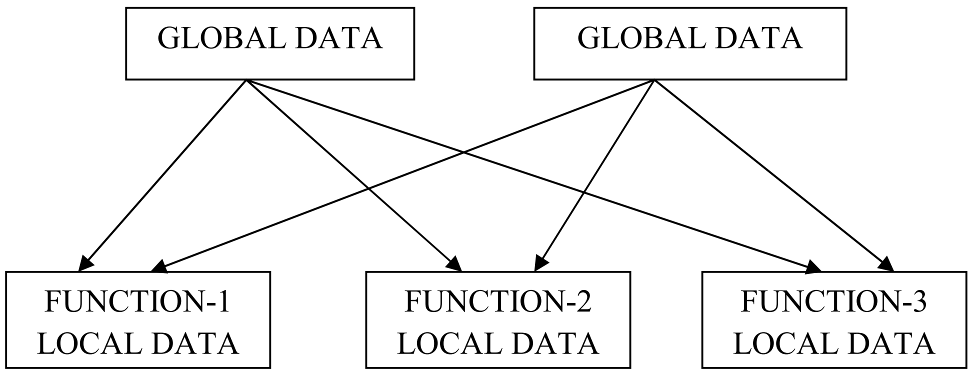
1 *Structured programming vs. object oriented programming*

1.1 Structured programming was a powerful tool that enabled programmers to write complex programs easily. However the programs became larger so the structured approach failed to make desired results in terms of bug (error) free, easy-to-maintain and reusable programs.

1.2 Procedure oriented programming:

- Conventional programming using high level languages such as COBOL, FORTRAN and c is commonly known as pop. In the pop approach, the problem is viewed as a sequence of things to be done such as reading, calculating and printing.
- A number of functions are written to accomplish these tasks.
- The primary focus is on functions.
- Pop basically consists of writing a list of instructions or actions for the computer to follow and these instructions are organized into groups known as functions.
- While we concentrate on the development of functions very little attention is given to the data that are being used by the different functions.
- In a multifunction program many important data items are placed as global so that they may be accessed by all the functions. Also each function may have its own local data.



Relationship of data & functions in pop

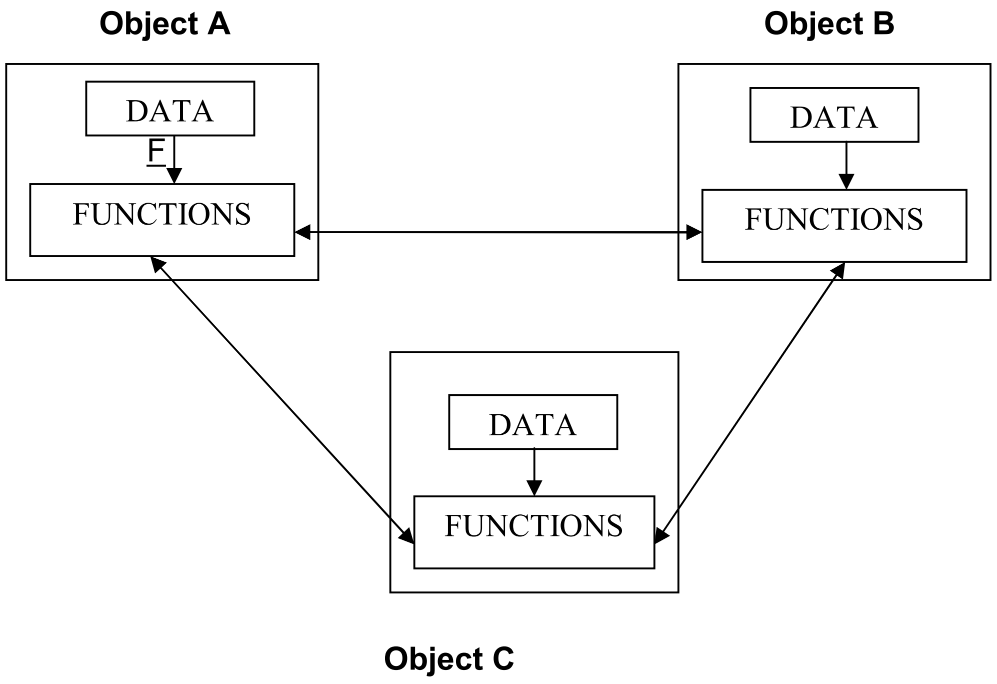
- In a large program it is very difficult to identify what data is used by which function.
- So, the global data are more helpless to an unintentionally change by a function.
- Another drawback is that it does not model real world problems very well.

Characteristics of pop:-

1. Give importance on doing things (algorithms)
2. Large programs are divided into smaller programs known as functions.
3. Most of functions share global data.
4. Data move openly around the system from function to function.
5. Functions transform data from one form to another.
6. Employs top-down approach in program design.

1.3 Object-Oriented Programming Paradigm:-

1. The basic reason of inventing object-oriented approach is to remove some drawbacks encountered in pop.
2. OOP treats data as critical element in the program development and doesn't allow data to flow freely around the system.
3. It ties (binds) the data more closely to the functions that are operated on it and protects data from accidental modification from outside functions.
4. OOP allows decomposition of a problem into a number of entities called objects and then builds data and functions around these objects.
5. The data of an object can be accessed by only the functions associated with the object.



Features of object oriented programming:-

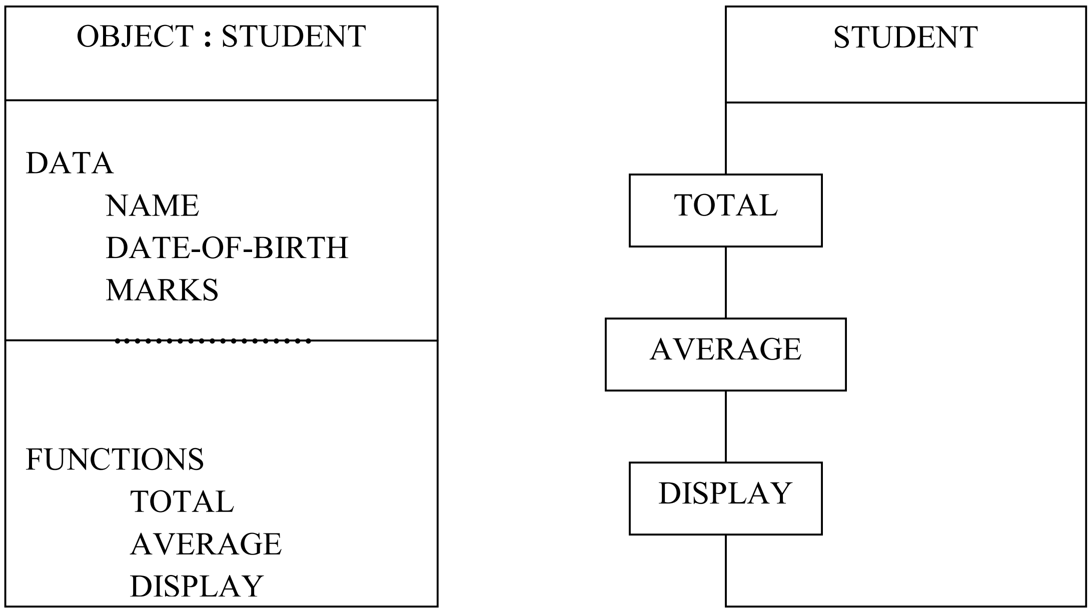
1. Emphasis on data rather than procedure.
2. Programs are divided into objects.
3. Data structures are designed such that they characterized the objects.
4. Functions that operate on the data of an object are tied together in the data structure.
5. Data is hidden and cannot be accessed by external functions.
6. Objects may communicate with each other through functions.
7. Now data and functions can be easily added whenever necessary.
8. It follows bottom-up approach in program design.

