2...

Beginning with C++

Objectives...

- To understand Basic Concepts of C++.
- To learn Operators, Namespace, Manipulators, Variables in C++.
- To learn Data types and keywords.
- To study about functions.

2.1 INTRODUCTION

- C++ is a general purpose programming language which has been derived from C programming language.
- C++ could be considered a superset of C language with extensions and improvements with object oriented features included in it. C++ runs on a variety of platforms, such as Windows, Mac OS, UNIX, LINUX etc.
- C++ is regarded as a middle-level language, as it comprises a combination of both high-level and low-level language features.
- C++ is popular languages because of following reasons:

  1. C++ is ideally suited for development of reusable software.
  2. C++ is highly flexible language with versatility.

2.1.1 Tokens

[S-19, W-22]

- The smallest individual units in a program are known as Tokens.
- A token is a group of characters that logically belong together.
- Token is a sequence of characters from the character set of C++, which is identified by the compiler.
- C++ tokens are keywords, literals, identifiers, operators, other symbols (separators) are shown in Fig. 2.1.
- A C++ program or application is written using these tokens, white spaces and the syntax of the language.

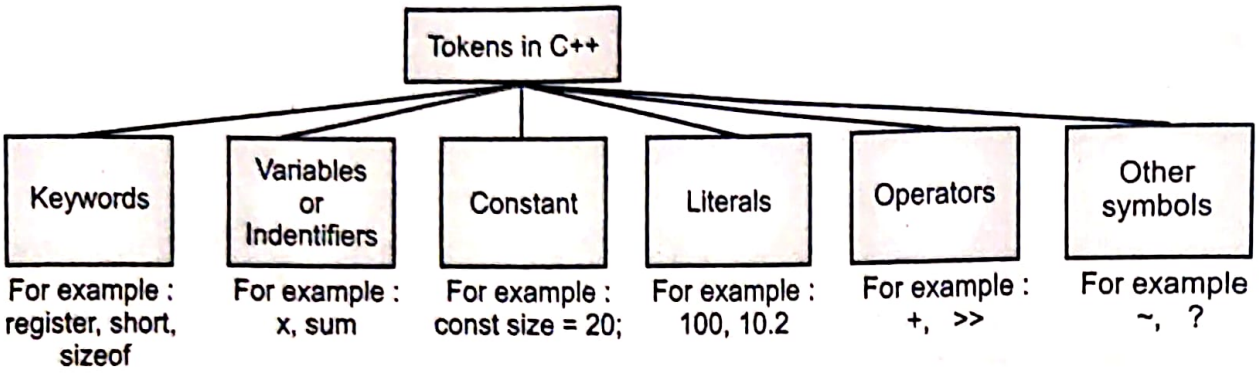


Fig. 2.1: Tokens in C++

2.1.2 Identifiers [W-18]

- An Identifier is any name of variables, functions, classes etc. given by the programmers.
- Identifiers are the fundamental requirement of any language.
- The identifier is a sequence of characters taken from C++ character set.
- The rules for the formation of an identifier are:
  1. An identifier can consist of alphabets, digits and/or underscores.
  2. An identifier must not start with a digit. It start with an alphabet or underscore(_).
  3. C++ is case sensitive that is uppercase and lowercase letters are considered different from each other.
  4. Identifier should not be a reserved word.
  5. Identifier contains maximum 32 characters.
- Some valid identifiers are: `result, x, _n1, acc_no, basicpay`

2.1.3 Constants [W-18]

- Constant refer to fixed values that the program cannot alter or change.
- A constant is an explicit number or character (such as 1, 0.5, or 'c') that does not change. As with variables, every constant has a type.
- A number which does not change its value during execution of a program is known as a constant.
- A constant in C++ can be of any of the basic data types, const qualifier can be used to declare constant as shown below:

```
const float pi = 3.1415;
```

- The above declaration means that pi is a constant of float types having a value 3.1415.
- Examples of valid constant declarations are:

```
const int rate = 50;
const float pi = 3.1415;
const char ch = 'A';
```

- The following types of constants are available in C++:

1. Integer Constants:

- Integer constants are whole number without any fractional part. C++ allows three types of integer constants.

(i) **Decimal Integer Constants:** It consists of sequence of digits and should not begin with 0 (zero).
For example, 124, –179, +108.

(ii) **Octal Integer Constants:** It consists of sequence of digits starting with 0 (zero).
For example, 014, 012.

(iii) **Hexadecimal Integer Constant:** It consists of sequence of digits preceded by ox or OX.

2. Character Constants:

- A character constant in C++ must contain one or more characters and must be enclosed in single quotation marks.
- For example 'A', '9', etc. C++ allows nongraphic characters which cannot be typed directly from keyboard, e.g., backspace, tab, carriage return etc. These characters can be represented by using an escape sequence.
- An escape sequence represents a single character. The following table gives a listing of common escape sequences.

Table 2.1: Escape Sequence

Escape Sequence	Nongraphic Character
\a	Bell (beep)
\n	Newline
\r	Carriage Return
\t	Horizontal tab
\0	Null Character
\\	Blackslash
\?	Quotation mark (?)
\v	Vertical tab
\b	backspace
\'	Single quote (')
\"	Double quote (")

2. Floating Constants:

- Floating Constants are also called real constants.
- They are numbers having fractional parts.
- They may be written in fractional form or exponent form.
- A real constant in fractional form consists of signed or unsigned digits including a decimal point between digits.

For example 3.0, -17.0, -0.627 etc.

2.1.4 Literals

- Literals (often referred to as constants) are data items that never change their value during the execution of the program.

String Literals:

- A sequence of character enclosed within double quotes is called a String Literal.
- String literal is by default (automatically) added with a special character '\0' which denotes the end of the string.
- Therefore, the size of the string is increased by one character.
- For example, "COMPUTER" will re represent as "COMPUTER\0" in the memory and its size is 9 characters.

2.2 KEYWORDS

- The keyword implements specific C++ language features.
- Keywords are explicitly reserved identifiers and cannot be used as name for the program variables.
- It is mandatory that all the keywords should be in lowercase letters.
- The keywords are reserved words, predefined by the language and user cannot change.
- Keywords are used for specific purposes in C++ and compiler can interprets these words.
- Table 2.2 shows the list of keywords which are common in C and C++ language and Table 2.3 shows the keywords specific to C++.

Table 2.2: Keywords common to C and C++

auto	default	float	return	union
break	do	for	shortsigned	unsigned
case	double	goto	sizeof	void
char	else	ifint	staticstruct	volatile
const	enum	long	switch	while
continue	extern	register	typedef	

Table 2.3: Keywords specific to C++

asm	inline	template	using
bitand	new	this	virtual
bitor	namespace	throw	wchar_t
catch	operator	true	
class	private	try	
delete	protected	typeid	
friend	public	typename	

Table 2.4: New Keywords which are supporting new features in C++

bool	false	true
const_cast	mutable	typeid
dynamic_cast	namespace	typename
explicit	reinterpret_cast	using
export	static_cast	wchar_t

2.3 DATA TYPES

2.3.1 Basic Data Types

- In programming, we store the variables in our computer's memory, but the computer has to know what kind of data we want to store in them, since it is not going to occupy the same amount of memory to store a simple number than to store a single letter or a large number. and they are not going to be interpreted the same way.
- The memory in our computers is organized in bytes.
- A byte is the minimum amount of memory that we can manage in C++. A byte can store a relatively small amount of data: one single character or a small integer, (generally an integer between 0 and 255).
- In addition, the computer can manipulate more complex data types that come from grouping several bytes, such as long numbers or non-integer numbers.
- Data types in C++ are classified into different categories is shown in Fig. 2.2.

