y;  
}
```

- Here, function returns reference to x or y (not value) and return type is int&.



2.4 INLINE FUNCTIONS

- The functions can be made inline by adding KEYWORD inline to the function definition.
- An inline function is a function that is **expanded in line** when it is invoked.
- The **compiler replaces** the function call with the function code.
- Inline function **saves time** of calling function, saving registers etc.
- Preprocessor **macros** of C is similar to inline function but they are not really functions. So usual error checking is not possible.



CONTINUE..

- If function has **very few lines** of code and **simple expressions** then only it should be used as inline otherwise behave as normal function.
- For ex
 - If a loop, a switch , goto exists,
 - If function is not returning any value or contains static variables.
 - If it is recursive.
- Then function does not work as inline.



Example:

```
inline int cube(int n)
{
    return n*n*n;
}

int main()
{
    int c;
    c = cube(10);
    cout<<c;
}
```

Function call is replaced with expression

So, $c = \text{cube}(10);$

Becomes $c = 10 * 10 * 10;$ at compile time.



- **Advantages:**

- Reduce execution time
- Increase readability
- Strong type checking

-

- **Disadvantages:**

- Function definition should appear before function call.
- It is a request to make function inline, it depends on compiler to make inline or not.
- Increase the size of file



2.5 DEFAULT ARGUMENTS

- Default values are **specified** when the function is declared.
- We must add default arguments **from right to left**.
- We **cannot provide** a default value to a particular argument **in the middle of an argument list**.
- Default arguments are **useful** in situations where some arguments always have the same value.



Example:

```
#include <iostream>
void f(int a=0, int b=0)
{
    cout<< "a= " << a << ", b= " << b<<endl;
}
int main()
{
    f();
    f(10);
    f(10, 99);
    return 0;
}
```

Output: a=0 , b=0

a=10 ,b=0

a=10, b=99



2.6 CONSTANT ARGUMENTS

- In c++ ,an argument to a function can be declared as constant using keyword **const**.
- Syntax:
 - **const datatype
variablename;**
- Example1: `int square(const int a);`
- `float area(const float r);`
- In above statement, the **const** keyword tells the compiler that the function should not modify the argument.



Example2: *[Write any one example]*

```
int fun(const int x)
{
    int x=y;
    x=x+1;      //Invalid
    y=x;        //valid
    return (y);
}
```

Const argument cannot modify.



2.7 FUNCTION OVERLOADING

- Overloading means to use the same thing for different purpose.
- C++ allows overloading of functions.
- **Definition:** “When multiple functions with **same name** are used for **different task and purpose** it is known as function overloading”.
- **Compiler Differentiate** it by,
 - Different no of arguments
 - Different types of arguments
 - Different return type
- Example: Area of shape is defined as
float area(float r); //area of circle
float area(float l, float b); //area of rectangle



```
#include<iostream.h>
Using namespace std;
Const float pie=3.14;
float area(float r) //function to compute area of circle
{   return(pie*r*r);
}
float area(float l,float b) //function to compute area of rectangle
{   return(l*b);
}
void main()
{   float r,l,b;
    cout<<"Enter radius:";
    cin>>r;
    cout<<"Enter length and breadth:";
    cin>>l>>b;
    cout<<"Area of circle="<<area (r)<<endl;
    cout<<"Area of rectangle="<<area(l, b)<<endl;
}
```



2.9 OVERVIEW OF C STRUCTURE

structure is another user defined data type available in C that allows to combine data items of different kinds.

Syntax:

```
struct structure_name
{
    data_type member1;
    data_type member2;
    .
    .
    data_type member;
};
```

Example:


```
struct person{    char name[50];    int citNo;    float salary;};
```



2.8 CLASSES AND OBJECTS AND

2.10 DEFINING CLASS AND CREATING OBJECTS

Class

- **Definition:** “Class is a collection of objects.”
 - It allows the data and function to be **hidden** if necessary.
 - Declare the class using keyword **class** and work as **built in data type**.
 - So, we can create the variable of that data-type, which is called as objects.
 - A class specification has generally two parts:
 - **Class declaration:** describes the type and scope of its members.
 - **Class function definitions:** describes how the class functions are implemented.
- 

Syntax of class declaration:

```
class class-name
```

```
{
```

```
    private:
```

```
        variable declarations;
```

```
        function declarations;
```

```
    public:
```

```
        variable declarations;
```

```
        function declarations;
```

```
};
```

- The variable which is declared inside the class declaration that is known as **data members**.
- The functions which is declared inside the class declarations that is known as **member functions**.
- **Private** and **public** specifies the scope (where it is used) of the class members.



Access Specifies:

Private:

- It can be accessed only from within the same class.
- They cannot be accessed outside the class.
- Encapsulation is possible due to the private access modifier.
- Only the member function of the class can access the private data member and the private member functions of the same class.

Public:

- It can be accessed from outside the class also.
- There are no restrictions for accessing public members of a class.

By default, the scope of the class members is **private**.



- **Example of class declaration:**