- We **define** "object-oriented programming an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand".

2 *Basic OOP concepts : objects, classes, encapsulation, data hiding, inheritance, polymorphism*

2.1 Objects:-

- Objects are the basic run-time entities in an object-oriented system.
- They may represent a person, a place, a bank account, and a table of data or any item that the program has to handle.
- Programming problem is analyzed in terms of objects and the nature of communication between them.
- Program objects should be chose such that they match closely with the real world objects.
- For example, we can say that individual student is individual object.



Two ways of representing an object

2.2 Classes :-

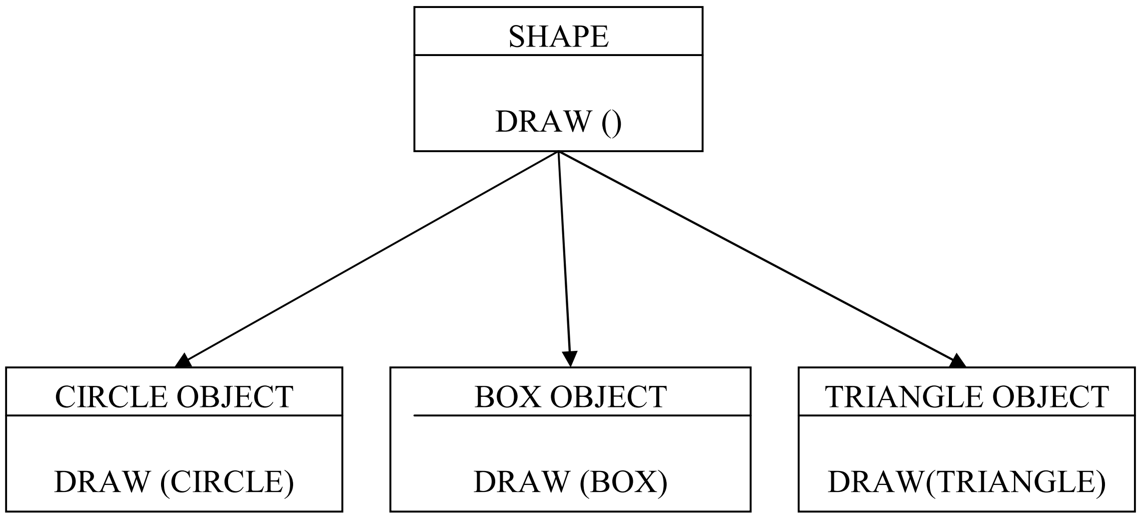
- It is set of entity like the student is class and individual student is object.
- We know that objects contain data and code to manipulate that data.
- The entire set of data and code of an object can be made a user-defined data type with the help of a class.
- Objects are variables of the type class.
- Once a class has been defined, we can create any number of objects belonging to that class.
- A class is a collection of objects of similar type.
- Example: - mango, apple & orange are members of the class fruit.
- The syntax used to create an object is no different than the syntax used to create an integer object in c.
- If fruit is defined as a class, then the statement is fruit mango, apple, orange;

2.3 Data abstraction and encapsulation :-

- The enclosing or covering of data & functions into a single unit (called class) is known as encapsulation.
- Data encapsulation is the most striking feature of the class.
- The data is not accessible to the outside world and only those functions which are enclosed in the class can access it.
- These functions provide the interface between the object's data and the program.
- This insulation of the data from direct access by the program is called data hiding or information hiding.
- Abstraction refers to the representation of important features without including the background details or explanations.
- Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight and cost and functions to operate on these attributes.
- They encapsulate (bind) all the essential properties (attributes) of the objects that are to be created.
- These attributes are called data members.
- The functions that operate on these data are called methods or member functions.

2.4 Polymorphism :-

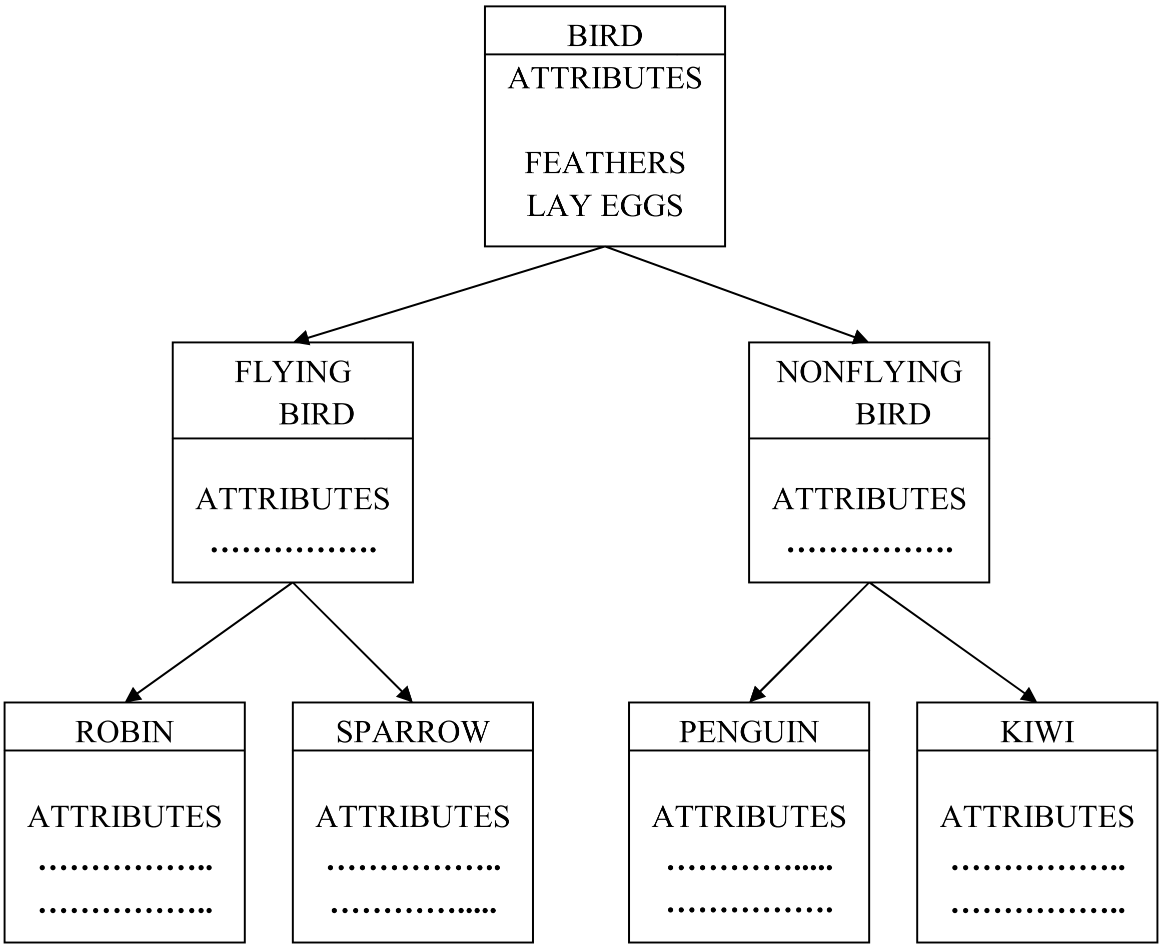
- Polymorphism means the ability to take more than one form.
- An operation may execute in different behavior in different instances.
- The behavior depends upon the types of data used in the operation.
- Consider the operation of addition which generates a sum of two numbers.
- If we pass strings as operands then this function concate these two strings.
- The process of making an operator (addition, subtraction etc.) To exhibit different behaviors in different instances is known as operator overloading.
- If the function name is same but they have different number of argument or/ and different types (data types) of arguments, then this type is called function overloading.



