**DEPARTMENT OF COMMERCE (CA)**

# OBJECT ORIENTED PROGRAMMING WITH C++ (SEMESTER-IV)

**II-B.COM (CA)** **SUBJECT CODE - 18BCA42C**

## UNIT – II

**Tokens – basic and user – defined data types – operators in C++ - Operator overloading – operator precedence – control structures – decision Making and looping statements – functions in C++ - the main function – Functions prototyping – call by reference – return by reference – inline Functions – functions overloading.**

### Tokens

The smallest individual units in a program are known as tokens. C++ has the following tokens:

- Keywords
- Identifiers
- Constants
- Strings
- Operators.

A C++ program is written using these tokens, white spaces, and the syntax of the language.

### Keywords:

They are explicitly reserved identifiers and cannot be used as program variables or other user-defined program elements.

C++ Keywords

1. char
2. int
3. if
4. then

### Identifiers:

Identifiers refer to the names of variables, functions, arrays, classes etc. created by the programmer. Each language has its own rules for naming these identifiers. They are both common to both C and C++:

- Only alphabetic characters,digits and underscores are permitted.
- The name cannot start with a digit.
- Uppercase and lowercase letters are distinct.
- A declared keywoird cannot be used as a variable name.

A major difference between C and C++ is the limit on the length of a name. While ANSI C recognizes only the first 32 characters in a name, C++ has no limit the length of a name.

**Basic Data Types:**

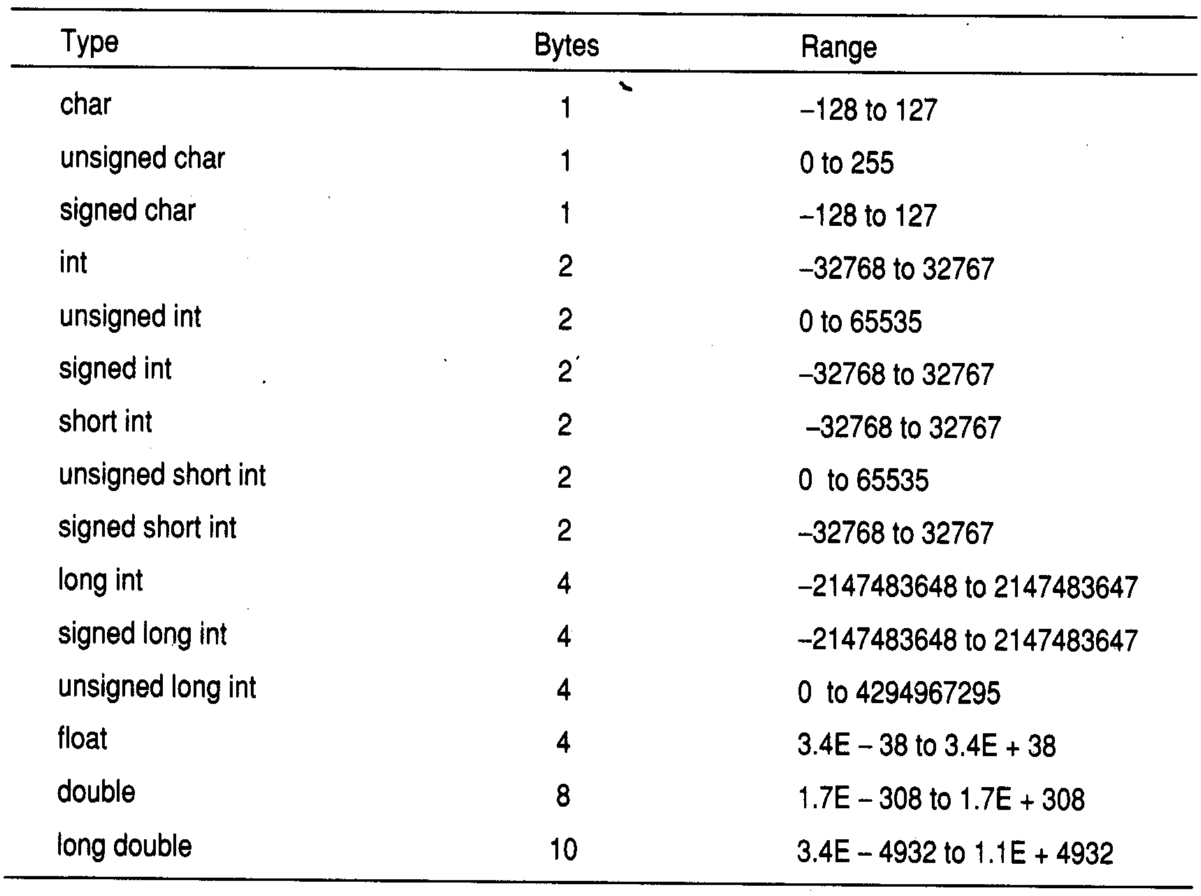
Both C and C++ compilers support all the built in  data types. With the exception of void, the basic data types may have several modifiers preceding then to the serve the needs of various situations. The modifiers signed, unsigned , long, and short may be applied to character and integer basic data types.