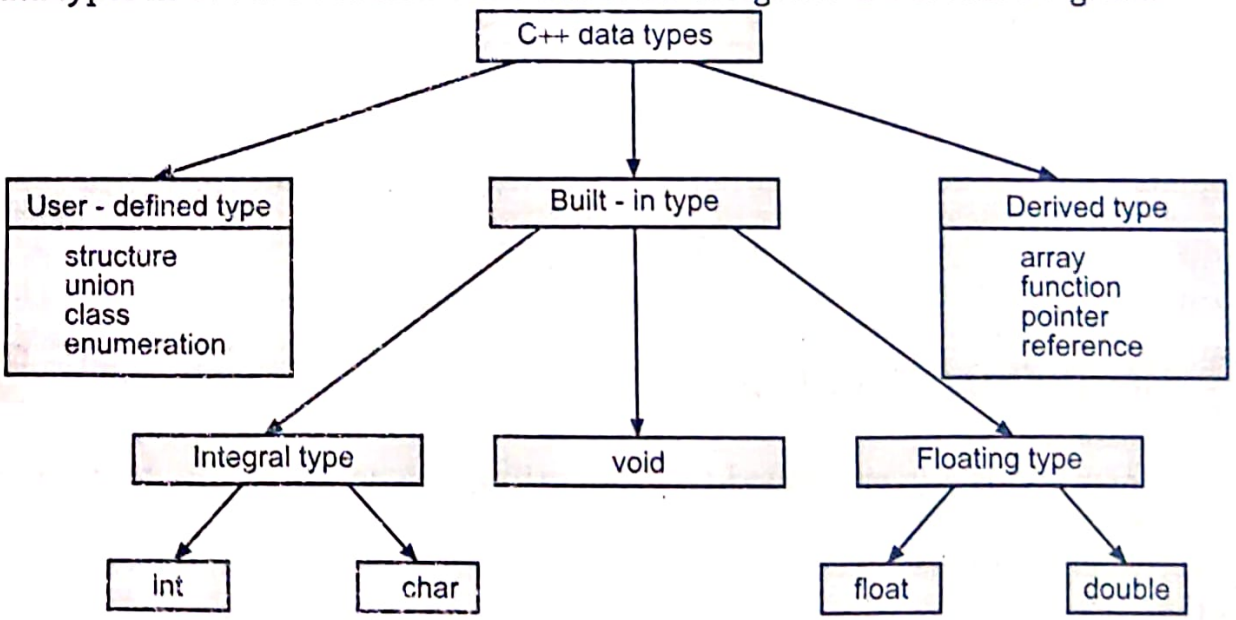


Fig. 2.2: Hierarchy of C++ data types

- Following table list down seven basic C++ data types:

Table 2.5: Basic C++ Data Types

Type	Keyword
Boolean	bool
Character	char
Integer	int
Floating point	float
Double floating point	double
Valueless	void
Wide character	wchar_t

- The following table shows the variable type, how much memory it takes to store the value memory, and what is maximum and minimum value which can be stored in such type of variables.

Table 2.6: Data Types, size and its range

Data Type	Typical Bit Width	Typical Range
char	1 byte	-127 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-127 to 127
int	4 bytes	-2147483648 to 2147483647
unsigned int	4 bytes	0 to 4294967295
signed int	4 bytes	-2147483648 to 2147483647
short int	2 bytes	-32768 to 32767
unsigned short int	Range	0 to 65,535
signed short int	Range	-32768 to 32767
long int	4 bytes	-2,147,483,647 to 2,147,483,647
signed long int	4 bytes	s3ame as long int
unsigned long int	4 bytes	0 to 4,294,967,295
float	4 bytes	+/- 3.4e +/- 38 (~7 digits)
double	8 bytes	+/- 1.7e +/- 308 (~15 digits)
long double	8 bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	2 or 4 bytes	1 wide character

2.3.2 User Defined Data Types [W-22]

1. Enumerations:

- An enumeration is a user defined type consisting of a set of named constant called enumerators.
- Enumerations are a way of defining constants.
- **Syntax:**

```
enum tag {member1, member2 ....., membern};
```

- Here, enum is the keyword and tag is the name of enumeration and member1, member2 ..... are identifiers which represents integer constant values.
- The member should be unique. Once, the enumeration is defined then variable or object of that type is defined.

Syntax:

```
enum tag {member1, member2 .....} variables;
                OR
enum tag {member1, member2 .....} object_name;
```

- Here, all members take integer values starting with zero i.e. member 1 is assigned 0, member 2 is assigned 1 ..... and so on.

For example,

```
enum colors {black, green, red, yellow, blue};
enum colors background;
```

Here,

```
Black = 0
green = 1
red = 2
  .
  .
  .
```

- Enumeration variables are particularly useful as flags to indicate various options are to identify various conditions.
- It increases clarity of the program. If we want to have blue colors for the background, it is easier to understand with,

  ```
  background = blue;
  ```

  than with,

  ```
  background = 4;
  ```
- It makes program more readable.
- Enumerated types are valuable when an object can assume a known and reasonably limited set of values.

2. Structure:

- 'C' structure is a collection of data items of different data types. Structure is a collection of variables under a single name.
- Variables can be of any type such as, int, float, char etc. The main difference between structure and array is that arrays are collections of the same data type and structure is a collection of variables under a single name.

Syntax:

```
struct structure_Name
{
   data_type member 1;
   data_type member 2;
   ..........
};
```

- The structure is declared by using the keyword struct followed by structure name, also called a tag.
- Then the structure members (variables) are defined with their type and variable names inside the open and close braces { }.
- Finally, the closed braces end with a semicolon denoted as; Structure in C++ is define as,

```
struct student
{
   int roll_no;
   char name[20];
};
```

- For example,

```
struct student s;
printf ("%s",  s.name);
```

3. Union:

- A union is a user defined data or class type that, at any given time, contains only one object from its list of members, (although that object can be an array or a class type).

Syntax:

```
union tag
{
   member1;
   member2;
   .
   .
   member n;
} objectname;
```

Example:

```
union example
{
   int a;
   char b;
   float c;
} e;
```

4. Class:

- Class is user define data type with data elements and functions.
- A class in C++ is an encapsulation of data members and functions that manipulate the data.
- The class can also have some other important members which are architecturally important.
- The data members can be of any legal data type, a class type, a struct type etc., they can also be declared as pointers and accessible normally as like other data members.
- **Access Level:** The classes in C++ have three important access levels. They are Private, Public and Protected. The explanations are as follows:

  (i) Private: The members are accessible only by the member functions or friend functions.

  (ii) Protected: These members are accessible by the member functions of the class and the classes which are derived from this class.

  (iii) Public: Accessible by any external member.

- **For example:**

```
class Example_class
{
   private:
         int x; //Data member
         int y; // Data member
   public:
         Example_Class()//Constructor for the C++
         {
            x = 0;
            y = 0;
         };
         ~Example_Class() //destructor for the C++
         { }
         int Add()
         {
            return x+y;
         }
};
```

2.3.3 Derived Data Types in C++

1. Arrays:

- In C, array size is exact same length of the string constant.

  For example: `char string[5] = 'abcde';  //valid in C.`

- But in C++, the array size is one larger than the number of characters present in the string. Here, "\0" is treated as a one character.

  For example:
  ```
  char string[5] = "abcde";   //Invalid in C++
  char string[5] = "abcd";    //Valid in C++
  ```

2. Functions:

- A function is a discrete block of statements that perform certain task.
- Once, a function has been written to play a particular role it can be called upon repeatedly throughout the program.
- Functions are either created by the user or supplied by the system. We can also write our own function according to the requirements.
- A function must be declared before its use, such a declaration is known as a function prototype it ends with semicolon.

  Syntax: `return_type function_name (parameters);`

  For example: `int max (int x, int y);`

3. Pointers:

- A pointer is a variable that represents the location, (rather than the value) of a data item such as a variable or an array element, (i.e. pointers refer *l*-value).

(i) const pointers:

- C++ also has constant pointer declaration and pointer to constant declaration.

For example:

```
int *const m = "xyz";  //constant pointer
int const *m = &l;     //pointer to a constant
```

- In constant pointer we cannot change address of 'm' and in pointer to a constant, the contents of 'l' cannot change.