Polymorphism

2.5 Inheritance :-

- Inheritance is the process by which objects of one class get the properties of objects of other class.
- It supports the concept of hierarchical classification.
- In OOP, the concept of inheritance provides the idea of reusability.
- This means that we can add additional feature to an existing class without modifying it.
- This is possible by deriving a new class from the existing one.
- The new class will have all the combined features of both the classes.
- The following diagram & example describe that how inheritance is done.



Property inheritance

- In above diagram the main class is bird, the flying bird and non-flying bird are derived from bird class.
- So, both have attributes of bird class.
- Bird robin and sparrow can fly, so both are derived from flying bird class.

2.6 Dynamic binding :-

- Binding refers to the linking of a procedure call with the code to be executed as the response.
- Dynamic binding (also known as late binding) means that the code associated with a given procedure call at run-time.
- It is associated with polymorphism and inheritance.
- This means when a function called the appropriate arguments (numbers & types) are matched.
- This process is done at run-time so it is called dynamic binding.

2.7 Message passing :-

- An object oriented program consists of a set of objects that communicate with each other.
- The process of programming in an object-oriented language, involves the following basic steps.
    1. Creating classes that define objects and their behavior.
    2. Creating objects from class definitions, and
    3. Establishing communication among objects.

- Objects communicate with one another by sending and receiving information.
- The concept of message passing makes it easier to talk about building systems that directly model or simulate their real-world counterpart.
- A message for an object is a request for execution of a procedure, and so it invokes or calls a function (procedure) as the receiving object which generate the desired result.
- Message passing involves the name of the object, the function (message) name, and the information to be sent.
- *Example:-*

```
                 employee e1;
      class name ↙            ↘ object
                 e1.salary (name);
          object ↙      ↓      ↘ information (arguments)
               function name (message)
```

3 *Introduction to c++: structure of a c++ program, data types, variables, constants, expressions, statements and operators.*

3.1 Introduction to c++:

- C++ is an object-oriented programming language.
- It was developed by Bjarne Stroustrup at 'AT&T' bell laboratories in Murray Hill, New Jersey, USA in the 1980's. C++ is an extension of 'C' within a major addition of the class construct feature of simulu67.
- The idea of C++ come from the C increment operator ++, thereby suggesting that C++ is an incremented version of C.
- C++ is superset of C.

- The most important facilities that C++ adds on to c are classes, inheritance, function overloading and operator overloading.
- These features enable creating of obstruct data types; inherit properties from existing data types and support polymorphism. So making of C++ is a truly object-oriented language.

3.1.1 Features of c++ :-

- Like C, the C++ program is a collection of functions.
- As usual, execution begins at main ( ).
- Every C++ program must have a main ( ) function.
- Like C, the C++ statements terminate with semicolumns.

3.1.2 Advantages of OOP

- Using inheritance we can remove unwanted code and expand the use of existing classes.
- The principle of data hiding helps the programmer to build secure programs that can't be separated by code in other parts of the program. So, the procedural languages had not data hiding concepts.
- It is possible to create multiple instances (cases/samples) of an object to co-exist (exist at the same time) without any interference.
- It is possible to map objects in the problem domain to those in the program.
- It is easy to partition the work in a project based on objects.
- Object-oriented systems can be easily upgraded from small to large systems.
- Software complexity can be easily managed.
- We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and get higher productivity.

3.1.3 Disadvantages of OOP :-

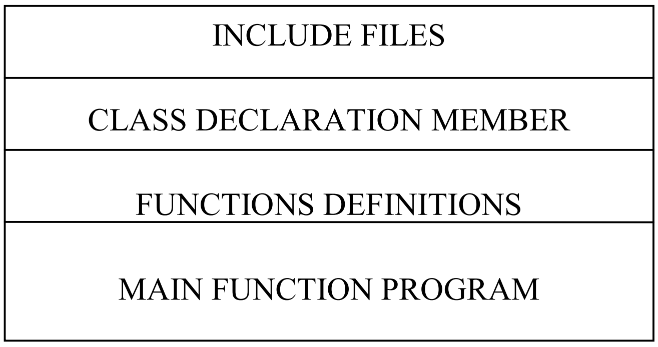
1. Compiler overhead.
2. Runtime overhead.
3. Re-orientation of s/w developer to object-oriented thinking.
4. Requires the mastery over the following areas :
   - Software engineering
   - Program methodologies
5. Benefits only in long run while managing large s/w projects.

3.1.4 Applications of OOP :-

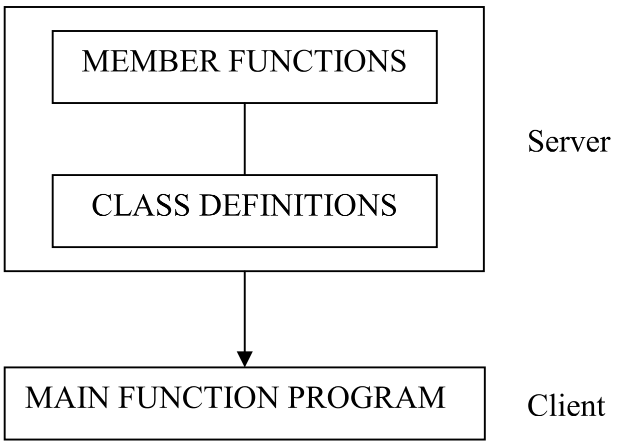
1. Real time systems.
2. Simulation and modeling.
3. Object-oriented databases.

4. Hypertext, hypermedia and expert-text.
5. Artificial intelligence (AI) and expert systems.
6. Neural networks and parallel programming.
7. Decision support and office automation systems.
8. CIM / CAM / CAD systems.