Two normal uses of void are

1) to specify the return type of a function when it is not returning any value
2) to indicate an empty argument list to a function.

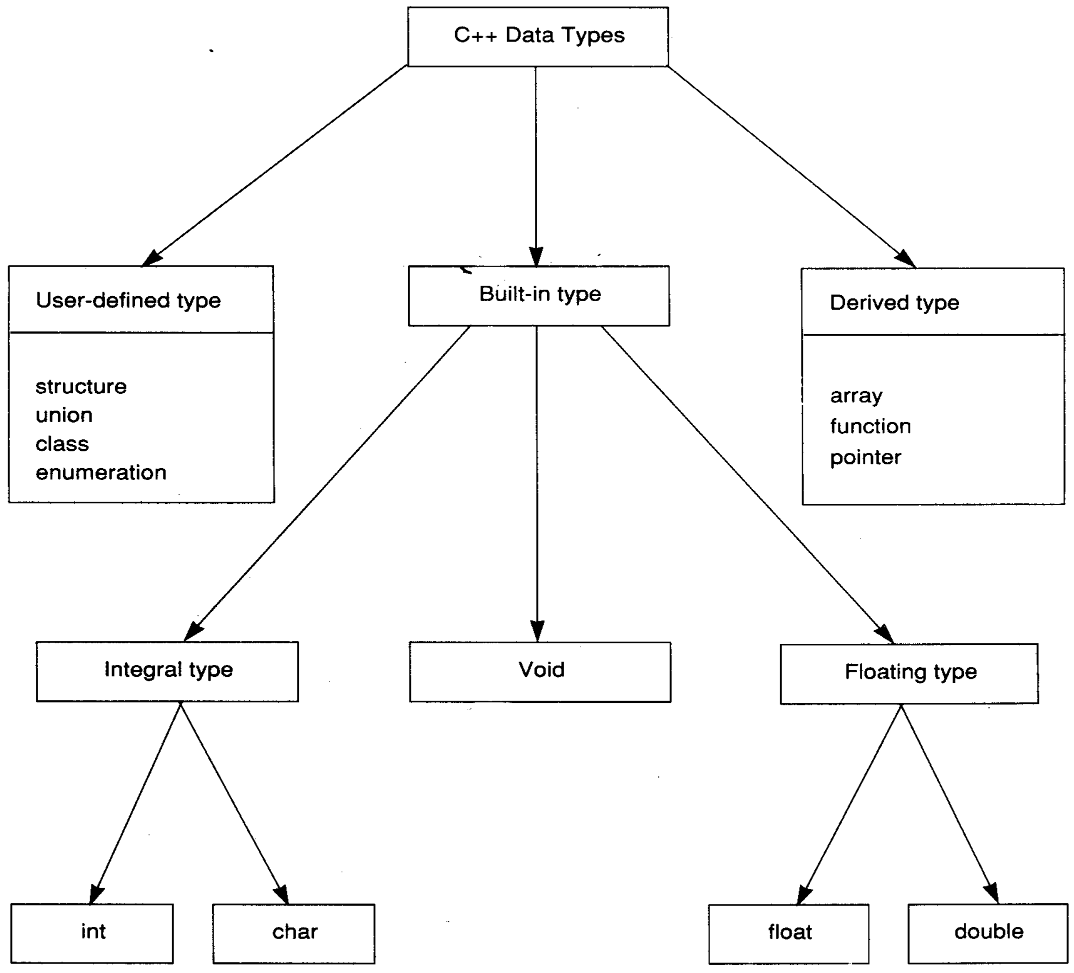
E.g. void fn1(void)