(ii) void pointer:

- void pointers used to point the other data type of variable in C++. If we define,

```
int *m;
float *l;
m = &l;  //results in compilation error
```

- For such type compatibility void pointers are used.
- **For example,**

```
void *v;
int *i;
float *f;
v = &i;
v = &f;
```

Here, void pointer v will hold pointer of any type.

(iii)void type:

- void is used for two purposes:
  1. If the function is not having any arguments, function_name(void);
  2. To specify the return type of function when function is not returning any value.

```
void function_name (void);
```

2.4 VARIABLES

- A variable is a named location in memory that is used to hold a value that may be modified by the program.
- Variable is the data name that refers to the stored value.
- A variable name is an identifier or a symbolic name assigned to memory location where data is stored.
- A variable provides us with named storage that our programs can manipulate.
- Each variable in C++ has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.
- The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C++ is case-sensitive.

2.4.1 Declaration of Variables

- All variables must be declared before use, although certain declarations can be made implicitly by content.
- A declaration of variable specifies a type, and contains a list of one or more variables of that type as follows:

  Syntax: `type variable_list;`
- Here, type must be a valid C++ data type including char, w_char, int, float, double, bool or any user defined object etc., and variable_list may consist of one or more identifier names separated by commas.
- Some valid declarations are shown here:

```
int i, j, k;
char c, ch;
float f, salary;
double d;
```

- A variable declaration with an initializer is always a definition. This means that storage is allocated for the variable and could be declared as follows:

```
int i = 100;
```

Program 2.1: Program for declaration of variables.

```
#include<iostream>
using namespace std;
int main()
{
   // declaring variables
   int a, b;
   int result;
   // initialization of variable
   a = 5;
   b = 2;
   a = a + 1;
   result = a - b;
   // print out the result
   cout << result;
   // terminate the program
   return 0;
}
```

Output:

```
4
```

Scope of Variables:

- A scope is a region of the program and broadly speaking there are three places where variables can be declared:
  (i) Inside a function or a block which is called local variables.
  (ii) In the definition of function parameters which is called formal parameters.
  (iii) Outside of all functions which is called global variables.
- All the variables that we intend to use in a program must have been declared with its type specifier in an earlier point in the code, like we did in the previous code at the beginning of the body of the function main when we declared that a, b, and result were of type int.
- A variable can be either of global or local scope.
- Variables that are declared inside a function or block are local variables. They can be used only by statements that are inside that function or block of code.
- Local variables are not known to functions outside their own.
- A global variable is a variable declared in the main body of the source code, outside all functions, while a local variable is one declared within the body of a function or a block.

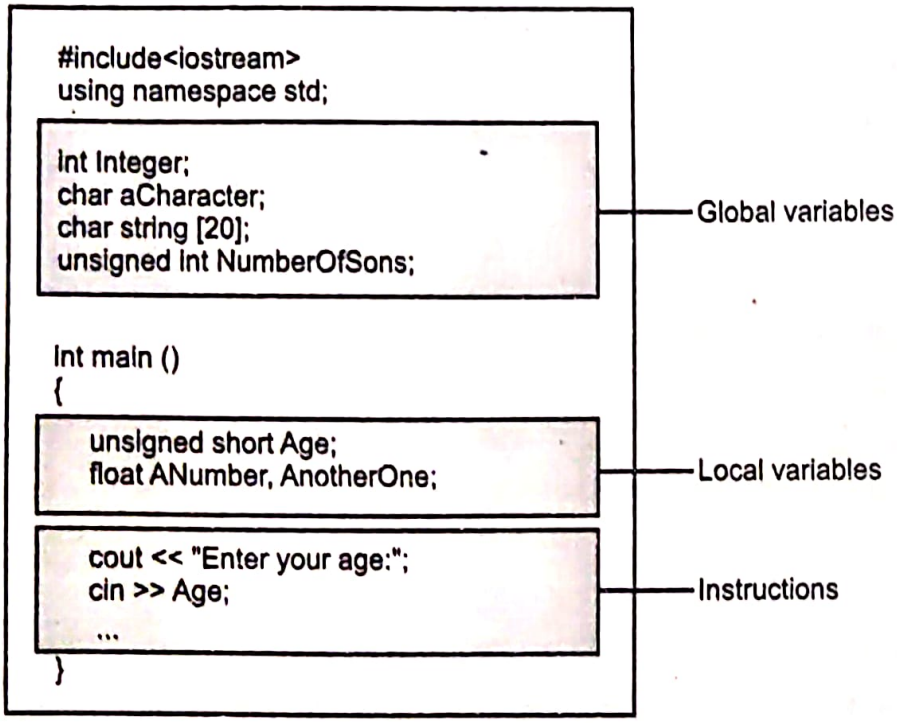


Fig. 2.3: Scope of Variable

- Global variables can be referred from anywhere in the code, even inside functions, whenever it is after its declaration.
- The scope of local variables is limited to the block enclosed in braces ({ }) where they are declared.
- For example, if they are declared at the beginning of the body of a function, (like in function main) their scope is between its declaration point and the end of that function.

- Global variables are defined outside of all the functions, usually on top of the program. The global variables will hold their value throughout the lifetime of your program.
- A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration.

Program 2.2: Program using global and local variables.

```
#include <iostream.h>
//using namespace std;
// Global variable declaration:
int g;
int main()
{
   // Local variable declaration:
   int a, b;
   // actual initialization
   a = 10;
   b = 20;
   g = a + b;
   cout << g;
   return 0;
}
```

Output:

```
30
```

2.4.2 Dynamic Initialization of Variables

- When declaring a regular local variable, its value is by default undetermined. But you may want a variable to store a concrete value at the same moment that it is declared. In order to do that, you can initialize the variable.
- There are two ways to do this in C++:
  (i) The first one, known as C-like initialization, is done by appending an equal sign followed by the value to which the variable will be initialized: `type identifier = initial_value;`

  For example, if we want to declare an int variable called a initialized with a value of 0 at the moment in which it is declared, we could write: `int a = 0;`

  (ii) The other way to initialize variables, known as constructor initialization, is done by enclosing the initial value between parentheses [( )]: `type identifier (initial_value);`

  For example: `int a(0);`
- Both ways of initializing variables are valid and equivalent in C++.

- Some examples are:

```
int d = 3, f = 5;     // initializing d and f.
byte z = 22;          // initializes z.
double pi = 3.14159; // declares an approximation of pi.
char x = 'x';         // the variable x has the value 'x'.
```

Program 2.3: Program for initialization of variables.

```
//initialization of variables
#include<iostream.h>
//using namespace std;
int main()
{
   int a=5;              // initial value = 5
   int b(2);             // initial value = 2
   int result;           // initial value undetermined
   a = a + 3;
   result = a - b;
   cout << result;
   return 0;
}
```

Output:

```
6
```

2.4.3 Reference Variables [S-18, 22]

- A reference variable is an alias or an alternative name for an object to which it is referring.
- It is a variable provides an alias (alternative name) for a previously defined variable.
- A reference variable is created as,

```
data_type &reference_name = variable_name;
```

- C++ references allows you to create a second name for the variable that you can use to read or modify the original data stored in that variable.
- While this may not sound appealing at first, what this means is that when you declare a reference and assign it a variable, it will allows you to treat the reference exactly as though it were the original variable for the purpose of accessing and modifying the value of the original variable even if the second name (the reference) is located within a different scope.
- This means, for instance, that if you make your function arguments references, and you will effectively have a way to change the original data passed into the function.
- This is quite different from how C++ normally works, where you have arguments to a function copied into new variables.
- Reference variables also allows you to dramatically reduce the amount of copying that takes place behind the scenes, both with functions and in other areas of C++, like catch clauses.

- For example:

```
int a = 20;
int &b = a;  // & is a reference operator
2132   ← Address
[20]   ← Value
a      ← Variable name
b      ← reference to a
```

- Manipulating a reference to an object allows manipulation of the object itself. This is because no separate memory is allocated for a reference.
- A reference to a variable is created using the address of operator (&). When the & operator is used in the declaration, it becomes the reference operator.
- A reference variable is declared as follows:

```
datatype &refvar = variable;
```

where, refvar is name of reference variable.