3.2 Structure of a c++ program



- It is a common practice to organize a program into three separate files.
- The class declarations are placed in a header file and the definitions of member functions go into another file.
- This approach enables the programmer to separate the abstract specification of the interface (class definition) from the implementation details (member function definitions).
- The main program that uses the class is placed in a third file which "includes" the previous two files as well as any other files required.
- This approach is based on the concept of client-server model as shown in figure below.
- The class definition including the member functions equivalent to the member server that provides services to the main program known as client.
- The client uses the server through the public interface of the class.

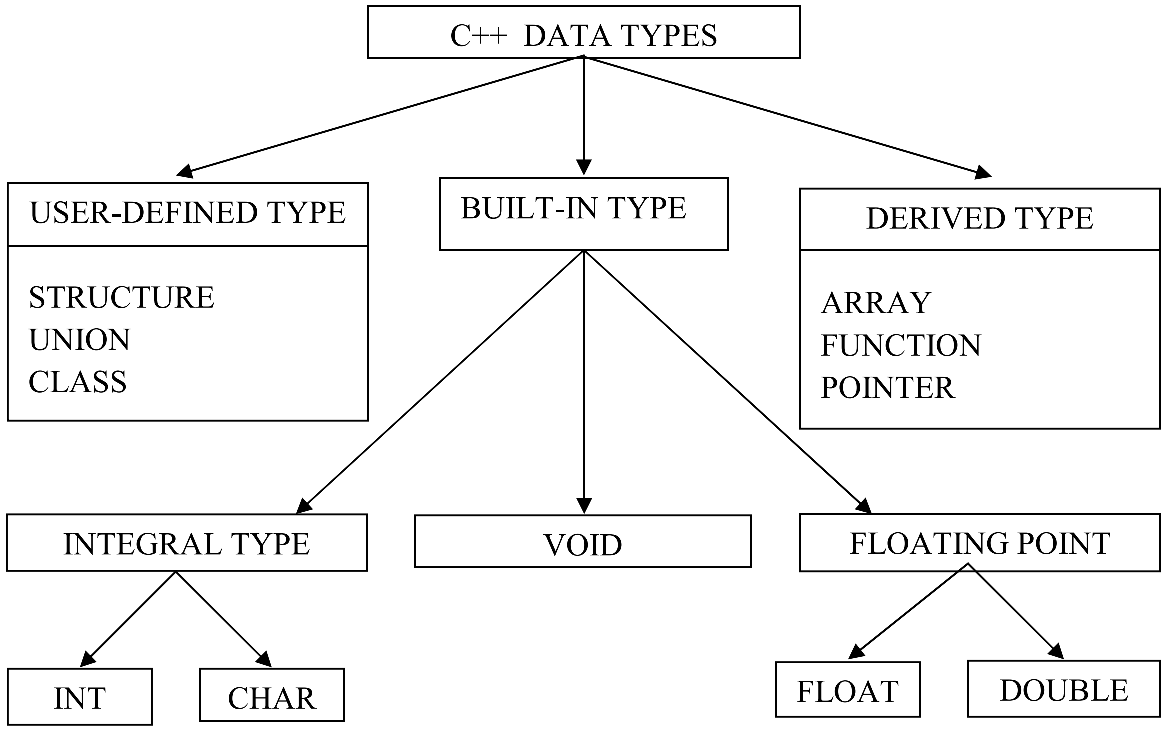


client-server model

3.3 Data types

3.3.1 Basic Data Types

Both C and C++ compilers supports all the built-in(also known as basic or fundamental) data types. The modifiers signed, unsigned, long and short may be applied to character and integer basic data types. However, the modifier long may also be applied to double. Data types representation in machine specific in c++.



Hierarchy of c++ data types

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	-31768 to 32767
Short int	2	-31768 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

3.3.2 User-defined data types :-

Structures and classes :-

- User-defined data-typs such as struct and union in ‘c’ are legal in c++.
- Class is also a user-defined data-type which is just like any other data-type, to declare variables.

Enumerated data-type :-

- It provides a way for attaching names to numbers.
- The enum keyword (from c) automatically enumerates a list of words by assigning them values 0, 1, 2, and so on… This facility provides an alternative means for creating symbolic constants.
- Example :-

```
enum shape {circle, square, triangle};
enum colour {red, blue, green, yellow};

void main( )
{
    cout<<”enter shape code :”;
    int code;
    cin>> code;
    while (code >= circle && code <= triangle)
    {
            switch (code)
            {
                    case circle:
                            ………..
                            ………..
                            ………..
                            break;
                    case rectangle:
                            ………..
                            ………..
                            ………..
                            break;
                    case triangle;
                            ………..
                            ………..
                            ………..
                            break;
            }
            cout<<”enter shape code :”;
            cin>>code;
    }
}
```

3.3.3 Derived data types :-

Arrays :-

- Initializing a character array in ANSI C, the compiler will allow us to declare the array size as the exact length of the string constant.
- Char string[3] = “xyz”;
- But in c++ the size should be one larger than the number of characters in the string.
- Char string[4] = “xyz”;

Functions :-

- Functions have undergone major changes in C++. While some of these changes are simple, others require a new way of thinking when organization our programs.
- Many of these modifications and improvements were driven by the requirements of the object-oriented concept of C++. Some of these were introduced to make the C++ program more reliable and readable. All the functions are discussed later.

Pointers :-

- Pointers are declared and initialized as in ‘c’

```
int *ip;        // int pointer
ip = &x;        //address of x assigned to ip
*ip = 10;       // 10 assigned to x through indirection
```

- C++ adds the concept of constant pointer and pointer to a constant.

```
char * const ptr1 = “hello”;  //constant pointer
```

- We cannot modify the address that ptr1 is initialized to.

```
int const *ptr2 = &m;            // pointer to a constant
```

- Pointers are extensively used in C++ for memory management and achieving polymorphism.

3.4 Variables

- We know that, in C, all variables must be declared before they are used in executable statements. This is true with C++ as well. However, there is a significant difference between C and C++ with regard to the place of their declaration in the program. C requires all the variables to be defined at the beginning of a scope. When we read a C program, we usually come across a group of variable declarations at the beginning of each scope level. Their actual user appears elsewhere in the scope, sometimes far away from the place of declaration. Before using a variable, we should go back to the beginning of the program to see whether it has been declared and, if so, of what type.