**Size and range of C++ basic data types**



| Type | Bytes | Range |
|---|---|---|
| char | 1 | –128 to 127 |
| unsigned char | 1 | 0 to 255 |
| signed char | 1 | –128 to 127 |
| int | 2 | –32768 to 32767 |
| unsigned int | 2 | 0 to 65535 |
| signed int | 2 | –32768 to 32767 |
| short int | 2 | –32768 to 32767 |
| unsigned short int | 2 | 0 to 65535 |
| signed short int | 2 | –32768 to 32767 |
| long int | 4 | –2147483648 to 2147483647 |
| signed long int | 4 | –2147483648 to 2147483647 |
| unsigned long int | 4 | 0 to 4294967295 |
| float | 4 | 3.4E – 38 to 3.4E + 38 |
| double | 8 | 1.7E – 308 to 1.7E + 308 |
| long double | 10 | 3.4E – 4932 to 1.1E + 4932 |

**Hierarchy of C++ data types**



**User-Defined Data Types**

**Structures and Classes :**

Data types such as struct and union in C are also valid in C++. C++ also permits us to define another user-defined data type known as class. The class variables are known as objects, which are the central focus of object –oriented programming.

Enumerated Data type:An enumerated data type is another user-defined type which provides a way for attaching names to number , thereby increasing comprehensibility of the code. The enum keyword automatically enumerates a list of words by assigning then values(0,1,2,..).

For e.g.

```
enum color(red,blue,green);
color background; // background is of type color
```

**Expressions:**

- An expression is combination of operands(variables) and operators

**Type of expressions :**

| | | |
|---|---|---|
| • constant expression | : | ex: 100 + sum(marks) |
| • integral expression | : | ex:a+b (a and b are integer variables) |
| • float expression | : | ex: x+y (x and y are float variables) |
| • pointer expression | : | ex: & x, prt +1 |
| • relational expression: | | ex: max > (a+b) |
| • logical expression | : | ex: (a >b) \|\| ( c< d) |
| • bitwise operator | : | a>> |

**Manipulators :**

Manipulators are operators that are used to format the data display.
The end1 manipulators, when used in an output statement.
Cout<<"m="<<m<<end1

**Operators in c++:**

All C operators are valid in C++. Additionally C++ supports:

| | |
|---|---|
| << | insertion operator |
| >> | extraction operator |
| :: | scope resolution operator |
| :: * | pointer-to –member declarator |
| --> * | pointer-to-member operator |
| delete | memory release operator |
| new | memory allocation operator |
| endl | line feed operator |
| setw | field width operator |

**Scope resolution operator:**

This operator is used to access global version of variable. Because a variable declared inside a block is said to be local to that block.

```
//   example for scope resolution operator

# include <iostream.h>
int  a=100;                   //  global  variable
main()
{
               int  a=50;
             cout <<"Value of a  is  "<<a<<endl;
             cout << "Value of a is << :: a<< endl;
}

output                    Value of a is  50.
                          Value of a is 100.
```

**Operator overloading in C++:**

**Operator overloading** is a compile-time polymorphism in which the **operator** is **overloaded** to provide the special meaning to the user-defined data type. **Operator overloading** is used to **overload** or redefines most of the **operators** available in **C**++. It is used to perform the operation on the user-defined data type.