```
class student
```

```
{    int no;
```

```
    char name[20];
```

```
public:
```

```
    void getdata( );
```

```
    void putdata( );
```

```
};
```

Here, no and name are the data members which has the private scope.

And getdata(), putdata() are the member functions which has the public scope.



Objects:

- Definition: “Object is a basic run-time entity in object oriented system.”
- Object is a one type of class variable.
- Object can be declared same as the variable declaration.
- **Syntax:**

`class-name object-name;`

- There are 2 ways to create object:

Example 1:

`student s1,s2,s3;`

where, student is the class name and s1,s2 and s3 are the object name.



Accessing Class Members:

In C++, we can access the class member outside the class.

For that, we have to use the object of that class.

Syntax:

Object-name.function-name(argument-list);

Example:

```
class student
{
    int no;
    char name[20];
public:
    void getdata( );
    void putdata( );    };

void main()
{
    student s1;
    s1.getdata( ); // member function access
    s1.putdata( ); // member function access
}
```



2.11 MEMORY ALLOCATION FOR OBJECTS

The memory space is allocated to the data members of a class only when an object of the class is declared

Since a single data member can have different values for different objects at the same time.

Every object declared for the class has an individual copy of all the data members.



Example:

Class book

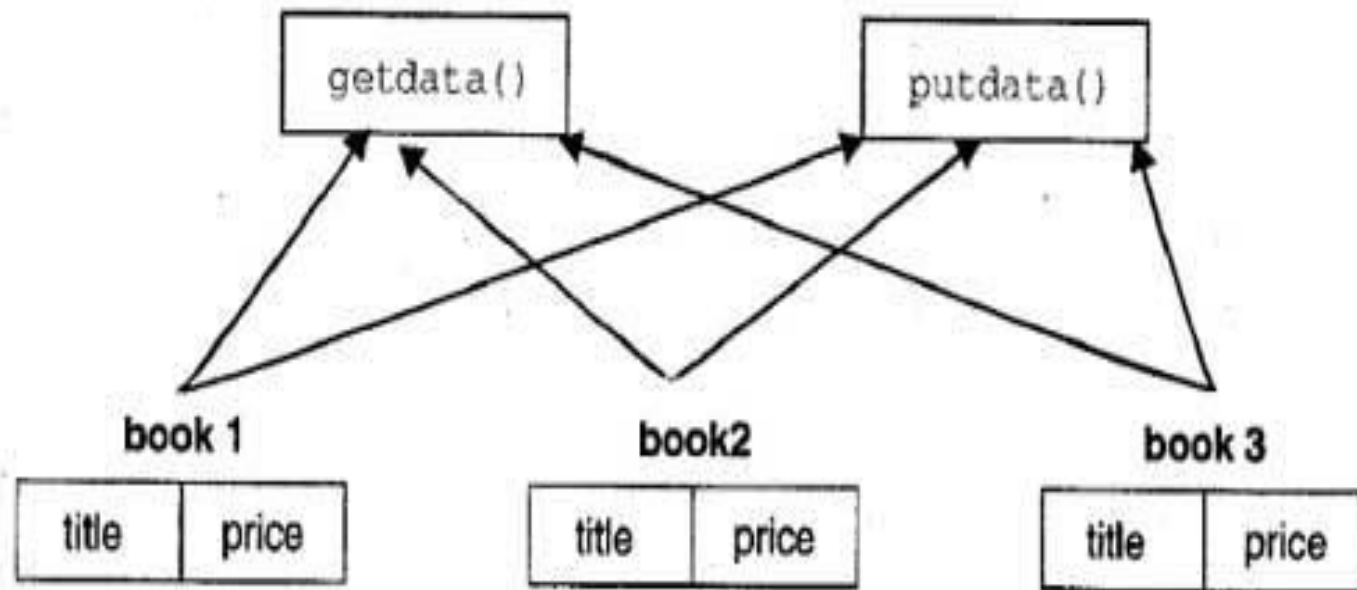
```
{ private:  
    char title[10];  
    float price;  
public:  
    void getdata( )  
    {  
        ....  
    }  
    void putdata( )  
    {  
        ....  
    }  
};
```



```
void main( )  
{  
    book book1,book2,book3;           //3 objects  
    book1.getdata( );  
    book2.getdata( );  
    book3.getdata( );  
    book1.putdata( );  
    book2.putdata( );  
    book3.putdata( );  
}
```



CONTINUE...



Memory Allocation for the Objects of the Class book



2.12 DEFINING MEMBER FUNCTION

- Member function definitions can be defined in two places:
 - Outside the class definition
 - Inside the class definition

❖Outside the class definition :

- 1.We can define Member functions outside the class also.
- 2.Their definitions are very much similar to normal functions.
- 3.They should have a function header and a function body.
- 4.Member functions have a membership label in the header that tells the compiler which class the function belong to.

Syntax of member function definition outside the class:

```
return-type class-name :: function-name(argument list)
{
    function Body
}
```



Example:

