C++

Ch. 01

Principles of

Object Oriented Programming, Tokens, Expressions and Control Statement,

What is Program?

- Simple meaning:
  - Program is a group of different things which are using for complete the work.
- Example:
  - Planning for Tour.
  - Party Arrangement.
  - Award function etc.

What is a Program?

- Computer meaning:
  - Program is a group of codes(variables, keywords, functions etc.) which help computer to generate output for user.
- Example:
  - You wish to show your name on the screen.
  - You wish to calculate addition of given values.
  - You wish to create a report card.

What is Programming?

- Programming means to arrange all the parts of program in proper order for meaningful purpose.
- Example:
  - Award function.
  - Annual function.
  - Party arrangement.
  - Lecture schedule.

What is Language?

- Language is a way of communication.
- Using any language we can send or receive information or command.

What is Programming Language?

- Programming Language is a platform where we can arrange particular command list to perform such task in the form of program.
- Example of programming languages like,
  - C,
  - C++,
  - C#.net,
  - VB.net etc…

Types of Programming Language.

- In human concept, there are two types of languages like:
  - Symbolic Language
  - Words Language
    - like, English, Gujarati, Hindi etc.
- Same as that there are two types of programming languages available in Computer :
  - POP
  - OOP

What is POP?

- POP stands for,
  - **P**rocedure
  - **O**riented
  - **P**rogramming
- Example of Procedure Oriented Programming Language
  - C
  - VB
  - FORTRAN
  - Pascal etc.

Procedure Oriented Programming

- **What is Procedure?**
  - The process which gives multiple works at a time is known as procedure.
  - In computer meaning:
    - Procedure means to execute all the code of program at a time.
  - Simple meaning:
    - Procedure is a block which can execute all the code of a task at a time.

Characteristics of POP :

- The main focus is on algorithms.
- Complex programs are divided into function
- Global data is shared by functions.
- There is no protection on data.
- It follows top-down approach.

Procedure Oriented Programming

- **Limitations of POP**
  - Time consuming because it works till starting to ending of process.
  - All code of program will get process at a time of execution when those codes are not necessary.

What is OOP?

- OOP stands for,
  - **O**bject
  - **O**riented
  - **P**rogramming
- Object oriented programming (OOP) is updated programming from Procedure Oriented Programming (POP).
- Example of Object Oriented Programming.
  - C++, JAVA, VB.NET, C#.NET etc.

OOP

- The object oriented programming approach came with major modifications in procedure oriented programming.
- First of all the data is protected in OOP.
- In OOP approach, the large and complex program is divided into **objects** in-place of function of POP.
- Each object is associated with its class and the data is accessible within the class.
- Any function from outside the class cannot access the data of Class.

OOP Characteristics :

- In OOP the importance is on **data** in place of function.
- Large programs are divided into **objects.**
- The data of objects is hidden and can’t be accessed by outside functions.
- Objects can pass information through functions.
- It follows Bottom-Up approach.

Top-Down Approach (POP):

- The Top-Down approach means the programmer stats the program with main function and then thinks of other steps to take such as functions and other coding.
- So the program execution also starts from top because of main function and goes down eventually is as follows.

Bottom-Up Approach (OOP):

- In Bottom-Up approach the programmer has to think for basic functionalities needed by program and then develop them.
- So after developing the functions and other structures the main function comes in last where all the functionalities will be used there.
- So the execution starts from bottom because the main function and goes upside to call other functions which are at top.

Difference between POP and OOP

POP	OOP
POP means Procedure Oriented Programming	OOP means Object Oriented Programming
Larger programs are divided into functions	Larger programs are divided into objects
In POP Functions are basic entity	In OOP Objects are basic entity
Main focus is on algorithms	Main focus is on objects

Difference between POP and OOP

POP	OOP
Functions share global data	Data is hidden and can’t be accessed by outside functions
Follows Top-Down approach	Follows Bottom-Up approach
Examples : C, Cobol, Fortran	Examples : C++, C#, Java etc

Features of OOP : (Basic Concept Of OOP)

- Objects
- Classes
- Data Abstraction
- Encapsulation
- Inheritance
- Polymorphism

Object :

- To know how can OOP work, we have to know that what is object.
- **Object**
  - In Simple meaning:
    - Generally all types of things are known as object.
    - Like: pen, book, apple, flower etc are known as objects.
- A one thing which is able to access more then one things when we want to use is known as object.

Object : Continue...

- Example:
  - Secretary of any company.
  - A peon of any college.
- In that above examples peon and secretary are the example of object who can handle any work which is required to execute.

Object : Continue...

- In computer meaning:
  - Object is a name which can access all the types of codes like variables, tokens, functions and structures.
- Example:

```
int a;
int b;
X is an object:
x.a;
x.b;
```

Class :

- In Simple meaning:
  - Class is a group which can store different types of things.
- Example:
  - There are many foods like,
    - Apple, Orange, Tomato, Potato etc.
  - So we have to share that things into two parts like,
    - **Fruits** and **Vegetables**
  - So **Fruit** and **Vegetables** are called as **Class** and Apple, Tomato are **Objects…**

Class : Continue...

- In computer meaning
  - Class is a group of different types of code like,
    - Variables, Functions, Tokens, Structures etc.
- Structure of class:

```
class A
{       Variables
        Functions
        Structures
};
```

Data Abstraction :

- Data abstraction is also known as data hiding.
- Data abstraction only shows some essential information and hide the background details such as how the functions work on data.