**Operator Precedence:**

- C++ enables multiple meanings to the meanings to the operators, their association and precedence remains the same. For example, the multiplication operator will continue having higher precedence than the add operator.
- The precedence and associativity of all the C++ operators are listed below in the order of decreasing precedence.

**Control Structures:**

One method of achieving the objective of an accurate, error –resistant and maintainable code is to use one or any combinations of the following three control structures:

1. Sequence structure (Straight line)
2. Selection structure ( branching)
3. Loop structure (Iteration or repetition).

These structures are implemented using one-entry, one-exit concept .

**The if statement**

The if statement is implemented in two forms:

1) Simple statement

```
if  (conditional exp)
{
        action1;
}
action2;
action3;
```

2) if..else statement.

```
if  (conditional expression)
{
        action1;
}
else
{
        action2;
}
action3;
```

**The switch Statement:**

This is a multiple-branching statement where, based on a condition, the control is transferred to one of the many possible points. This is implemented as follows:

```
switch (expression)
{
   casel:
      {
         action 1;
      }
   case2:
      {
         action2:
      }
   case3:
      {
         actions:
      }
   default:
      {
         action4;
      }
}
action 5;
```

**The do-while Statement**

The do-while is an exit-controlled loop. Based on a condition, the control is transferred back to a particular point in the program. The syntax is as follows:

```
do
{
    action 1;
}
while(condition is true);
action2;
```

**The while Statement**

This is also a loop structure, but is an entry-controlled one. The syntax is as follows:

```
While (condition is true)
{
    action 1;
}
action 2;
```

**The for Statement:**

The for is an entry-entrolled loop and is used when an action is to be repeated for a predetermined number of times. The syntax is as follows:

```
for (intial value;test;increment)
{
        action1;
}
action2;
```