- In above example, a is an integer variable whose address (*l*-value) is 2132 which is intialize to 20. In the second statement, 'b' is a reference to 'a'. If we manipulate the value of b (say 10), then the vlaue of the memory location 2132 gets changed or in other words the value of 'a' changes.

Program 2.4: Program for reference variable.

```
#include<iostream.h>
//using namespace std;
int main()
{
   int p = 50;
   int *a = &p;
   int &b = *a;
   cout<<"*a="<<*a<<"b="<<b<<endl;
   int x = 10;
   int &y = x;
   x = x + 10;
   cout<<"x="<<x<<"y="<<y<<endl;
   return 0;
}
```

Output:

```
*a=50 b=50
x=20 y=20
```

2.5 OPERATORS IN C++

- Operators are special symbols used for specific purposes.

1. Arithmetical Operators:

Arithmetical operators +, -, *, /, and % are used to performs an arithmetic (numeric) operation. You can use the operators +, -, *, and / with both integral and floating-point data types. Modulus or remainder % operator is used only with the integral data type. Operators that have two operands are called binary operators.

Table 2.7: Arithmetic Operators

Operators	Meaning
+	Addition
–	Subtraction
*	Multiplication
/	Division
%	Modulus

2. Relational Operators:

The relational operators are used to test the relation between two values. All relational operators are binary operators and therefore require two operands. A relational expression returns zero when the relation is false and a non-zero when it is true. The following table shows the relational operators.

Table 2.8: Relational Operators

Relational Operators	Meaning
<	Less than
<=	Less than or equal to
==	Equal to
>	Greater than
>=	Greater than or equal to
! =	Not equal to

3. Logical Operators:

The logical operators are used to combine one or more relational expression. The logical operators are:

Table 2.9: Logical Operators

Operators	Meaning
||	OR
&&	AND
!	NOT

4. **Unary Operators:**

C++ provides two unary operators for which only one variable is required.

For example, a = – 50;

a = + 50;

Here plus sign (+) and minus sign (-) are unary because they are not used between two variables.

5. **Assignment Operator:**

The assignment operator '=' is used for assigning a variable to a value. This operator takes the expression on its right-hand-side and places it into the variable on its left-hand-side.

For example: m = 5;

The operator takes the expression on the right, 5, and stores it in the variable on the left, m.

x = y = z = 32;

This code stores the value 32 in each of the three variables x, y, and z.

In addition to standard assignment operator shown above, C++ also supports compound assignment operators.

Table 2.10: Assignment Operators

Operators	Example	Equivalent to	Description
+ =	A + = 2	A = A + 2	Addition and Assignment
- =	A - = 2	A = A - 2	Subtraction and Assignment
% =	A % = 2	A = A % 2	Modulus and Assignment
/=	A/ = 2	A = A / 2	Division and Assignment
*=	A * = 2	A = A * 2	Multiplication and Assignment

6. **Increment and Decrement Operators:**

C++ provides two special operators '++' and '--' for incrementing and decrementing the value of a variable by 1.

The increment/decrement operator can be used with any type of variable but it cannot be used with any constant.

Increment and decrement operators each have two forms, pre and post.

The syntax of the increment operator is:

Pre-increment: `++variable`

Post-increment: `variable++`

The syntax of the decrement operator is:

Pre-decrement: `-- variable`

Post-decrement: `variable --`

In Prefix form first variable is first incremented/decremented, then evaluated.

In Postfix form first variable is first evaluated, then incremented/decremented.

```
int x,y;
int i=10,j=10;
x = ++i;      //add one to i, store the result back in x
y= j++;       //store the value of j to y then add one to j
cout<<x;      //11
cout<<y;      //10
```

7. **Conditional Operator:**

The conditional operator?: is called ternary operator as it requires three operands.

Syntax:

```
Conditional_ expression? expression1: expression2;
```

If the value of conditional expression is true then the expression1 is evaluated, otherwise expression2 is evaluated.

```
int a = 5, b = 6;
big = (a > b)? a: b;
```

The condition evaluates to false, therefore biggets the value from b and it becomes 6.

8. **Comma Operator:**

The comma operator gives left to right evaluation of expressions.

When the set of expressions has to be evaluated for a value, only the rightmost expression is considered.

```
int a=1, b=2, c=3, i;  // comma acts as separator, not as an operator
i = (a, b);            // stores b into i
```

Would first assign the value of a to i, and then assign value of b to variable i. So, at the end, variable i would contain the value 2.

9. **New C++ Operators:** C++ contains some new additional operators they are listed below:

Table 2.11: New C++ Operators

Operators	Meaning
new	Memory allocation operator
delete	Memory release operator
endl	Line feed operator
::	Scope resolution operator
	Pointer to member declarator
	Pointer to member operator
	Pointer to member operator
	Field width operator

2.5.1 Scope Resolution Operator [S-22, 23; W-18, W-22]

- The scope resolution operator is denoted by a pair of colons (::).
- It is use to access global variable even if the local variable has the same name as global.
- When the same function name is used for many classes, then this complexity is solved by using scope resolution operator to indicate the class with which the function is associated.
- This operator can be used to uncover a hidden variable.
- **Syntax:**

```
::variable_name
```

- Suppose in a C program, there are two variables with the same name a. One is global, (outside a function) and other is local, (inside a function).
- We attempt to access the variable a in the function, we always access the local variable (priority to local).
- C++ allows the flexibility of accessing both the variables using scope resolution operator (::).

Program 2.5: Simple scope resolution operator.

```
#include<iostream.h>
int a=10;
int main()
{
   int a=15;
   cout<<"\n Local a="<<a<<"Global a="<<::a;
   ::a=20;
   cout<<"\n Local a="<<a<<"Global a="<<::a;
   return 0;
}
```

Output:

```
Local a=15 Global a=10
Local a=15 Global a=20
```

- In the following example, the declaration of the variable X hides the class type X, but you can still use the static class member count by qualifying it with the class type X and the scope resolution operator.

```
#include<iostream.h>
// using namespace std;
class X
{
public:
static int count;
};
```

```
int X::count = 10;              // define static data member
int main ()
{
    int X = 0;                  // hides class type X
    cout<<X::count<<endl;       // use static member of class X
    return 0;
}
```

- There are two uses of the scope resolution operator in C++.
  1. The first use being that a scope resolution operator is used to unhide the global variable that might have got hidden by the local variables. Hence, in order to access the hidden global variable one needs to prefix the variable name with the scope resolution operator (::).

     For example:

```
int i = 10;
int main()
{
  int i = 20;
  cout << i;    // this prints the value 20
  cout <<::i;   // in order to use the global i one needs to prefix it
                   with the scope resolution operator.
}
```

  2. The second use of the operator is used to access the members declared in class scope. Whenever, a scope resolution operator is used the name of the member that follows the operator is looked up in the scope of the class with the name that appears before the operator.

Program 2.6: Use of scope resolution operator for class.

```
#include<iostream.h>
class T
{
   int a,b;
   public:   void getdata();
             void putdata();
};
void T::getdata()
{
   cin>>a>>b;
}
void T::putdata()
{
   cout<<"a="<<a<<endl;
   cout<<"b="<<b<<endl;
}
```

```
int main()
{
    T a;
    a.getdata();
    a.putdata();
    return 0;
}
```

Output:

```
10 20
a=10
b=20
```

2.5.2 Memory Management Operator [S-22; W-18]

- For dynamic memory allocation, C++ provides new operator. In C, we have malloc( ) and calloc( ) functions for dynamic allocation.
- For dynamic deallocation, C++ provides delete( ) operator, which is same as free( ) function in C.

1. **new operator:** This operator is used for allocation of memory. [W-22]

   Syntax:

   ```
   ptr_var = new datatype;
   ```

   For example,

   ```
   char *P;
   P = new char[5];    // reserves 5 bytes of contiguous memory of
                          characters
   int *x;
   x =  new int;    // allocate 2 bytes for integer
   ```

2. **Delete operator:** The delete operator deallocates or frees the memory allocated by new.