Encapsulation :

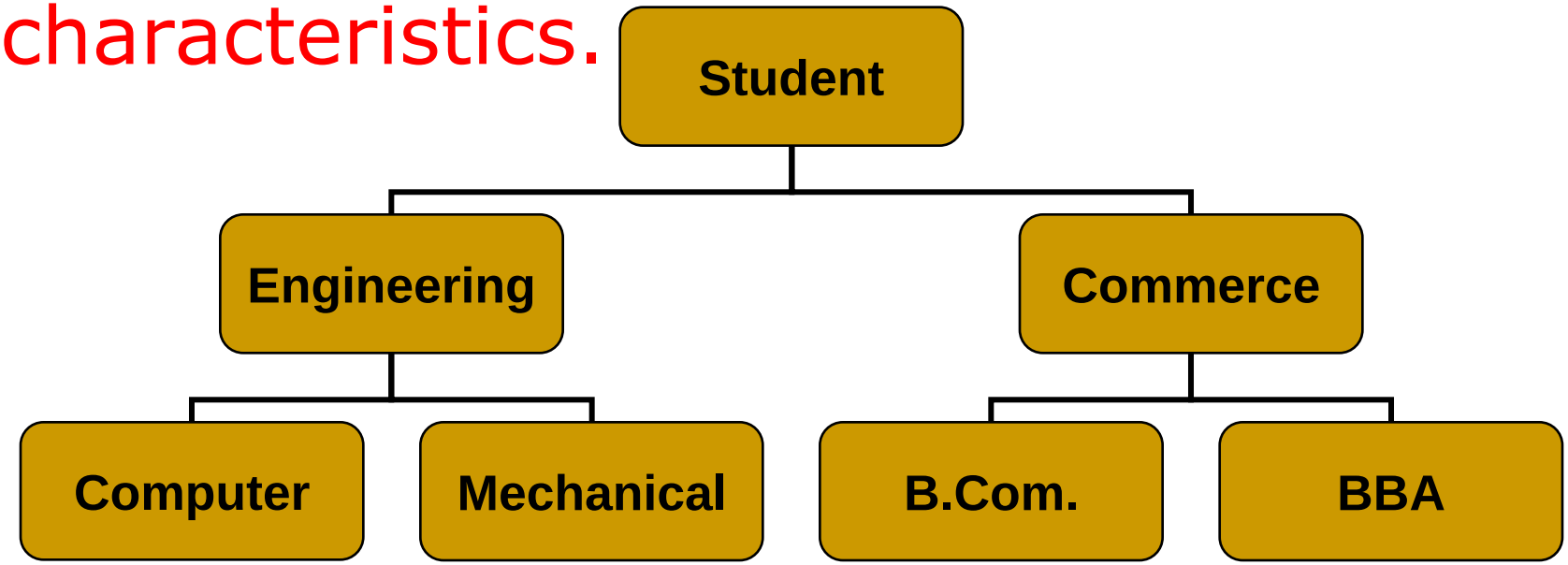
- Encapsulation word can be understood from the concept of medicine capsules in which the main element of the medicine is kept inside and is covered by the outside wrapper to protect it from the outside environment.
- Same way in OOP the data and functions are wrapped into a single unit that is called as class.
- The process of wrapping of data and functions is known as encapsulation.
- After this process the data is not accessible from the outside entities.

Inheritance :

- Inheritance is a concept in which a class uses the properties of another class acquired in heredity.
- In simple terms the son gets his father's properties derived in inheritance, same way a class can acquire properties of another class if it is permitted to do so.
- In short, The inheritance is a process in which object of one class gains some or all properties of another class.

Inheritance :

- Inheritance can have two main benefits:
  - It provides hierarchical classification to your program.
    - Hierarchical classification means you can classify the objects with similar characteristics.



Inheritance :

- Inheritance can have two main benefits:
  - Reusability
    - Reusability is a concept of "write once, use many times".
    - In inheritance you can use the properties of a class in its derived classes also.

Polymorphism :

- Polymorphism is a Greek term which means the ability to take more than one forms by a single entity.
- In OOP polymorphism plays an important role.
- In OOP polymorphism can be achieved by many ways such as function overloading, operator overloading, constructor overloading etc.

Polymorphism :

- Polymorphism is as a single person who can behave differently in different situations.
- In polymorphism a single function performs different operations when you pass different argument.
- Example
  - + operator. It perform addition if two numeric values passed (2+3=5).
  - It perform concatenation if to string passed (“abc” + “xyz” = “abcxyz”).

Benefits of OOP :

- It saves program development time and gives higher productivity.
- Repeated code can be removed using inheritance.
- Secure programs can be built using data hiding.
- Large program can be partitioned based on objects.
- Complexity can be reduced.
- The system developed under object oriented model can be easily upgraded.

Application of OOP :

- OOP can be very useful in building real time applications.
- Like,
  - Uses to develop Real-time applications such as medical software, stock management etc.
  - Artificial Intelligence and expert systems.
  - Office automation system.
  - CAD-CAM system.
    - (CAD=computer-aided design)
    - (CAM=computer-aided manufacturing)
  - Object oriented database system.

What is C++ ?

- Basically C++ is not a new language but it is just and updation to C programming language.
- C++ is an object oriented programming language.
- C++ is **developed by Bjarne Stroustrup at AT&T Bell Laboratories in New Jersey, USA** in the early **1980s**.
- Earlier the name of the language was "C with classes" but after sometimes it was renamed to C++.
- ++ is an increment operator of C language.
- Here, C++ is an increment to the C language.

Application of C++

- C++ is a dynamic language and capable to creating any kind of large program.
- Using C++ we can develop complex programs including compiler, editor, database system or any real time application.
- Using C,
  - Building CAD (Computer Aided Design) system for surveying in Civil Engineering.
  - Photos editing software like, Photoshop, Acrobat etc. are developed in C++

Application of C++

- Developing Operating System is possible.
- Almost all applications that can be built using C can be developed in C++ also
- C++ programs are easy to maintain.

Structure of C

- First we revise the structure of C programming before understand structure of C++.
- **Structure of C Programming**
  1. Documentation.
  2. File Include Section.
  3. Symbolic Constant Definition
  4. Global Variable Declaration
  5. Main Function.
  6. User Defined Function.