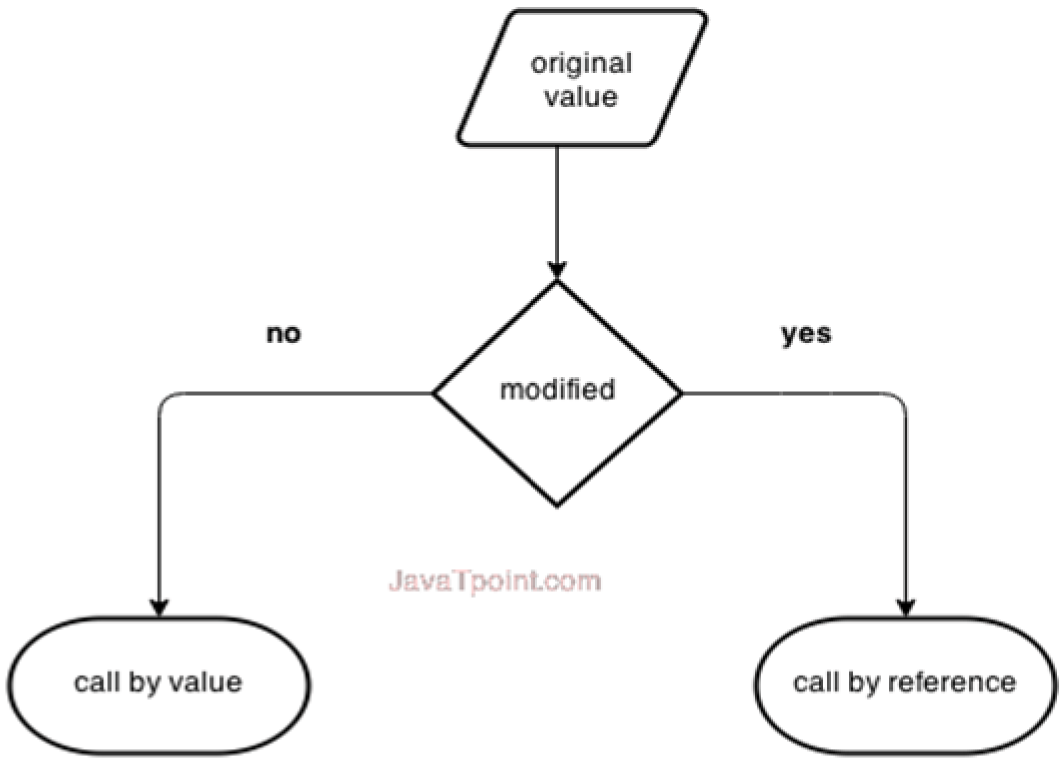
**Functions in c++:**

- A complex problem can be divided into many functions.
- Dividing a program into functions is one of the major principles of top-down, structured programming.
- Functions continue to be the building blocks of C++ program
- In C language, main( ) does not return any value.
- In C++, main( ) returns integer value (zero) to the operating system.

By normal convention , a  function should return 0 (zero) value to indicate successful execution.

**Call by value and call by reference in C++**

There are two ways to pass value or data to function in C language: call by value and call by reference. Original value is not modified in call by value but it is modified in call by reference.



Let's understand call by value and call by reference in C++ language one by one.

---

Call by value in C++

In call by value, **original value is not modified.**

In call by value, value being passed to the function is locally stored by the function parameter in stack memory location. If you change the value of function parameter, it is changed for the current function only. It will not change the value of variable inside the caller method such as main().

Let's try to understand the concept of call by value in C++ language by the example given below:

```
#include <iostream>
```

```
using namespace std;
void change(int data);
int main()
{
int data = 3;
change(data);
cout << "Value of the data is: " << data<< endl;
return 0;
}
void change(int data)
{
data = 5;
}
```

Output:

```
Value of the data is: 3
```

---

Call by reference in C++

In call by reference, original value is modified because we pass reference (address).

Here, address of the value is passed in the function, so actual and formal arguments share the same address space. Hence, value changed inside the function, is reflected inside as well as outside the function.

**Note:** To understand the call by reference, you must have the basic knowledge of pointers.

Let's try to understand the concept of call by reference in C++ language by the example given below:

```
#include<iostream>
using namespace std;
void swap(int *x, int *y)
{
 int swap;
 swap=*x;
 *x=*y;
 *y=swap;
}
int main()
```

```
{
 int x=500, y=100;
 swap(&x, &y);  // passing value to function
 cout<<"Value of x is: "<<x<<endl;
 cout<<"Value of y is: "<<y<<endl;
 return 0;
}
```

Output:

```
Value of x is: 100
Value of y is: 500
```

---

Difference between call by value and call by reference in C++

| No. | Call by value | Call by reference |
|---|---|---|
| 1 | A copy of value is passed to the function | An address of value is passed to the function |
| 2 | Changes made inside the function is not reflected on other functions | Changes made inside the function is reflected outside the function also |
| 3 | Actual and formal arguments will be created in different memory location | Actual and formal arguments will be created in same memory location |

**Static member functions:**

- A static function is a member function of a class and static member function can operate only on static data member of the class.
- The static member function is instance dependent, it can be called directly by using the class name and scope resolution operator. Example:
  Class name : : static function name ;
- If it is declared and defined in a class, the keyword Static should be used only on the declaration part.
- Example:

```
// Program for both static data member and static member function

#include <iosteam.h>
class  sample
{
          private:
                    static int count ;                              // static data  member
          public:
                    sample( );
                    static  void display ( );           // static member function
};

//  static  data definition
int  sample : : count =0;                                      // static data member initialization

sample : : sample ( )
{
          ++ count;
}

//  static member function definition
void sample : : display ( )
{
          cout <<" Counter  value  = " << count <<endl;
}


//  main function begins……….
int main ( )
{
          cout <<"Before creating an object " << endl;
          sample : : display ( );  // calling static member function
          cout <<" After creating objects "<<endl;
          sample obj1,obj2,obj3;
          sample : : display ( );
}
```

output :

```
Before  creating an object
Counter value = 0
After creating objects
Counter value= 3
```