- C++ allows the declaration of a variable anywhere in the scope. This means that a variable can be declared right at the place of its first use. This makes the program much easier to write and reduces the errors that may be caused by having to scan back and forth. It also makes the program easier to understand because the variables are declared in the context of their use.
- For example,

```
        Float x;
        Flot sum=0;
        Int a;
        For (i=0; i<5 i++)
```

<u>Dynemic Initialization of Variables</u>

- In c, a variable must be initialized using a constant expression, and the C compiler would fix the initialization code at the time of compilation. C++, however, permits initialization of the variables at run time. This is referred to as *dynamic initialization.* In C++, a variable can be initialized at run time using expression at the place of declaration. For example, the following are valid initialization statements:

```
        ……..
        ……..
        Int n=strlen(stirng);
    …….
```

<u>Reference Variables</u>

- C++ introduces a new kind of variable known as the *reference variable.* A reference variable provides and *alias* (alternative name) for a previously defined variable. For example, if we make the variable sum a reference to the variable total, then sum and total can be used interchangeably to represent that variable. A reference variable is created as follows:
- Data-type & reference-name = variable-name;
- Example:

```
    Float total=100;
    Float & sum=total;
```

<u>3.5 Constants</u>

- Identifiers refer to the names of variables, functions, arrays, classes etc… created by the programmer.
- They are the fundamental requirement of any language.
- Each language has its own rules for naming these identifiers.
- The following rules are common to both c and c++.
  1. Only alphabetic characters, digits and underscores are permitted.
  2. The name cannot start with a digit.
  3. Uppercase and lowercase letters are distinct.
  4. A declared keyword cannot be used as a variable name.

- Constants refer to fixed values that do not change during the execution of a program.
- They include integers, characters, floating point numbers and strings.
- Literal constants do not have memory locations.

- <u>Examples:-</u>

```
123     //  decimal integer
12.34   //  floating point integer
037     //  octal integer
0x2     //  hexadecimal integer
"c++"   //  string constant
'a'  //  character constant
```

<u>Tokens :-</u>

- The smallest individual units in a program are known as tokens.
  - Keywords
  - Identifiers
  - Constants
  - Strings
  - Operators

<u>Keywords :-</u>

- They are explicitly (specifically) reserved identifiers and cannot be used as names for the program variables or other-defined program elements.
- <u>Examples:-</u>

auto	double	inline	return	
break	else	int	short	
case	enum	long	signed	
char	extern	new	stuff	
class	float	operator	static	
const	for	private	struct	
continue	friend	protected	switch	
default	goto	public	this	
do	if	register	union	
unsigned	virtual	void	while	etc…….

3.6 Expressions

- An expression is a combination of operators, constants and variables arranged as per the rules of the language.
  1) Constant expressions.
  2) Integral expressions.
  3) Float expressions.
  4) Pointer expressions.
  5) Relational expressions.
  6) Logical expressions.
  7) Bitwise expressions.

1) **Constant expressions :-**

- Constant expressions consists of only constant values.
- Examples:-

```
15
20 + 5 / 2.0
'x'
```

2) **Integral expressions :-**

- Integral expressions are those which produce integer results after implementing all the automatic and explicit type conversions.
- Examples:-

```
m
m * n – 5
m * 'x'
5 + int(2.0)        // type conversion
```

- Where m and n are integer variables.

3) **Float expressions :-**

- Float expressions are those which, after all conversions, produce floating point results.
- Examples:-

```
x + y
x * y / 10
5 + float(10)
```

- Where x and y are floating point variables.

4) **Pointer expressions :-**

- Pointer expressions produce address values.
- Examples:-

```
&m
ptr
ptr + 1
"xyz"
```

- Where m is a variable and ptr is a pointer.

5) **<u>Relational expressions :-</u>**
   - Relational expressions yield results of type bool which takes a value true or false.
   - <u>Examples**:-**</u>

     ```
     x<=y
     a + b = = c + d
     m + n > 100
     ```
   - When, arithmetic expressions are used on either side of a relational operator, they will be evaluated first and then the results compared.
   - Relational expressions are also known as Boolean expressions.

6) **<u>Logical expressions :-</u>**
   - Logical expressions combine two or more relational expressions and produces <u>bool</u> type results.
   - <u>Examples**:-**</u>

     ```
     a > b && x = = 10
     x = = 10 || y = = 5
     ```

7) **<u>Bitwise expressions :-</u>**
   - Bitwise expressions are used to manipulate data at bit level. They are basically used for testing or shifting bits.
   - <u>Examples**:-**</u>

     ```
     x << 3          // shift three bit position to left
     y >> 1          // shift one bit position to right
     ```
   - Shift operators are often used for multiplication and division by powers of two.

<u>3.7 Operators</u>

<u>3.7.1 Arithmetic Operators</u>

- The C++ language has both unary and binary arithmetic operators. Unary operators are those, which operate on a single operand whereas, binary operators operate on two operands. The arithmetic operators can operate on any built-in data type. Arithmetic operators and their meaning are shown in below table.

Unary operator example :

```
Int x=5;
Int y;
    Y= -x;
```

Binary operator example:

```
Int x=5; int y=10;
Int c=x+y;
```

Table 1: Arithmetic operators	
Operator	**Meaning**
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

3.7.2 Relational Operators

- Using relational operators we can direct the computer to compare two variables. The C++ relational operators are summarized below, with their meanings. Pay particular attention to the equality operator; it consists of two equal signs, not just one.

Table 2: Relational Operators	
Operator	**Meaning**
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal
!=	Not equal

3.7.3 Logical Operators

- Logical operators in C++, as with other computer languages, are used to evaluate expressions which may be true or false. Expressions which involve logical operations are evaluated and found to be one of two values: true or false.

Table 4: Logical Operators			
Operator	**Meaning**	**Example of Use**	**Truth Value**
&&	AND	(exp 1) && (exp 2)	True if exp 1 and exp 2 are BOTH true.
||	OR	(exp 1) || (exp 2)	True if EITHER (or BOTH) exp 1 or exp 2 are true.
!	NOT	! (exp 1)	Returns the opposite truth value of exp 1; if exp 1 is true, ! (exp 1) is false; if exp 1 is false, ! (exp 1) is true.

3.7.4 Output operator :-

cout<< "c++ is better than c";

- The string in the quotation marks is to be displayed on the screen.
- The identifier cout is a predefined object that represents the standard output stream in c++ which is the same as used in c as "printf" statement.
- The operator << is called the insertion or put to operator.
- It inserts or sends the contents of the variable on its right to object on its left.

3.7.5 Input operator : -

cin>>number1;

- The identifier 'cin' is a predefined object in C++ that corresponds to the standard input stream. The operator '>>' is known as extraction or get from operator.
- It extracts or takes the value from the keyboard and assigns it to the variable on its right (number1).