Structure of C++

- **Structure of C++ Programming**
  1. Documentation.
  2. File Include Section.
  3. Global Variable Declaration.
  4. Class Declaration.
  5. Function Definition Section.
  6. Main Function.

Output Operator :

- Output Data Using **cout**
- In C++ we can use **cout operator** to display data on screen.
- The output operator (<<) is also known as insertion operator because it inserts the text or value of the variable passed with it.

Syntax : cout<<"string" or variable;

Example : cout<<"MONARCH Computer";

: cout<<val1;

Input Operator :

- Input Data Using **cin**
- In C++ we can use **cin operator** to enter some values to a variable.
- The input operator (>>) is also known as extraction operator because it extracts the text or value to the variable.

Syntax : `cin>>variable1>>variable2;`

Example : `cin>>a;`

: `cin>>a>>b;`

Introduction to Namespace

- To understand the concept of namespace, consider a situation in which you need to create to classes with same name.
- In any operating system we cannot create two files with same name in same folder.
- If we have to create two files with same name then we have to create those files in different folders.
- The namespace is the same concept to the folder.
- Namespace is used to resolve the name clash problem in your program.

Introduction to Namespace

- Namespace is just a collection of classes.
- We can add our classes, functions, variables and related code in a namespace to create a useful container of related code.
- Now we can create another namespace and add classes, functions or variables with same name of the old namespace without any clash
- To create a namespace we have to use **namespace** keyword

Introduction to Namespace

- Syntax of Namespace

```
namespace namespace_name
{
  // required statements…
}
```

- To use classes, functions or variable from name space we have to include a statement saying that you are using that namespace in your program.

Syntax : using namespace **namespace_name**

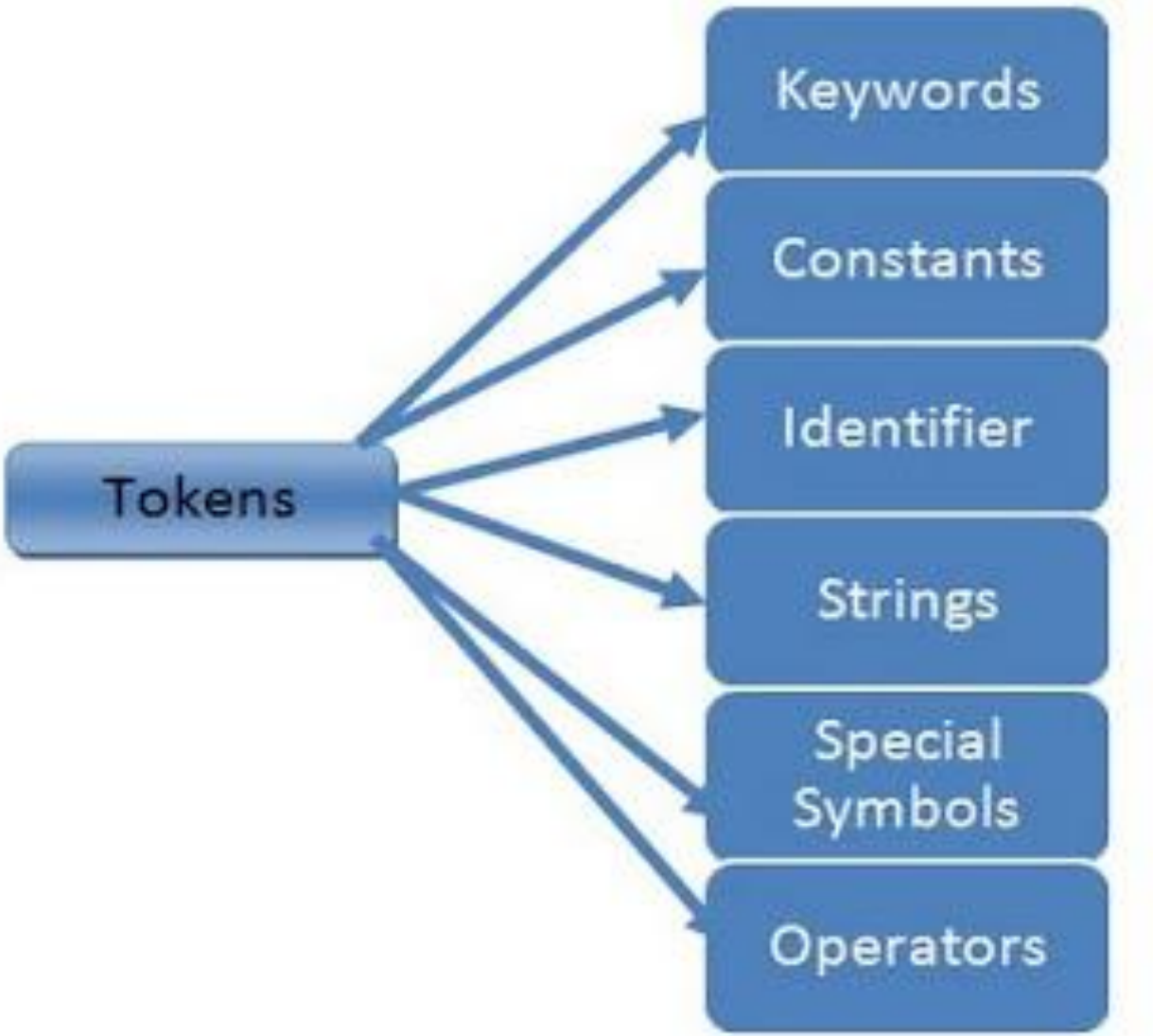
Introduction to Namespace

- NOTE :
  - The namespace concept will not run under Turbo C++ compiler.
  - We can use namespace in advanced C++ editor such as Microsoft Visual C++.