   Syntax:

   ```
   delete ptr_var;
   ```

   The ptr_var is the pointer to a data object which is created by new.

   For example,
   ```
   delete P;
   delete x;
   ```

   - If the array is allocated by new, then to free it we use the following syntax,
     ```
     delete [size] ptr_var;
     ```
   - If the size is not specified then entire array get deleted.

     For example, `delete [ ] a;`

Program 2.7: Program for new and delete operators.

```
#include<iostream.h>
#include<string.h>
```

```
int main()
{
    char *x;
    x = new char[10];
    strcpy(x, "COMPUTER");
    cout<<"\n the string is:"<<x<<endl;
    delete x;
    return 0;
}
```

Output:

```
the string is: COMPUTER
```

Note:

- For malloc( ) we use **sizeof** operator, for new "sizeof" is not required, it automatically compute the size.
- The operators new and delete are overloaded.
- Type casting is not required since new operator automatically returns the correct pointer type.

2.6 MANIPULATORS [S-22, 23; W-22]

- Manipulators are the functions used in input/output statement.
- All the manipulators are defined in header field **iomanip.h.**
- We can use chain of manipulator that is one or more manipulators in statement.

Table 2.12: Manipulators

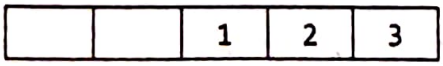
Manipulators	Equivalent I/O function
setw()	width()
setprecision()	precision()
setfill()	fill()
setioflags()	setf()
resetioflags()	unsetf()

1. Setting of field width:

- setw() is used to define the width of field necessary for the output for a variable.
- For example: `setw(5);` This function will reserve 5 digits for a number.

```
setw(5);
cout << 123;
```

Output:



2. Setting precision for float nos:

- We can control number of digits to be displayed after decimal point for float numbers by using setprecision() function.

```
setprecision(2);
```

- This function will display only two digits after a decimal point.
- For example:
```
setprecision(2);
cout<<3.14159;
```
Output: 3.14

3. Filling of unused positions:

- Filling of unused positions of the field can be done by using setfill().
```
setfill('*');
```
- Here, unused positions will be filled by using * sign.
```
setw(5);
setfill ('$');
cout << 123;
```
Output:

$	$	1	2	3

4. setbase():

- It is used to show the base of a number, for example:
```
setbase(10);
```
- It shows that all numbers are decimal numbers.

5. Flags:

- C++ defines some format flags for standard input and output, which can be manipulated with the flags(), setf() and unsetf() functions.
- The flags() function either return the format flags for the current stream or sets the flags for the current stream to be f.

Syntax:
```
fmtflags flags();
fmtflags flags(fmtflags f);
```

6. setf() and unsetf(): [S-19]

- setf() function can also be used with only one argument. Arguments will be as follows:

Table 2.13: Arguments

Arguments	Use
ios:: showbase	It helps to use base indicator an output
ios:: uppercase	It helps to use uppercase letters.
ios:: skipws	It skips (blank) white spaces from input
ios:: showpoint	It helps to display trailing zeros.
ios:: showpos	It helps to display + sign for number.

- The function unsetf( ) is used to clear the given flags associated with the current system.

Syntax:
```
void unsetf(fmtflags flags);
```

7. endl:

- This manipulator used in an output statement, causes a linefeed to be inserted. It has the same effect as using the new line character "\n".

For example:

```
cout <<"a=" << a << endl
     <<"b=" << b << endl
     <<"c=" << c << endl;
```

If we assume value of variable as 2215, 181 and 17 the output will:

a = 2 2 1 5

b = 1 8 1

c = 1 7

2.7 FUNCTIONS

- Functions are the basic building blocks of C++ and the place where all program activity occurs. A function is a group of statements that together perform a task.
- A function is a subprogram that acts on the program data and often returns values.
- A program written with numerous functions is easier to maintain, update and debug than one very long or large program.
- Functions are used to implements large programs. In structured programming functions are used to break the larger programs into smaller ones; which makes debugging of program easier.
- Each function has its own name. When that name is encountered in a program, the execution of the program branches to the body of that function.

Features of Functions:

1. The structure of a function definition is look like the structure of main(), with its own list of variables declaration and program statements.
2. A function can have a list of zero or more parameters inside its brackets, each of which has a separate type.
3. A function may have more than one "return" statement in which case the function definition will end execution as soon as the first "return" is reached.
4. Function declarations are like variable declaration, (functions specify which return type the function will return).
5. A function has to be declared in a function declaration at the top of the program and before it can be called by main().

Advantages of Functions:

1. To reduce the size of a program by calling them at different places in the program.
2. Functions can be compiled separately.
3. Using functions, programs are easier to design, debug and maintain.
4. Reuse of the function is possible in multiple programs.
5. The functions provide interfaces to communicate with object.

How a Function Works?

- A C++ program does not execute the statement in a function until the function is called or invoked.
- When the C++ program's function is called or invoked, the control passes to the function and returns back to the calling part after the execution of function is over or exit.
- The calling program can send information to the functions in the form of argument. An argument of function stores data needed by the function to perform its task.
- Following program shows working of functions:

Program 2.8: Program for working of function.

```
#include<iostream>
using namespace std;
int add(int x, int y)  //declaration of function
{
    int z;
    z = x + y;
    return(z);
}
int main()
{
    int a;
    a = add(10, 20);  //calling a function
    cout<<"The result is" << a;
    return 0;
}
```

Output:

```
The result is 30
```

- In above program, we declare the variable a of type in main() then a call to a function named add().
- We will able to see the similarity between the structure of the call to the function and the declaration of the function itself in the code lines below.

```
int add(int x, int y)
          ↑      ↑
a = add(10,    20);
```

- Within the main() we called the add(), passing 10, 20 values that corresponds to the int x and int y parameters declared for the add().
- add() declares a new variable int z and by means of the expression z = x + y; it assigns to z the result of x plus y, because the passing parameters for x and y are 10, 20 respectively the result in 30.
- The function has elements like: 1. A function declaration, 2. A function definition, 3. A function call, and 4. Returning from function. Let us see in detail.

1. A Function Declaration:

- A function declaration tells the compiler about a function's name, return type and parameters. This is also referred as Function Prototyping.
- A function is declared with a prototype, which consists of return type, a function name and argument list.
- **Syntax for function declaration:**

```
return_type function_name(argument1, argument2, ...)
```

- A function can be declared either in the main function or in a class.
- For example,

```
int add (int a, int b);     //function add with 2 arguments
void func(void);            //function prototype with no arguments
float display();            //function prototype with no arguments
```

Program 2.9: Program illustrates the function declaration.

```
#include<iostream.h>
using namespace std;
int main()
{
   int a, b,c;
   //function declaration
   cout<<"Enter two numbers \n";
   cin>>a>>b;
   int add (int, int) ;
   cout<<"addition is"<<c;
}
//function definition
int add(int x, int y)          //function header
{
   int sum;
   sum = x + y;
   return sum;
}
```

Output:

```
Enter two numbers
  5       6
addition is 11
```

2. Function Definition:

- The function definition tells the compiler what task the function will be performed.
- Function definition consists of function prototype and actual body of a function. It can be written anywhere in C++ program.
- The function definition consists of two parts:
  (i) Function header or Prototype, and
  (ii) Function body.