3.7.6 Cascading of i/o operator : -

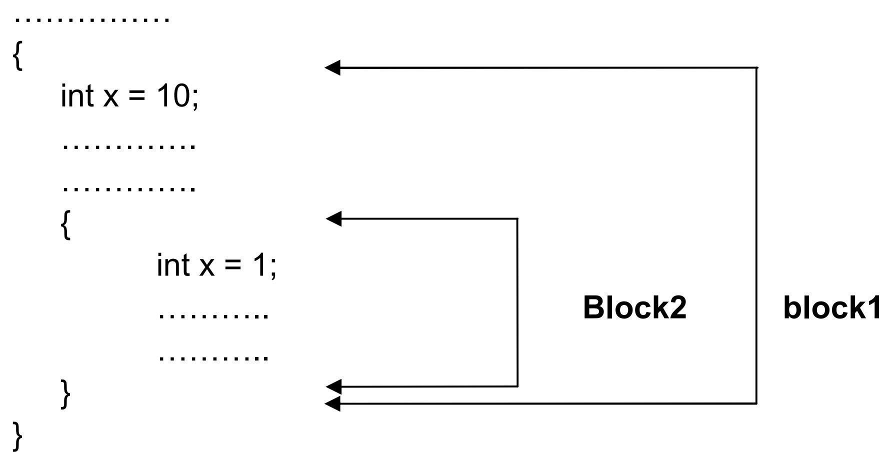
- The multiple use of << / >> in one statement is called cascading.
- Examples: -
  - cout<< "total = "<<tot<<"\n";
  - cout<<"sum = "<<"\n";
  - cout<<"sum ="<<sum<<"\n";
  - cout <<äverage ="<<avg<<"\n";

- We can also cascade input operator.
  - cin>>number1>>number2;

3.7.7 Scope resolution operator :

- Like C, C++ is also a block-structured language. Blocks and scopes can be used in constructing programs. We know that the same variable name can be used to have different meanings in different blocks. The scope of the variable extends from the point of its declaration till the end of the block containing the declaration. A variable declared inside a block is said to be local to that block

Example :-



```
……………
{
    int x = 10;
    …………
    …………
    {
            int x = 1;
            ………..
            ………..
    }
}
```

- In c, the global version of a variable cannot be accessed from within the inner block.
- C++ resolves this problem by introducing a new operator :: called the scope resolution operator.
- This can be used to uncover a hidden variable.

  :: variable_name
- This operator allows access to the global version of variable.

Example :-

```
#include<iostream.h>
int m=10;      // global m
int main( )
{
    int m=20; // m re-declared, local to main
    {   int k=m;
        int m=30;      // m declared again, local to inner block
        cout<<"we are in inner block \n";
        cout<<"k="<<k<<"\n";
        cout<<"m="<<m<<"\n";
        cout<<"::m="<<::m<<"\n";
    }
    cout<<"\n outer block \n";
    cout<<"m="<<m<<"\n";
    cout<<"::m="<<::m<<"\n";
    return 0;
}
```

Output :-

```
we are in inner block
    k=20
    m=30
    ::m=10
        outer block
    m=20
    ::m=10
```

::m will always refer to the global m.

3.7.8 Manipulators :-

endl

- Manipulators are operators that are used to format the data display.
- The endl manipulator, when used in an output statement, causes a linefeed to be inserted.
- It has the same effect as using the newline character "\n".

setw

- The numbers should be right justified. This form of output is possible only if we can specify a common field width for all the numbers and force them to printed right justified. The setw manipulator does this job.

- Normally value is left justified like…..

1	2	
1	2	3
1		

- For example:

```
M=12;
        N=123;
        O=1;

Cout<<setw(5)<<m<<endl;
        Cout<<setw(5)<<m<<endl;
        Cout<<setw(5)<<m<<endl;
```

Output :

	1	2
1	2	3
		1

3.8 ***Comments*:-**

- C++ enables a new comment symbol // (double slash).
- Comments start with a double slash symbol and terminate at the end of the line.
- Note that there is no closing symbol.
- The double slash comment is basically a single line comment.
- Example:

```
// this is an example of
// c++ program to illustrate
// some of its features
```

- The c comment symbols /*, */ are still valid and are more suitable for multiline comments.
- Example:

```
/* c++ is the
   superset of c*/
```

- Valid statement:

```
for ( i=0; j<n; /*loops n times */ j++)
```

- Invalid statement:

```
for (j=0; j<n; //loops n times j++)
```

- This an invalid statement only for looping purpose.

4 Usage of header files

4.1 The iostream file :-

- #include <iostream.h> (for new version which supports ANSI C++ features)
  or <iostream>
- This directive causes the preprocessors to add the contents of the iostream file to the program.
- It contains declarations for the identifier cout and the operator <<.
- The header file iostream should be included at the beginning of all programs that use input\output statements.

4,2 Some header files which are used in c++ :-

<float.h>	contains the floating-point size limits of the system.
<math.h>	contains function prototypes for math library functions.
<stdio.h>	contains function prototypes for the standard input\output library functions and information used by them. (only used in 'c')
<string.h>	contains function prototypes for c-style string processing functions.

4.3 Return type of main( ) : -

- In C++, main( ) returns an integer type value in the operating system.
- So, every main( ) in C++ should end with a return(0) statement; otherwise a warning or an error might occur.
- Return type for main( ) is explicitly specified as int.
- Note, that the default return type for all functions in C++ is int.

4.4 Use of class: -

```
#include<iostream.h>
class person
{
    char name[20];      } by default private, if not declared
    int age;             }  with public/protected.

    public:
            void getdata(void);
            void display(void);
}; ← terminated by semicolon
```

```
void person::getdata(void)
{    cout<<"ënter name:";
     cin>>name;
     cout<<"enter age:";
     cin>>age;
}
void person::display(void)
{    cout<<"\n name: "<<name;
     cout<<"\n age: "<<age;
}
void main( )
{    person p;
          p.getdata( );
          p.display( );
     }
```

5 *Control flow statements: if else, for loop, while loop, do while loop, switch, break, continue*

Control Flow Statements:

- Control structures are used for creating program which is accurate, error-resistant and maintainable. It is to use one or any combination of the following three control structures:**-**