Tokens

- Tokens are the building blocks of the C++ program.
- Every single CHARACTER of C++ program file can be considered as a C++ token.
- A token is a group of characters that logically belong together. The programmer can write a program by using tokens

Types of Token.



Keywords...

- Keyword are predefined reserved identifiers that have special meanings.
- They cannot be used as user define identifiers like, variable name, function name, class name in your program.
- Following is the list of Keywords...

asm	insert an assembly instruction
auto	declare a local variable
break	break out of a loop
case	a block of code in a switch statement

Keywords...

catch	handles exceptions from throw
char	declare a character variable
class	declare a class
const	declare immutable data or functions that do not change data
continue	bypass iterations of a loop
default	default handler in a case statement
delete	make memory available
do	looping construct

Keywords...

double	declare a double precision floating-point variable
else	alternate case for an if statement
enum	create enumeration types

Keywords...

extern	tell the compiler about variables defined elsewhere
float	declare a floating-point variable
for	looping construct
friend	grant non-member function access to private data
goto	jump to a different part of the program
if	execute code based off of the result of a test
inline	optimize calls to short functions

Keywords...

int	declare a integer variable
long	declare a long integer variable
namespace	partition the global namespace by defining a scope
new	allocate dynamic memory for a new variable
operator	create overloaded operator functions
private	declare private members of a class
protected	declare protected members of a class

Keywords...

public	declare public members of a class
register	request that a variable be optimized for speed
return	return from a function
short	declare a short integer variable
signed	modify variable type declarations
sizeof	return the size of a variable or type
static	create permanent storage for a variable

Keywords...

struct	define a new structure
switch	execute code based off of different possible values for a variable
template	create generic functions
this	a pointer to the current object
throw	throws an exception
try	execute code that can throw an exception

Keywords...

typedef	create a new type name from an existing type
union	a structure that assigns multiple variables to the same memory location
unsigned	declare an unsigned integer variable
using	import complete or partial namespaces into the current scope

Keywords...

virtual	create a function that can be overridden by a derived class
void	declare functions or data with no associated data type
volatile	warn the compiler about variables that can be modified unexpectedly
while	looping construct

Identifiers ...

- An identifier is a name given to a variable, function, class, object, structure, union, enumeration. Identifier is a label to identify it in the program.
- We can give any name as an identifier but there are some rules to declare an identifier.
  - Keywords cannot used as an identifier.
  - White space is not allowed.
  - No any special symbol except underscore (_) is allowed.
  - Digits are allowed but identifier should not start with a digit.
  - Uppercase and lowercase letters are different.

Constants :

- As the name suggests, constant are fixed values that should not be changed throughout the program execution.
- Constants can be integer, floating point, characters or strings.
- For Example,
  - Integer Constant : 123, 555, 2569 etc
  - Floating Point Const : 1.23, 40.10, 1.75 etc
  - Character Constant : ‘a’, ‘b’, ‘x’, ‘Y’ etc
  - String Constant : “ABC”, “Computer” etc

String :

- String are the words you write in the program such as in printing message or in string constants.

Special Symbol :

- Special Symbols are those characters which are used in the program to perform various tasks.

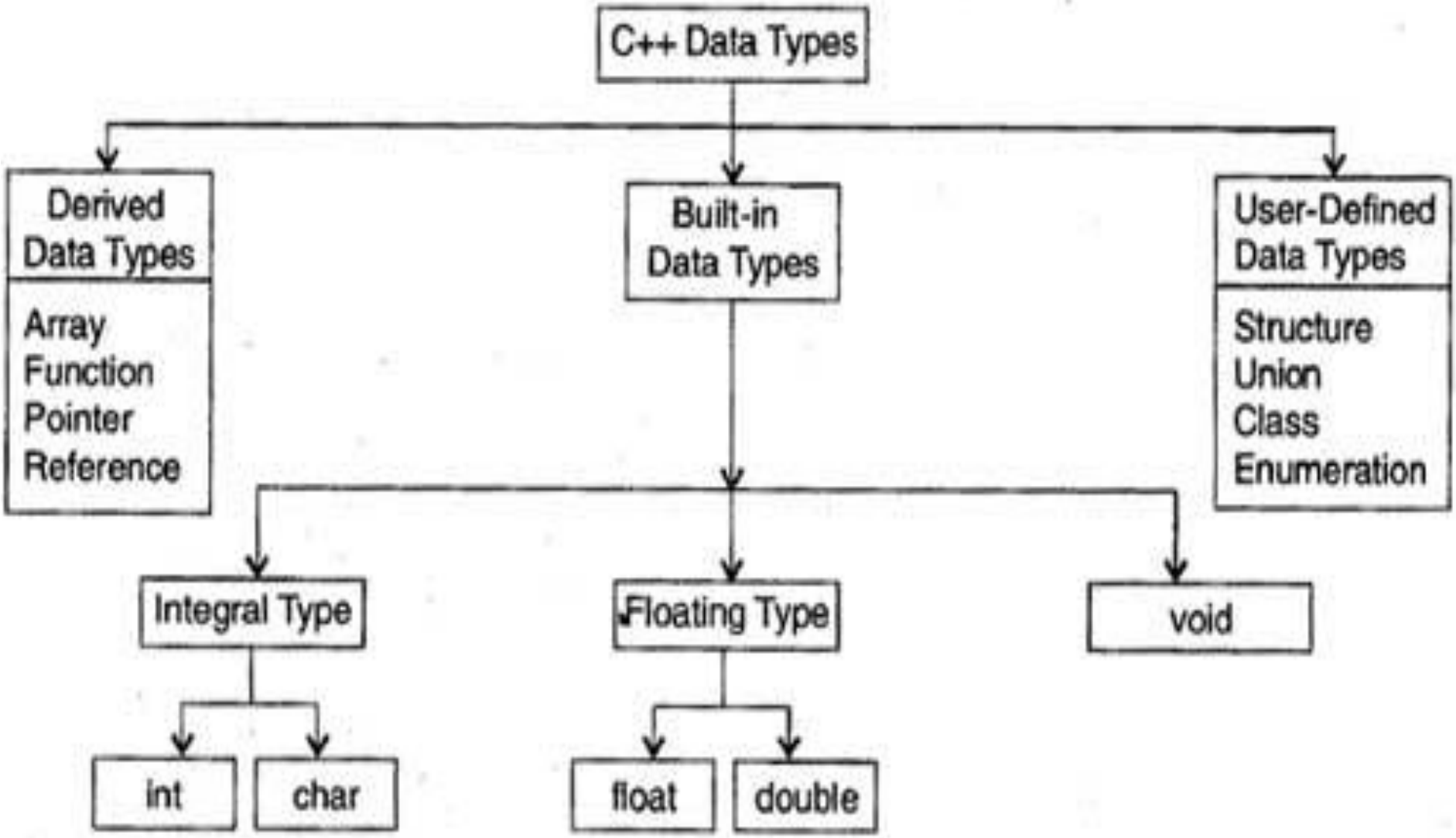
Operators :