- The function header is similar to declaration. The function header is a replica of function declaration. The basic difference between these two is that the declaration in the calling function will end with semicolon whereas, in the called function it is not.
- The function body consists of local variables (if needed) and simple or compound statements.
- **Syntax:**

```
return_type function_name(argument list)
{
    local_variables declaration;
    statements;
}
```

- Here, argument list consists of any number of arguments with any data type or no arguments. The return_type is the type of return value from the function. If we are using return statement, then return_type should be specified otherwise void is used as return_type.
- For example,

```
int add (int a, int b)  //function header
{
    int total           //local variable
    total = a + b;
    return total;              //function body
}
```

Program 2.10: Program for function definition.

```
#include<iostream>
using namespace std;
int mult(int a, int b)
{
   int prod;
   prod = a * b;
   return(prod);
}
int main()
{
   int x = 3, y = 4, z;
   z = mult(5, 4);
   cout<<"The first product is="<<z<<"\n";
   cout<<"The second product is="<<mult(x,y)<<"\n";
   cout<<"The third product is="<<mult(5,4);
   return 0;
}
```

Output:

```
The first product is=20
The second product is=12
The third product is=20
```

3. A Function Call:

- Function can be called directly by simply writing the function name with arguments or indirectly in the expression if it returns a value.
- We have to pass the actual parameters (arguments) to the functions. In the function definition, the arguments are referenced. Therefore, calling a function is also known as Function Reference.
- The arguments which we pass should match with the formal parameters which are declared into the function. The data type and the order should also be matched.
- For example,

```
(i) add (x, y);              //function add called
(ii) display( );             //function display called
(iii) larger = max(a,b);     //function max called
(iv) total = add(10, 20)     //function add called
```

4. Returning from Function:

- To return from function we use return statement.
- *return* statement pushes a value onto the return stack and control is return back to the calling function.
- If return type is void, the function does not return any value.

Syntax:

```
return;  OR
return value;
```

Program 2.11: Program for return statement.

```
/* function void */
#include<iostream>
using namespace std;
void func(void)
{
 cout<<"This is void function";
}
int main()
{
 func();
}
```

Output:

```
This is void function
```

2.8 FUNCTION PROTOTYPING [W-18]

- One of the most important features of C++ is the function prototype.
- A function prototype is a declaration that delivers the return type and the parameters of a function.
- A function is declared with a prototype.

- The function prototype tells the compiler the name of the function, the type of data returned by the function, the number of parameters the function expects to receive, the types of the parameters and the order in which these parameters are expected.
- The compiler uses function prototype to validate function calls.
- The function prototype is also called the function declaration.
- Functions are declared similar to variables, but they enclose their arguments in parenthesis [()].
- The function prototype describes the function interface to the compiler by giving details like the numbers and type of arguments and the type of return values.
- **Syntax:** `return_type function_name (list of parameters);`
- **For examples:**
  1. `int add(x, y); //Declaration of add with x, y arguments.`
  2. `int max( );  //Declaration of max with no argument.`

2.9 INTERACTION BETWEEN FUNCTIONS

- The function [main() and other] interacts with each other through parameters. We pass the parameter to the function for the communication of calling function and the called function.
- There are two types of parameters actual parameters (used in the function call) and formal parameters (used in function definition).

2.9.1 Call by Reference [S-18, W-22]

- When a function is called the address of the actual parameters are copied on to the formal parameters, though they may be referred by different variable names.
- This content of the variables that are altered within the function block are return to the calling function program. As the formal and the actual arguments are referencing the same memory location or address. This is known as call by reference, when we use this concept is functions.
  - Called function makes the alias of the actual variable which is passed.
  - Actual variables reflect the changes made on the alias of actual variables.
  - Number of values can be modified together.
  - For call by reference concept C++ uses a reference operator (&).
  - & operator allows true call by reference functions, eliminating the need to dereference arguments passed to the function.
  - The & operator is placed before the variable name of argument in the parameter list of the function. Then the function can be called without passing the address and without dereferencing within the function.

Program 2.12: Program to swap the values by call by reference.

```
#include<iostream.h>
void swapr (int &p, int &q)
{
    int dummy;
    dummy = p;
    p = q;
    q = dummy;
}
```

```
int main()
{
    int x = 225, y = 305;
    cout<<x<<"\t"<<y<<endl;
    swapr(x,y);
    cout<<"After swap"<<endl;
    cout<<x<<"\t"<<y;
```

```
305
225
```

- Let us see how this program gets executed in the memory. This is shown by Fig. 2.4.
- In the Fig. 2.4, p is alias of x and q is alias of y. Therefore, we do not need 4 bytes allocated for p and q, [each will have 2 bytes].
- Here, it first copies the value of x i.e. p to dummy. Then the value of q or y is assigned to P and then the value of dummy is assigned to y or q. In this way the values are swapped without wasting more amount of memory. This is because the formal parameters are of reference type.
- The compiler always treats them similar to pointers but compiler does not allow the modification of the pointer value. The changes made to the formal parameters p and q are reflected on the actual parameters x and y as shown in Fig. 2.4.
- The following points can be noted about reference parameters:

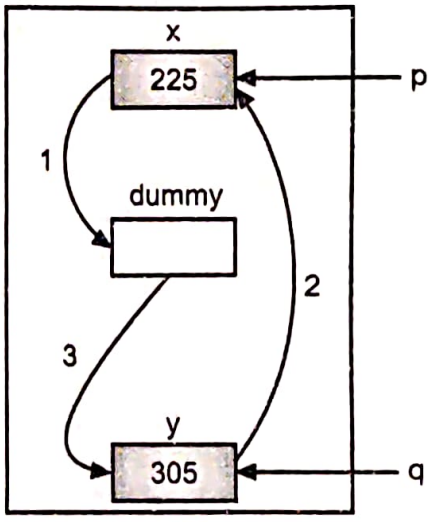


Fig. 2.4: Call by Reference

1. Here, parameters passed to a function by its reference and not by its value. Reference means address of variable is passed to function.
2. A reference cannot be null.
3. It should always refer to a object or a variable.
4. Once, a reference points towards one object or variable, then it cannot be changed. Even it should not point towards different object.
5. The reference does not require any explicit mechanism to dereference the memory. Whenever the program execution terminates, all the memory locations gets free. References are also lost.
6. It is good for performance because it eliminates the overhead of copy large amount of data.

2.9.2 Call by Value [W-22]

- In call by value, we called function makes the copy of actual variable passed.
- The actual variables do not reflect the changes made on the copies of actual variables. In call by value only one value can be returned.

Program 2.13: Program for call by value.

```
#include<iostream.h>
void swap (int &p, int &q)
{
   int dummy;
   dummy = p;
   p = q;
   q = dummy;
}
int main()
{
   int x = 225, y = 305;
   cout<<x<<"\t"<<y<<endl;
   swap(x,y);
   cout<<"After swap"<<endl;
   cout<<x<<"\t"<<y;
}
```

Output:

```
225      305
After swap
305      225
```

Explanation of Program 4.6:

- Here, we have used three variables of type integer p, q and dummy. Hence, we required 3 memory location each of 2 bytes. Therefore, 3 * 2 = 6 bytes are allocated for swapping the values of two variables as shown in Fig. 2.5.
- Fig. 2.5 specifies the steps with numbers:
  1. value x is passed to p.
  2. value y is passed to q.
  3. value p is passed to dummy.
  4. value q is assigned to p.
  5. value of dummy assigned to q.
- The above program has not return any values. The return value is actually the result of computation in the called functions.

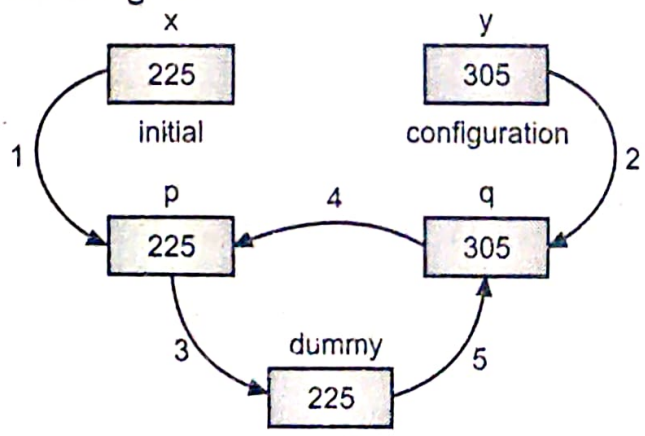


Fig. 2.5: Call by Value

- The returned value is stored in a data type in the functions. We can also return the expressions. If return statement is not present into the function, it means that the return type is **void**.
- We know that there is not restriction in passing any number of values to a function. But for return value there is a restriction. So we can pass whole array to a function but it can return single value only. The program 2.14 explain this concept well.