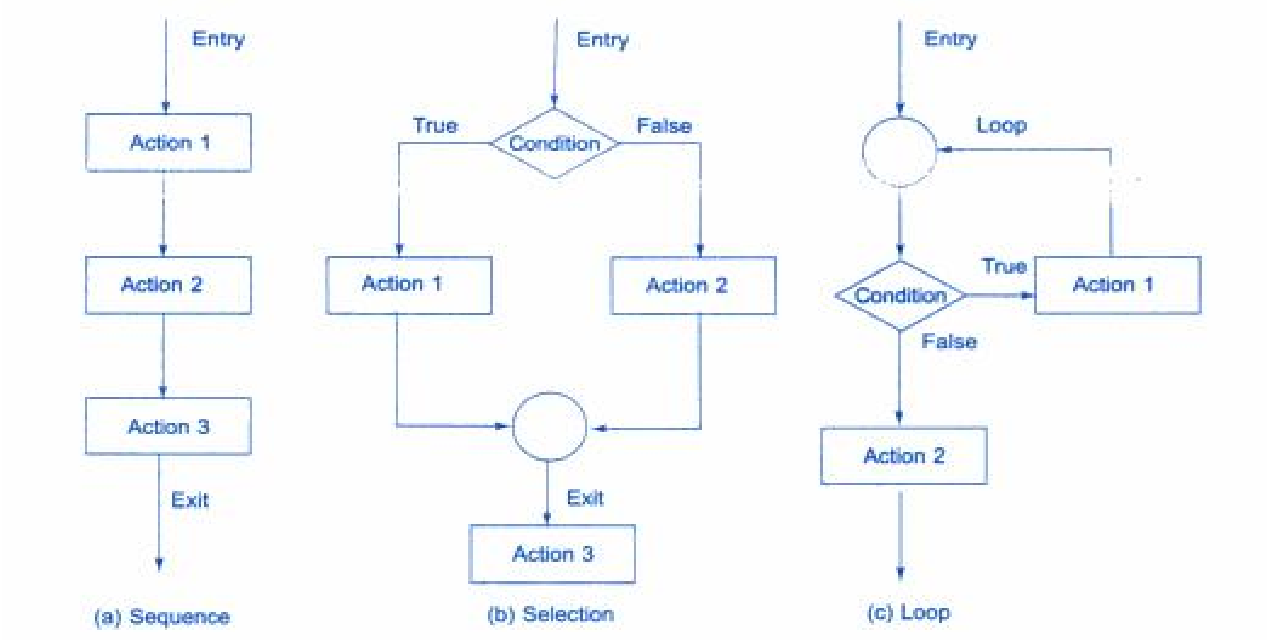


Fig. 3.4 ⇔ *Basic control structures*

1. Sequence structure (straight line)
2. Selection structure (branching)
3. Loop structure (iteration or repetition)

- In real-world, several activities are initiated or repeated based on some decisions. Such activities can be programmed by specifying the order in which computations are carried out.
- Flow control is the way a program causes the flow of execution to advance and branch based on changes in the data state. Branching iteration, dispatch and functions call are all different forms of flow control.
- From following diagram, we can differentiate control structure as selection and looping control structure.

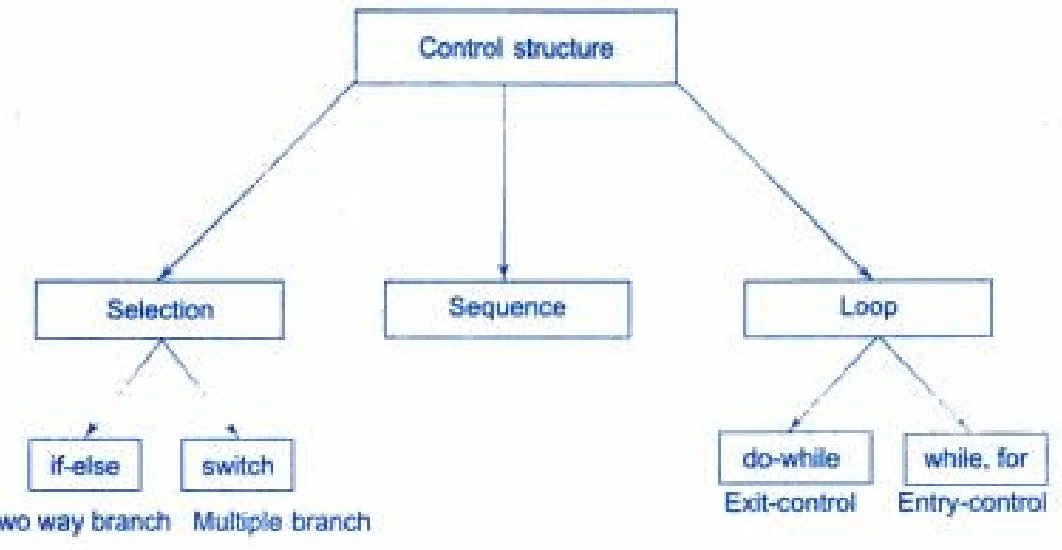


Fig. 3.6 ⇔ *C++ statements to implement in two forms*

<u>5.1Selection structure:</u>

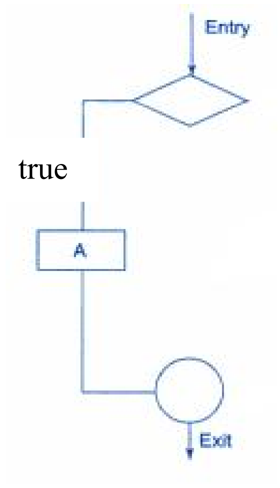
- Selection statements (branching statements) alter sequential execution of program statements.
- It controls the execution of program. Following are the branching statements supported by c++. Following are the branching statement.

<u>5.1.1 Simple if</u>

- The if construct is a powerful decision making statement which is used to control the sequence of the execution of statements. It alters the sequential execution using the following syntax.

```
If (test-expression)
    Statement;
```

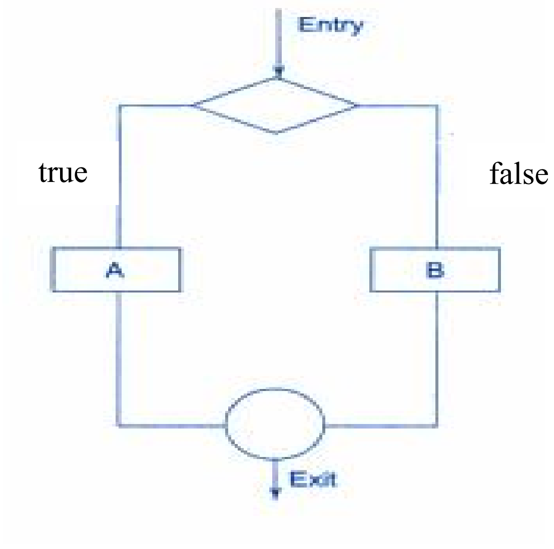
- The test-expression should always be enclosed in parentheses. If test-expression is true, then statement immediately following it is executed. Otherwise, control passes to the next statement following if construct.



```
#include<iostream.h>
{
    void main()
    {
            int a,b;
            cin>>a;
            cin>>b;
            if(a>b)
            {       cout<<"a is big";     }
            cout<<"prog. complete";
    }
}
```

5.1.2 If…else

- The simple if statement will execute a single statement or a group of statements, when the test expression is true. It does nothing when the test expression fails. C++ provides if-else construct to perform some action even when the test expression fails.

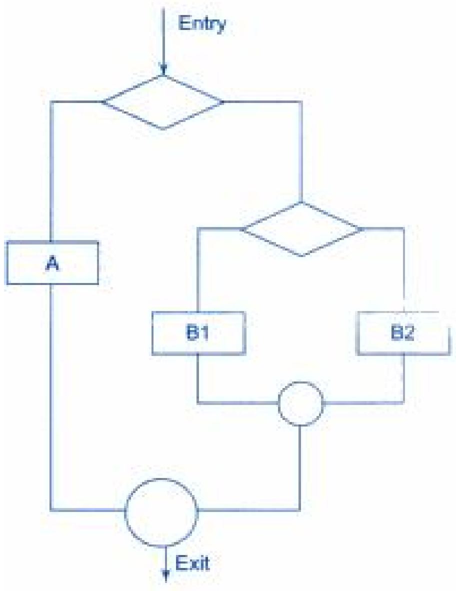


```
If (test-expression)
{
    Statement;
}
Else
{
    Statement;
}
```

```
#include<iostream.h>
{
    void main()
    {
            int a,b;
            cin>>a;
            cin>>b;
            if(a>b)
            {               cout<<"a is big";            }
            else
            {               cout<<"b is big";            }
            cout<<"prog. complete";
    }
}
```