- In the program, we can use operators in expressions such as : **c=a+b**
- Here, **+** and **=** both are operators

Data Types in C++ :

- We always use different types of data in our programming.
- We have various types of data. Like, Sr_No, Name, Weight, age etc.
- All data has its own data type.
- Data types means type of data to store in specified data type.
- For example,
  - We can store **Sr_No** in **Integer Data Type**
  - We can store **Weight** in **Floating Data Type**

Data Types in C++ :



Data Types in C++ :

- There are main three categories of data types.
  - Basic Data Types (Built-in)
  - User Defined Data Types
  - Derived Data Types
- Basic Data Types :
  - Basic Data Types, also known as fundamental or **built in data types** of C++ are
    - Integer
    - Floating Point
    - Void

Data Types in C++ :

- Basic Data Types :
- Integer :
  - The variables of floating point data types can store the numeric values without any fractional part.
    - Char, int, long are main integer
- Floating point :
  - The variable of floating point data types can store the numeric values with fractional part.
    - Float, double, long double are floating point data type.

Data Types in C++ :

- Basic Data Types :
- Void :
  - The void data type is follows its name, it doesn't store any value.
  - But it has mainly two uses
    - First to specify the return type of a function.
    - Second use is to declare a generic pointer. Generic pointer is a pointer variable which can point any other type of pointer variable.
      - Example : void ***gp**;

Data Types in C++ :

- Basic Data Types :

Data Type	Size in Bytes	Range
char	1	-128 to 127
unsigned char	1	0 to 255 (ascii)
int	2	-32768 to 32767
unsigned int	2	0 to 65535
long	4	-2147483648 to 2147483674
unsigned long	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 1.1 E 4932

Data Types in C++ :

- User Define Data Types :
  - Structure
  - Unions
  - Class
  - Enumerations *are user define data types*.

Data Types in C++ :

- User Define Data Types :
  - Structure & Union
    - Structure and union are user defined data type as they are defined by user.
  - How to define a structure ?

    Syntax :

```
struct <structure name>
{
        variable list;
}
```

Data Types in C++ :

- User Define Data Types :
  - Classes
    - Class is another user defined data type of C++. It offers some more functionalities than a structure.

Data Types in C++ :

- User Define Data Types :
- Enumerations
  - Using enumeration, we can enumerate a list of values. It assign 0 to first value, 1 to second value and so on.
  - Example

    enum fruits{apple, mango, grapes};

    Here, apple will be assigned with **0**

    mango **1** and grapes **2**
    - **We can also assign specified numer to particular item to…**

Enum week{mon=1, tue, wed, thu, fri, sat, sun};

Data Types in C++ :

- Derived Data Types :
  - Array, pointers, functions and reference variable are considered as derived data types. **Derived Data Types** are derived from the basic data types.

Data Types in C++ :

- Symbolic Constants :
  - In C++ we can create symbolic constant by many ways :
    - Using the **const** keyword
    - Using **enum** keyword
    - Using **#define** directive…
  - Constant cannot redefine during program execution.

Type Compatibility :

- Type Compatibility :
  - Type compatibility refers to whether the types of all variables are compatible to one another or not when written in an expression.
  - For example :
    - 3 variable of different data types in an expression must be compatible to each other.

Type Compatibility :

- Type Compatibility :
  - Example :

```
int a=10;
float b=20;
double c;
c=a+b;
```

  - Here, in expression c=a+b, a(int) and b(float) and c(double) all are different but it executes program without error because of compatibility.

Declaration of Variables :

- In C++ we have to declare all the variables before using them.
- In C, we have to declare all the variables at the beginning of the program.
- In C++, we can declare any variable when it is needed in any line.

Dynamic Initialization of Variable :

- As the declaration of variables, all the variables must be initialized before using them.
- In C++ we can initialize your variables dynamically (runtime) also.
- So the variable will get its initial value when you run the program.