Program 2.14: Program to find largest no in an array.

```
#include<iostream.h>
int main()
{
   int a[] = {5, -2, 82, 15, 24, 9};
   int s = 5, max;
   int maxi (int a[], int s);    //prototype;
   max = maxi(a,s);
   cout<<max;
}
int maxi (int p[], int s)
{
   int i, maxp = 0;
   for (i=0;i<s;i++)
   {
         if (p[i]>maxp)
         {maxp = p[i];}
   }
   return maxp;
}
```

Output:

```
82
```

2.9.3 Return by Reference [S-19]

- The function can also return the reference of variable. This reference variable is actually an alias for the referred variable.
- The method of returning reference is used generally in operator overloading. This method will form a cascade of member function calls.

  For example: `cout << k << a;`
- This statement is a set of cascaded calls which returns reference to the object cout.

Program 2.15: Program for function which returns value by reference.

```
#include<iostream.h>
int &Large (int &m, int &n);   //function prototype
int main()
{
   int a,b,m,n;
```

```
    cout<<"Enter values for two integers";
    cin>>a>>b;
    //the function call is given
    //the reference which is returned is assigned to - 1
    Large (m, n) = - 1;
    cout<<"First number"<<m;
    cout<<"second number"<<n;
}
int &Large (int &p, int &q)
{
    if (p>q)
    {return p;}  //function returns the value through reference
    else
    {return q;}
}
```

Output:

```
Enter values for two integers     10     20
First number     10
Second number     - 1
If we executive the program one more time then output will be:
Enter values for two integers       8      3
First number     - 1
Second number     3
```

- When the program gets executed, it returns the reference to the variable which holds the larger value. And this assigns the value - 1 to it.

2.10 INLINE FUNCTIONS [S-18, 19, 22; W-18]

- C++ inline function is powerful concept that is commonly used with classes. If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.
- Any change to an inline function could require all clients of the function to be recompiled because compiler would need to replace all the code once again otherwise it will continue with old functionality.
- The advantage of using function is to save the memory space. Consider a situation where a small function is called number of times calling a function requires certain overheads such as:
  - Passing to the function,
  - Passing the values or push arguments into the stack,
  - Save the registers,
  - Return back to calling function.

- In such a situation (if small program called number of times), execution time requires more. For the above problem, C programming language provides macros. But macros are called at execution time, so usual error checking does not occur during compilation.
- For the above situation, C++ compiler put the code directly inside the function body of calling program. Every time when function is called, at each place the actual code from the function would be inserted, instead of a jump to the function. Such functions are called inline functions.
- A function can be defined as an inline functions by writing the keyword inline to the function header as shown below:

```
inline function_name (list of arguments)
{
   //function body
}
```

- When we do not specify keyword inline then only one copy of function code (object code) is called depending upon the functions calls. This is illustrated in Fig. 2.6.

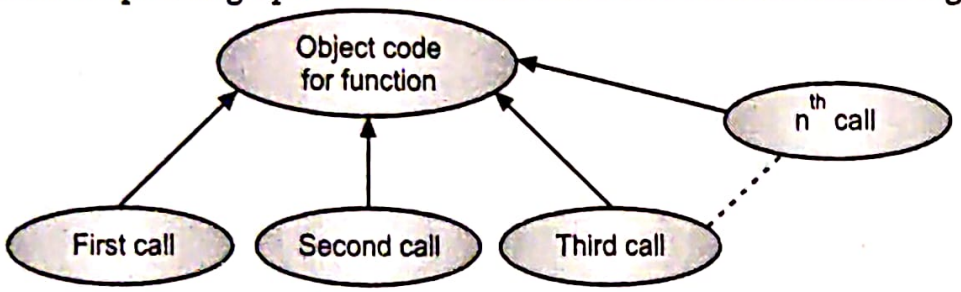


Fig. 2.6: Calls Given to a Function

- But as we said above, lot of time is needed for all these calls to execute. If we make this function inline then the calling statement is replaced with the function code. So it minimizes the time spent on jumping, pushing arguments and so on. This is shown in Fig. 2.7.

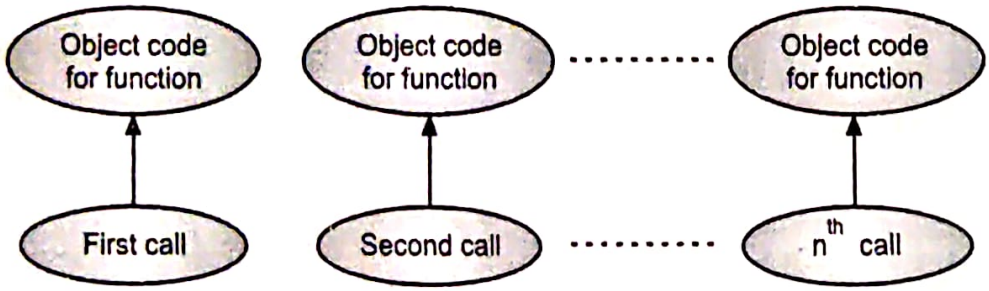


Fig. 2.7: Calls given to Inline Function

Program 2.16: Program illustrates the use of inline function to compute square of a number. **[S-18, 22, 23]**

```
#include<iostream.h>
inline int sq (int x)
{
   int p;
   p = x * x;
   return p;
}
```

```
int main()
{
   int n;
   cout<<"Enter number";
   cin>>n;
   cout<<"square is"<<sq(n)<<endl;
}
```

Output:

```
Enter number 12
square is 144
```

2.11 DEFAULT ARGUMENTS [S-18, 22, 23, W-22]

- C++ allows to define an argument whose value will be automatically used by the compiler, if it is not provided during the function call, this argument is called as default arguments.
- When declaring a function, we can specify a default value for each parameter in C++. In C++, the concept of default argument is used. We know that when a function call is given the number of arguments of function definition and function call should be same; otherwise, error will be displayed. But in C++ we can specify a default value for each parameter when function is declared.
- The parameters which does not have any default arguments are placed first whereas those with default values are placed later. Therefore, we can call the same function with fewer arguments than the defined one in prototype. The default values are specified when the function is declared.
- Compiler takes care of all the default arguments and values through the prototype. In a short way, this means that when a function uses default arguments, the actual function call has the option to specify new values or use the default values.
- The default arguments are generally used in the situation where some arguments are having the same value.

Program 2.17: Program shows the use of default argument.

```
#include<iostream.h>
int mult(int a, int b=2, int c=3, int d=4);  //prototype
int main()
{
   cout<<mult(1,5)<<'\n';
   cout<<mult(3,1,4)<<'\n';
   cout<<mult(6)<<'\n';
   cout<<mult(5,2,1,3)<<'\n';
}
```

```
int mult(int a, int b, int c, int d)
{
    int product;
    product = a * b * c * d;
    return product;
}
```

Output:

```
60
48
144
30
```

- In the first call, we have specified 2 arguments, hence default value of c and d are used. In second call default value d is used. In 3rd call, default value of b, c, d are used.
- The default arguments are checked for the type at the time of declaration whereas they are evaluated at the time of call. The default values are checked from right to left. Therefore, if we have declared the prototype in following manner, compiler tells it whether it is valid or invalid.
- **For example:**
  1. `add (int a, int b, int c = 30, int d = 40); valid`
  2. `add (int a = 5, int b, int c = 30, int d = 40); invalid`
  3. `add (int a, int b, int c = 20, int d); invalid`
  4. `add (int a, int b, int c, int d = 40); valid`
- It is very important to role the two main points about default arguments:
  1. If we first declare a function and then define it, make sure to specify the default argument values in the declaration and not in definition.
  2. Don't specify the default values in the function definition if they are already specified in function declaration.