5.1.3 Nested if

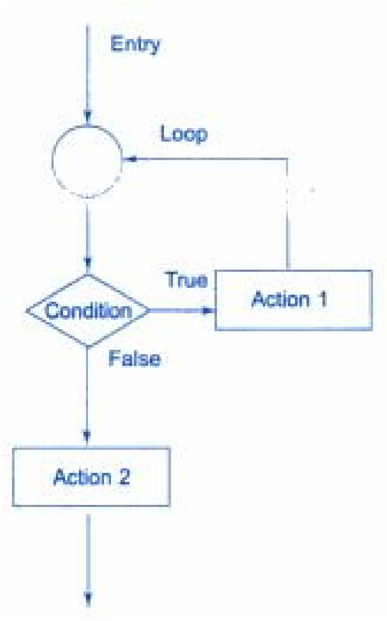
- Multi way decisions arise when there are multiple and different action to be taken under each condition.
- Here, if test-expression1 is true, the whole chain is terminated. Only if test-expression1 is found false, the chains of events continue. At any stage if an expression is true, the remaining chain will be terminated.



```
        If (test-expression1)
        {   Statement1;  }
        Else if(test-expression2)
        {
            Statement2;
            }
            Else
            {
                    Statement3;
            }
    #include<iostream.h>
    {
        void main()
        {
                int a,b;
                cin>>a;
                cin>>b;
                if(a>b)
                {       cout<<"a is big";      }
                if else(b>a)
                {       cout<<"b is big";      }
                else
                {       cout<<"both are same";     }
                cout<<"prog. complete";
        }
    }
```

5.2 Loop structures:

- Loops cause a section of code to be executed repeatedly until a termination condition is met. The following are the looping statements.



5.2.1 For loop

- The for loop is useful while executing a statement a fixed number of times. Even here, more than one statement can be enclosed in curly braces to form a compound statement. The control flow in the for loop is shown below.
- The for loop is useful while executing a statement a fixed number of times.
- *Syntax:*

```
        for(initialization ; condition ; updation)
{
        body of loop
}
        statement ;
```

- *Description*: The initialization part is executed only once. Next the condition is evaluated.
  - If the condition is false then the next statement after for loop is executed.
  - If the condition is true then after executing the body of loop and then update part is executed. The condition is evaluated once again and the whole process is repeated as long as the condition becomes false.
- *Example*: Write a C++ program to display 1 to N number using for loop.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    int n, i ;
            clrscr();
    cout<<"\n Enter any positive number : ";
            cin>>n;
            for( i=1 ; i<=n ; i++)
                    cout<<"\n "<<i;
    getche();
}
```

Output

```
Enter any positive number : 3
1
2
3
```

5.2.2 While loop

- It is used when the number of iteration to be performed are not known in advance.
- *Syntax*:

```
        while ( condition )
{
        body of loop
}
statement ;
```

- *Description*: Initially condition is tested.
  - If the condition is true then the body of loop is executed and the execution continues as long as it remains true.
  - If the condition is false then loop is terminated and next statement is executed.

- *Example*: Write a C++ program to display 1 to N number using while loop.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    int n, i ;
clrscr();
        cout<<"\n Enter any positive number : ";
cin>>n;
i=1 ;
while ( i<=n )
{
        cout<<"\n "<< i;
i++;
}
getche();
}
```

- *Output*

```
Enter any positive number : 3
1
2
3
```

5.2.3 do… while loop

- *Syntax*:

```
        do
        {
                body of loop
        } while ( condition ) ;
statement ;
```

- *Description*: Initially body of loop is executed and then condition is evaluated. The body of loop is executed as long as the condition remains true. The loop is terminating when condition becomes false.

- *Example*: Write a C++ program to display 1 to N number using do…while loop.

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

```
void main()
{
     int n, i ;
clrscr();
         cout<<"\n Enter any positive number : ";
cin>>n;
i=1 ;
do
{
         cout<<"\n "<<i;
i++;
}while ( i<=n ) ;
getche();
}
```

- *Output*

```
Enter any positive number : 3
1
2
3
```

5.3 Switch….case

- The switch statement provides a clean way to dispatch to different parts of a code based on the value of a single variable or expression.
- It is a multi-way decision making construct that allows choosing of a statement among several alternatives.
- The switch statement is mainly used to replace multiple if-else sequence which is hard to read and hard to maintain.

```
         switch (option)
         case 1:
                  statements;
                  break;
         case 2:
                  statements;
                  break;
                  .
                  .
                  .
         case n:
                  statements;
                  break;
         default:
                  statements;
                  break;
```

- In the above segment, if option 1, then the first case will be executed and the control will pass to the next statement after switch. Otherwise, the rest of the case statement will be evaluated in the same way. If none of them match, then the last case with the default will be executed.

```
#include<iostream.h>
{
    void main()
    {
            int option;
            cin>>option;

                    switch (option)
                    {
            case 1:
                    cout<<”today is Monday”
                    break;
            case 2:
                    cout<<”today is Tuesday”
                    break;
            case 3:
                    cout<<”today is Wednesday”
                    break;
            case 4:
                    cout<<”today is Thursday”
                    break;
            case 5:
                    cout<<”today is Friday”
                    break;
            case 6:
                    cout<<”today is Saturday”
                    break;
            case 7:
                    cout<<”today is Sunday”
                    break;
                    default:
                    cout<<”invalid no.”;
                    break;
            }
    }
}
```

5.4 Break & Continue

break statement:

- *Syntax*:
  break ;
- *Description*: A break statement terminates the execution of loop and the control is transferred to the statement immediately following the loop.
- A break statement is very useful in switch statement.
- *Example*: Following C++ program display square of given number and use 0(zero) for exit.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    int  a;
    clrscr();
    while ( 1 )
            {
            cout<<"\n Enter any number : ";
            cin>>a;
            if ( a==0)
                    break;
            cout<<"\n Square of a is : "<< a * a;
    }
    getche();
}
```

- *Output*:

```
Enter any number : 3
Square of a is : 9
Enter any number : 7
Square of a is : 49
Enter any number : 0
```

continue statement:

- *Syntax*:
  continue ;
- *Description*: A continue statement skips the remaining statement in the loop and control transfer to the condition.
- *Example*: Following C++ program display sum of only positive numbers.

```
#include<iostream.h>
#include<conio.h>
void main()
{
    int  a, total=0;
    clrscr();
    while ( 1 )
        {
        cout<<”\n Enter any number…0 for exit : “;
        cin>>a;
        if ( a==0)
            break;
        if ( a < 0)
        {
            cout<<“\n Negative number skipping…”:
            continue;
        }
        total = total + a;
    }
    cout<<“\n Total = ”<< total;
    getche();
}
```

• *Output*:

```
Enter any number…0 for exit : 20
Enter any number…0 for exit : 10
Enter any number…0 for exit : -7
Negative number skipping
Enter any number…0 for exit : 30
Enter any number…0 for exit : 0
Total = 60
```