Dynamic Initialization of Variable :

```
void main()
{ int number;
  cout<<"Enter Number :";
  cin>>number;
  double range=number+1;//Dynamic initialization
  for(int x=1;x<=number;x++)
  {  cout<<x<<endl;  }
}
```

Reference Variable :

- In C++ we will learn a new kind of variable known as reference variable which is very useful in programming.
- It provides an alternate name to your variable that can be used for your old variable.
- We can use both variable to perform common effect to variable values.
- Syntax :

```
Data_Types& reference_verialbe=Variable;
```

Reference Variable :

- Example :

```
int a=10;
int& b=a;
cout<<"Value of A = "<<a;
cout<<"Value of B = "<<b;
```

- What is Operator?
  - The symbols which are used to perform mathematical or logical operation those symbols are known as operator.
  - Like,
    - +, -, *, /, &&, ||, !=, <, >, etc.
- What is Operand?
  - Operand means the variable or value which is used before and after the operator.
  - Like
    - A+B (Here, **A** and **B** are operand and **+** is operator)

Operators

(1) Arithmetic Operator

(2) Relational Operator

(3) Logical Operator

(4) Assignment Operator

(5) Increment / Decrement Operator

(6) Conditional Operator

(7) Bitwise Operator

(8) Special Operator

Arithmetic Operator

- To make arithmetic processing we can use arithmetic operators.

Operator	Purpose
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Reminder after integer division

Relational Operator

- To check relationship between two variable we can use different relational operators.

Operator	Meaning
<	Less then
<=	Less then or Equal to
>	Greater then
>=	Greater then or Equal to
= =	Equal to
!=	Not Equal to

Relational Operator

- For Example,

  a=1, b=2, c=3;

Expression	Interpretation	Logical Value
a<b	True	1
(a+b)>c	False	0
(a+b)>=c	True	1
(a+b)=c	True	1
(a+b)<c	False	0
(a+b)<=c	True	1
c!=3	False	0
c==3	True	1

Logical Operator

- To provide such logical operations in C we can use logical operators. As given below :

Operator	Meaning
&&	AND
||	OR
!	NOR

Logical Operator

- **Logic for && :** While all the conditions will be true using && then logical output will be true else output will be false.

Truth Table for &&

Input A		Input B	Output
True	&&	False	False
False	&&	True	False
False	&&	False	False
True	&&	True	True

Logical Operator

- **Logic for || :** While any of the condition will be true using || then logical output will be true else output will be false.

Truth Table for ||

Input A		Input B	Output
True	||	False	True
False	||	True	True
True	||	True	True
False	||	False	False

Logical Operator

- **Logic for ! :** While any of the condition will be true using || then logical output will be false else output will be true.

Truth Table for !

Input A	Output
True	False
False	True

Conditional Operator

- C programming offers the conditional operator used in C language is (?) question mark and (:) column.
- It is used for checking the condition and gives output according to the condition.
- If condition is true then it will execute the first value or it will execute the second value given in the syntax.

Conditional Operator

- Syntax :

  <condition>?<True Value>:<False Value>

- Purpose:
  - If condition will be true then it will return the true value else it will return the false value.

Conditional Operator

- Example:

```
#include<iostream.h>
void main()
{   int a,b,c;
    a=10;
    b=12;
    c=(a>b)?a:b;  // It will store value of B
    cout<<"Value of C is "<<c;
}
```

Assignment Operator

- Assignment operators are use to assign any value to any data to any identifier. The most commonly sign used as assignment operator is = (equal).
- Identifier = Expression

```
a=10                    a=a-1
a=a-10                  a-=1
a= (a*2) +a             a=a*a
a=a+1                   a*=a
a+=1                    a=a/2    or   a/=2
```

Sizeof Operator

- To check the size of variable in bytes we can use sizeof operator.
- For Example,

```
int a=150;
float b=25000;
cout<<“Size of A is "<<sizeof(a);
printf<<“Size of B :%d”<<sizeof(b);
```

Increment/Decrement Operator

- C++ programming offers two special operator ++ and -- called as increment and decrement operators.
- There are “unary” operators are only able to work with proper variable. It will not work on constant or numeric value.
- Here a++ is valid but 6++ is invalid. These operators can be used either before or after their operand. Like a++ is valid and ++a is also valid.

Increment/Decrement Operator

- For Example,

```
a=10;
a++;    // now a =11
- -a;    // now a=10
++a;    // now a=11
a- -;    // now a=10
```

Bitwise Operator

- Bitwise operators are used to perform bit manipulations.

  Generally these operators used to perform hardware related task by c++ programming.

Operators	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	Shift Left
>>	Shift Right

Special Operator

- C++ language supports many special operators too. They are used for some special processes in C++ programming. T
- he special operators used in C++ programming language are,
  - Comma (,)
  - Pointer Operators (& and *)
  - Member Selection (. and ->)
  - sizeof( ) operator

Hierarchy of Operators

1. First evaluate parentheses expression.
2. If parenthesized are nested, the evaluation begins with the innermost sub-expression.
3. Arithmetic operators without parentheses will be evaluated from left to right using the rules of operators.
4. There are two distinct priority levels of arithmetic operators.

   High priority : * / %

   Low priority : + -

New Operators in C++

- C++ contains number of useful and powerful operators.
- The most operators and common to the operators in C.
- Now discussing the operators new to C++

Sr. No	Operator	Meaning
1	<<	Insertion Operator
2	>>	Extraction Operator

New Operators in C++

Sr. No	Operator	Meaning
3	::	Scope Resolution Operator
4	::*	Pointer to member declaration operator
5	->*	Pointer to member access operator
6	.*	Pointer to member access operator

New Operators in C++

Sr. No	Operator	Meaning
7	new	Memory Allocation operator
8	delete	Memory release operator
9	endl	New line operator
10	setw	Field width operator

Scope Resolution Operator (::)

- In a program, we can have same variable names in different blocks.
- The variable of outer block can be access by inner block.
- The variables declared in outer block are known as the global variable and the variables declared in the inner block are referred as local variables.
- Thus the local and global variables define their scopes regarding how long they are visible.

Scope Resolution Operator (::)

- You can use scope resolution operator to access the global variable in C++.

Syntax :

```
::variable_name;
```

Example :

```
int a=111;
void main()
{       int a=10;
        cout<<"Local A ="<<a;
        cout<<"Global A ="<<::a;
}
```

Member Referencing Operators:

- The member reference operators (::*, * and ->*) are used to declare and access the pointer to members.