Program 2.18: Program which consist a function to find out the rate of interest with default arguments. [S - 18]

```
#include <iostream>
using namespace std;
float si(int p, int n, int r=5)
{
    return (p*n*r)/100;
}
int main()
{
    int p, n, r;
    cout<<"Enter principal amount: ";
    cin>>p;
```

```
    cout<<"Enter duration (in years): ";
    cin>>n;
    cout<<"Enter rate of interest: ";
    cin>>r;
    cout<<"Simple interest = "<<si(p, n, r);
    cout<<"Simple interest = "<<si(p, n);
    return 0;
}
```

Output:

```
Enter principal amount: 1000
Enter duration (in years): 2
Enter rate of interest: 10
Simple interest = 200Simple interest = 100
```

Program 2.19: Write a program to calculate area and circumference of a circle using inline function. [W-18, 22]

```
#include<iostream.h>
#include<conio.h>
const pi = 3.14159;

inline float circum(float x)   //finds circumference of circle
{
    return(2*pi*x);
}
inline float area(float x)
//finds area of circle
{
    return(pi*x*x);
}
void main()
{
    float r;
    clrscr();
    cout << "\n Enter the radius of the circle: ";
    cin >> r;
    cout << "\n Circumference: " << circum(r);
    cout << "\n Area: " << area(r);
    getch();
}
```

Output:

```
Enter the radius of the circle: 6
Circumference: 37.68
Area: 113.04
```

Summary

- The smallest individual units in a program are known as tokens.
- An Identifier is any name of variables, functions, classes etc. given by the programmers.
- Constant refer to fixed values that the program cannot alter or change.
- A variable is a named location in memory that is used to hold a value that may be modified by the program.
- A function is a block of code that performs some specific operation or task. The function can invoked, or called, from any number of places in the program. The values that are passed to the function are the arguments, whose types must be compatible with the parameter types in the function definition.
- Functions are of two types in C++ namely, Library functions (User can use library function by invoking function directly; they do not need to write it themselves) and User-defined Functions (defined/developed by user for their own needs).
- Every C++ program has at least one function, which is main(), and all the most trivial programs can define additional functions.
- A function declaration (also called function prototype) consists of a return type, a function name, and a list of arguments as number and type of arguments.
- A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.
- If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the formal parameters of the function.
- The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
- By default, C++ uses call by value to pass arguments. In general, this means that code within a function cannot alter the arguments used to call the function.
- The call by reference method of passing arguments to a function copies the reference of an argument into the formal parameter. Inside the function, the reference is used to access the actual argument used in the call. This means that changes made to the parameter affect the passed argument.
- An inline function is a function whose code is copied in place of each function call. In other words, each call to inline function is replaced by its code. Inline functions can be declared by prefixing the keyword inline to the return type in the function prototype.
- Default arguments are used in calls where trailing arguments are missing.

Check Your Understanding

1. Which of the following is legal declaration of a reference?
   (a) int &a = 10;
   (b) int &a = m;
   (c) int &a = sqr(m);
   (d) int *a = &15;

2. Which of the following statements is illegal?
   (a) int *p = new int (15);
   (b) int *p = new int;
   (c) int *p = new int(10);
   (d) delete p( );
3. Which of the following is not keyword?
   (a) bool
   (b) abstract
   (c) protected
   (d) mutable
4. Which of the following keywords is used to control access to a class member?
   (a) default
   (b) break
   (c) protected
   (d) goto
5. Which of the following statements is true in C++?
   (a) Classes cannot have data as public members.
   (b) Structures cannot have functions as members.
   (c) Class members are public by default.
   (d) None of these
6. The ..... operator extracts the value of a variable from cin object.
   (a) extraction
   (b) insertion
   (c) instance
   (d) none
7. The words which has predefined meaning and cannot be changed by the users are known as .....
   (a) constants
   (b) identifiers
   (c) keywords
   (d) none
8. The maximum number of characters used in identifiers are .....
   (a) 356
   (b) 31
   (c) 8
   (d) none
9. The dynamic memory allocation can be done through ..... operator.
   (a) new
   (b) delete
   (c) pointer
   (d) none
10. Following which keyword is used to declare a constant?
    (a) const keyword
    (b) #define preprocessor
    (b) both (a) & (b)
    (d) none of these
11. What is?: called?
    (a) ternary operators
    (b) binary operator
    (c) arithmetic operator
    (d) none of these
12. Which keyword is used to access to variable in namespace?
    (a) static
    (b) using
    (c) const
    (d) dynamic

Answers

1. (b)	2. (d)	3. (b)	4. (c)	5. (d)	6. (b)	7. (c)	8. (b)	9. (a)	10. (c)
				11. (a)	12. (b)				

Practice Questions

Q.I Answer the following questions in short:

1. What is Token?
2. Enlist various keywords of C++.
3. Define operator. What are the types of Operators in C++?
4. Enlist various user and derived data types.
5. What is function? Enlist its features.
6. How a function works?
7. When should we use an inline function?
8. Does an inline function increase the code size and improve performance?

Q.II Answer the following questions:

1. Explain basic data types in C++.
2. What are manipulators used in C++? Explain one in detail.
3. Explain the term Reference Variable in detail.
4. What is Operator? What are its Types?
5. Explain Memory Management Operators in brief.
6. Explain Scope Resolution Operator with example.
7. Write a program to print year is leap or not.
8. In C++, a variable can be declared anywhere in the scope. What is the significance of this feature?
9. What do you mean by dynamic initialization of a variable? Give an example.
10. What are the benefits of pass by reference method of parameter passing?
11. What are default arguments? Write a program to compute tax. A tax-compute function takes two arguments: amount and tax percentage. Default tax percentage is 15% of income.
12. Write an inline function for finding minimum of two members.

Q.III Define the term:

1. Identifier
2. Constant
3. Token
4. Reference variable
5. Variable
6. Function
7. Inline function
8. Default argument

Previous Exam Questions

April 2018

1. What is Reference Variable? [2 M]

Ans. Refer to Point 2.4.3.

2. What is INLINE function? [2 M]

Ans. Refer to Point 2.10.

3. Explain default argument with the help of suitable example and also write in which situation default arguments are useful. [4 M]

Ans. Refer to Point 2.11.

4. What is need of call by reference? Explain with example. [4 M]

Ans. Refer to Point 2.9.1.

5. Write a C++ program to calculate square and cube of an integer number by using inline function. [4 M]

Ans. Refer to Program 2.16.

6. Write a program which consist a function to find out the rate of interest with default arguments. [4 M]

Ans. Refer to Program 2.18.

October 2018

1. Define the following terms: [2 M]

(i) Identifier

Ans. Refer to Point 2.1.3.

(ii) Constant

Ans. Refer to Point 2.1.4.

2. What is function prototype? [2 M]

Ans. Refer to Point 2.8.

3. Write a note on memory management operators in C++. [4 M]

Ans. Refer to Point 2.5.2.

4. What is inline function? Write difference between macro and inline function. [4 M]

Ans. Refer to Point 2.10.

5. Write a program to calculate area and circumference of a circle using inline function. [4 M]

Ans. Refer to Program 2.19.

6. Explain scope resolution operator with an example. [4 M]

Ans. Refer to Point 2.5.1.

7. Trace the output of the following program and explain it. Assume there is no syntax errors. **[4 M]**

```
void position (int & C1, int C2 = 3)
{
    c1+ = 2;
    c2+ = 1;
}
int main ( )
{
    int P1 = 20, P2 = 4;
    Position (P1);
    Cout << P1 << "," << P2 << endl;
    Position (P2, P1);
    Cout << P1 << "," << P2 << endl;
}
```

Ans. Refer to Point 2.11.

April 2019

1. What is return by reference? **[2 M]**

Ans. Refer to Point 2.9.3.

2. What is setf() function? **[2 M]**

Ans. Refer to Point 2.6.

3. What is tokens in C++? Explain in detail. **[4 M]**

Ans. Refer to Point 2.1.2.

4. Explain inline function. Write the circumstances in which inline function will work like a normal functions. **[4 M]**

Ans. Refer to Point 2.10.