```
void student :: getdata ( )  
{  
    cout<< "Enter the no. :";  
    cin>> no;  
    cout<< "Enter Name:";  
    cin>>name;  
}
```



Inside the class definition:

Whenever we replace the function declaration by the actual function definition inside the class then it is called the member function definition inside the class.

Example:

```
class student
```

```
{
```

```
    int no;
```

```
    char name[20];
```

```
public:
```

```
    void getdata( );
```

```
    void putdata( )
```

```
    { ----
```

```
    }
```

```
};
```

When a function is defined inside the class it is treated as an inline function.



2.14 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 functions.

Example:

```
class student
```

```
{    -----
```

```
    public:
```

```
    void putdata()
```

```
    {    -----
```

```
        totalmarks();
```

```
    }
```

```
    void totalmarks()
```

```
    {    -----
```

```
    }
```

```
};
```



2.15 Private Member functions

Normally we define data members as private and function member as public.

But in some cases **we require to declare function as private.**

That private member function is **allowed to access within class only.**

Example, swap function is used as private.


```
int size=5;
class array
{
    private:
        int a[size];
        void swap(int i,int j)                //Private
        function
        {
            int temp;
            temp=a[i];
            a[i]=a[j];
            a[j]=temp;    }
```



Public:

```
void getarray( )
{
    for(int i=0,i<size;i++)
        cin>>a[i];
}

void sort( )
{
    for(int i=0,i<size-1;i++)
    {
        for(int j=i+1;j<size;j++)
        {
            if(a[i]>a[j])
                swap(i,j);
        }
    }
}
```



```
void putarray( )  
{  
    for(int i=0,i<size;i++)  
        cout<<a[i];  
}  
};
```

```
void main( )  
{  
    array a1;  
    a1.getarray();  
    a1.sort();  
    a1.putarray( );  
}
```



2.16 ARRAYS WITHIN A CLASS

Arrays can be declared as the members of a class.

The arrays can be declared as private, public members of the class.

Here in this program class declare the array with size 5,

Program get 5 elements of array and display that element.

```
const int size=5;
```

```
class array
```

```
{    private:
```

```
    int a[size];
```

```
    public:
```

```
        void getarray()
```

```
        {
```

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

```
                cin>>a[i];
```

```
        }
```



```
void putarray( )  
{  
    for(int i=0,i<size;i++)  
        cout<<a[i];  
}
```

```
};
```

```
void main( )
```

```
{
```

```
    array a1;
```

```
    a1.getarray( );
```

```
    a1.putarray( );
```

```
}
```



2.17 STATIC DATA MEMBER AND STATIC MEMBER FUNCTIONS

Characteristics or Advantages:

○ Static data members:

- It is **initialized to zero** when the first object of its class is created.
- **Only one copy** of a static variable is created for entire class
- Static members are **declared inside the class and defined outside the class.**
- It is **visible only** within the class but its **lifetime** is the entire program.
- Syntax:

data-type class-name:: variable-name= value;

where, the value is optional.



○ **Example1:**

```
int item :: count;
```

where, item is the class-name and count is the static data member.

○ **Static member functions:**

- Static member functions are associated with a class, not with any object.
- Invoked using class name
- Access only static members of the class.
- They cannot be virtual.
- They provide encapsulation.
- A static member function does not have this pointer.
- Syntax: **class-name :: function-name;**



Example :

```
#include<iostream.h>
#include<conio.h>
class test
{
    int code;
    static int count;
public:
    void setcode( );
    void showcode( );
    static void showcount( )
    {
        cout<<"Count : "<<count<<endl;
    }
};
int test::count;
```



```
void main( )
```

```
{
```

```
-----
```

```
test::showcount( );
```

```
-----
```

```
test::showcount( );
```

```
-----
```

```
-----
```

```
}
```

VPMP POLYTECHNIC



2.17 ARRAY OF OBJECTS

- The arrays can be used as member variables in a class.
- An array is a collection of same data type or group of data item that stored in a common name.
- **Syntax:**
data type name [size]={list of values};
- Example: `int student[10];`



2.19 PASSING OBJECTS AS AN ARGUMENT, RETURNING OBJECT

- A copy of entire value or object is passed to function, which is called call by value method.
- Only address of objects is transferred to function, which is called by reference or pass by reference.

VPMP POLYTECHNIC



2.20 FRIEND FUNCTION, POINTER TO MEMBERS

- To make an outside function friendly to a class by using **keyword friend**.
- The function is defined elsewhere in the program like a normal C++ function.

Characteristics of the Friend Function:

- It is not in the class to which it has been declared as friend.
- It is not in the scope of the class, it cannot be called using the object of that class.
- It can be invoked like a normal function without the help of the object.
- It can be declared either in the public or private part of a class.
- It has the objects as arguments.



CONTINUE...

- **Advantages:**

- We can access other class members in our class, if we use friend function.
- We can access the members without inheriting the class.

- **Disadvantages:**

- Maximize size of memory occupied by object.
- It breaks the concept of “Data Hiding” of OOP because it access private data.