Memory Management Operators:

- In C, we used malloc() and calloc() function for dynamic memory allocation and the free function to release the allocated memory.

Memory Management Operators:

- Dynamic memory allocation is used when you do not know how much memory will require in advance.
- C++ also allows these functions for dynamic memory allocation but it introduces two operators:
- **new** and **delete** for dynamic memory allocation and deallocation. These operators are contained in header file "iostream.h".
- The new operator is used to create a new object and delete operator is used to destroy the object.

Memory Management Operators:

- The object created by new operator remains in existence until it is deleted by the delete operator.
- Syntax :

  data type *pointer_variable=new data_type;
- Example

  int *p=new int;

  double *x=new double;
- The new operator allocates the required memory and returns the address of the created object.

Memory Management Operators:

- We can also initialize the memory using the new operator.
- Syntax :

  pinter_variable=new data_type(value);
- Example :

  int *p1=new int(100);

  double *p2=new double(1.2);
- We can also allocate memory to an array using new operator as follows:

  pointer_variable= new data_type[size];
- Example : int *a=new int[5];

Memory Management Operators:

- Advantages of using **new** operator over malloc() function:
  - Automatically calculates the size of object, so no need to use sizeof().
  - Automatically returns the correct pointer type, so no need of type casting.
  - Object can be initilized at the time of allocating memory.
  - The **new** and **delete** operators can be overloaded.

Manipulators :

- C++ provides some manipulators that are used for formatting output.
- The commonly used manipulators are **endl** and **setw.**
  - **Endl (end line)**
    - The **endl** manipulator is used to insert a new line. It works similar to "\n".
    - But as it is a manipulator, **it is not included in double quotes**.
    - **Example**

      cout<<"My Name"<<endl<<"Monarch";

Manipulators :

- **setw (set width)**
  - The setw manipulator is used to specify the width of variable to be printed in cout. **(**Note : **iomanip.h** must include**)**
  - Syntax :

    setw(int width);
  - Example :

```
void main()
{ int a=10, b=22020;
  cout<<"A="<<setw(6)<<a<<endl;
  cout<<"B="<<setw(6)<<b<<endl; }
```

Output

Type Cast Operators :

- Type casting is a process which converts a value from one data type into another data type.
- During the programming we can convert integer value to float value and float value to integer value as per requirement using type cast operator

Example :

```
int a=20;
cout<<(float)a/3); //  It will convert int to float
```

Expressions and Their Types :

- In programming we may need to use expressions containing variables, constants and operators or sometimes functions calls also.
- Expressions can be one of the following :
  - **Integer Expressions**
  - **Float Expressions**
  - **Pointer Expressions**
  - **Relational Expressions**
  - **Logical Expressions**
  - **Bitwise Expressions**
  - **Constant Expressions**

Expressions and Their Types :

- Integer Expressions :
  - The integer expressions return the integer values after completing.
  - Example

```
int a=10, b=20, c;
c=a+b;
```

- Float Expression :
  - The float expression return the floating point results:
  - Example

```
float a=1.5, b=1.75, c;
c=a+b*0.5;
```

Expressions and Their Types :

- Pointer Expression :
  - The pointer expression returning address of some variable.
  - Example :

```
int a, *p;
p=&a;
```

- Relational Expression :
  - The relational expressions contain comparisons or conditions and return true or false type values.
  - Example :

```
if(a>b)
```

Expressions and Their Types :

- Logical Expression :
  - The logical expressions combining relational expressions.
  - Example :

    ```
    if(a>b && a>c)
    ```
- Bitwise Expression :
  - The bitwise expressions containing bitwise operators.
  - Example :

    ```
    a=x>>2;
    x=a & b;
    ```

Expressions and Their Types :

- Constant Expressions :
  - The constant expressions containing only constant values.
  - Example :

    a=2+4*5/2;
- Special Assignment Expressions :
  - You can use more than one assignment operator in an expression to produce a chained is known as assignment expressions.
  - Example :

    a=b=c=10;

Implicit Conversion :

- Whenever in an expression, the variables or operands of different types are combined, then the automatic type conversion takes place.
- Automatic type conversion is known as the implicit type conversion.
- Example :

```
a=2+3.2*1.2;
```

Operator Precedence :

- Operator precedence plays an important role in an expression when there are different operators are used.
- Example :

ans=10+20*3+6/2-1

Select Control Structure :

- If Statement
  - We can write conditional statement IF in different ways...
    - if (Wimple if Without Else Statement)
    - if....else
    - if....else if ladder
    - Nested if (if within if)
- Switch Statement

if (Simple if without else)

Syntax :

```
if(condition)
{
    statements;
}
```

- Purpose:
  - If statement is the decision control statement. This statement will check the given condition. *If condition is TURE then it execute given statements.*