**The main Function**

C does not specify any return type for the main()function which is the starting point the execution of a program.

```
Main(){
{
//main program statements
}
```

**Function Prototyping**:

Function prototyping is one of the major improvements added to C++functions. The prototype describes the function interface to the compiler by giving details such as the number and type of arguments and the type of return values.

Syn:
Type  function –name (argument-list);

**Friend functions:**

C++ allows a mechanism, in which a non-member function has access permissions to the private member of the class. This can be done by declaring the non member function 'friend' to the class whose private data is to be accessed.

- friend is the keyword which is used for this purpose.
- The keyword friend must precede the function declaration whereas function definition must not.
- The function can be defined at any place in the program like normal function.

Syntax for friend function:

```
             class<class_name>
           {
             private:
                   private member variables;
                   private member functions;
             public:
                   public member variables;
                   public member functions;
                   friend return_type function_name( );        //declaring a        member
function as a friend to the class
           };
```

- The friend function has the following properties:
- There is no scope restriction for the friend function; hence can be called directly without using objects.

- Unlike member functions of the class, friend function cannot access the member directly. It uses the object and the dot operator to access the private and public member of the class.
- By default friendship is not shared. Ex: If class X is considered to be the friend of class Y., this doesn't mean that Y has privileges to access private member of the class.
- Use friend functions rarely as it violates the rule of encapsulation and data hiding.
- The function can be declared in either public or the private section without changing its meaning.

Sample Program-10

```
#include<iostream.h>
#include<conio.h>
class acc
{
 private:
        char name[20];
        int accno;
        float bal;
public:
        void read( )
        {
          cout<<‘‘name:”;
          cin>>name
          cout<<‘‘a/c no:”;
          cin>>accno;
          cout<<‘‘balance:”;
          cin>>bal;
         }
friend void showbal(ac);   //friend function declaration
};
void showbal( ac a)
{
cout<<‘‘Balance of acc no.”<<a.accno<<‘‘is Rs.”<<a.bal;
   }
void main( )
{
  ac  k;
  k.read( );
  showbal(k);
}
```

In this pgm, ‘showbal( )’ is the non member function that has the access over the private member i.e., ’name’, ‘accno’, ‘bal’ of the class, just because it has been declared as friend to the class. So access to private member is possible.

**Inline Functions (similar to macro definition and expansion).**

- When a function is called, the control is transferred to the function  and various register contents are saved on the stack.
- When the function is small, a substantial percentage  of execution time may be spent in such overheads.
- To eliminate the overheads for small  function, C ++ supports new function called inline function.
- An inline function is a small function that is expanded in line when it is called (similar to macro definition and macro expansion).
- The compiler substitutes function codes for function calls.

Example :  finding area of various surfaces Refer  class notes.

**Function Overloading in C++:**
You can have multiple definitions for the same function name in the same scope. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list. You cannot overload function declarations that differ only by return type.

**REFERENCE:**

1. E. Balaguruswamy , " Object oriented programming with C++", TataMcGraw Hill publishing company  Limited, 1998.
2. K.R, Venugopal, Rajkumar, T. Ravishankar, "Mastering C++", tata mc graw –  Hill publishing company Limited, 1998.
3. D. Ravichandran, Programming with C++", Tata McGraw – Hill published  Company Limited.

**Prepared By Dr.N.Shanmugavadivu**