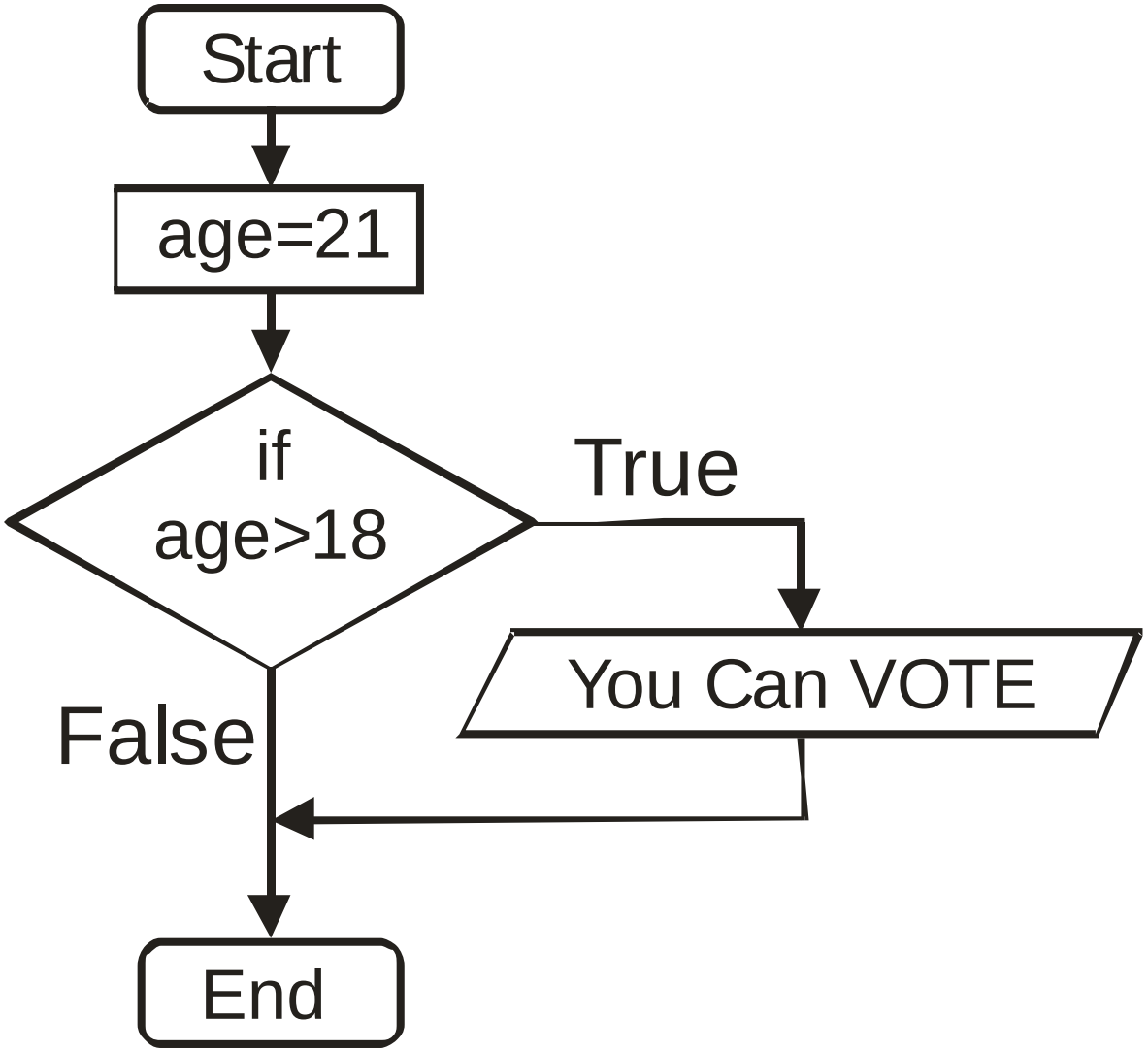
if (Simple if without else)

Example: To check that you can VOTE or Not

```
#include<iostream.h>
#include<conio.h>
void main( )
{
  int age;
  cin<<"Enter Your Age : ";
  cout>>age;
  if(age>18)
        cout<<"You can VOTE";
}
```

if (Simple if without else) Flowchart

Example: To check that you can VOTE or Not



if....else

Syntax : if(condition)

statements;

else

statements2;

- Purpose:
  - If statement is the decision control statement. This statement will check the given condition. If condition is TURE then it execute given statement or execute other statement.

if ... else

Example: To check that you can VOTE or Not

```
void main( )
{       int age;
        cin<<"Enter Your Age : ";
        cout>>age;
        if(age>18)
                cout<<"You can VOTE";
        else
                cout<<"You can not VOTE";
}
```

if....else if ladder

Syntax :

```
if(condition)
        statement block1;
else if(condition)
        statement block2;
else
        statement block3;
```

- Purpose:
  - You can use else…if ladder to execute a code based on multiple conditions. Here each condition is tested one after another. If condition is true then it execute the statement block and terminate the ladder.

Nested if statement...

Syntax :
```
if(condition1)
{       if(condition2)
        {
                //statement block1
        }
}
else
{
        //statement block2
}
```

- Purpose:
  - You can nest multiple if blocks to test the one condition based on another condition. If condition1 is true then it checks the second condition.

Switch Statement :

Syntax : **switch(**variable or expression**)**

```
{       case value1:
                statements;
                break;
                .......
        case valueN:
                statements;
                break;
        default:
                default_statements;
}
```

- Purpose:
  - When we have relatively more number of conditions at that time we can use switch.

Def. Switch...

- Write a program to enter day number of week day and print weekday name.

Output :

Enter Day Number : 5

Thursday

Looping Statements :

- Loop means...
  - In computer **programming**, a **loop** is a sequence of instructions that is continue repeated until a specified condition is satisfied.
  - Loop is a concept to reduce repeated code.
- Looping can be divided in two type :
  - Entry Control Loop
    - for & while
  - Exit Control Loop
    - do...while

For Loop :

- For loop is the simplest loop and becomes very famous.
- It is entry-controlled loop, because the condition is tested before entering the block.
- Syntax :

```
for(initialization;condition;incr/decrement)
{
    //Block of Statements
}
```

While Loop :

- While is a simple loop which just tests a condition and if the condition is true the block executed again.
- While loop is an entry-controlled loop as the condition is checked at the time of entering the block.
- Syntax :

```
while(condition)
{
    //Block of Statements
}
```

do...while Loop :

- Do while loop is an exit-controlled loop. It first executes the block and then tests condition.
- If the condition is true, the block of code is executed again.
- Syntax :

```
do
{
    //Block of Statements
} while(condition);
```

- **Functions in C++**

Introduction...

- A function is a block of statement that performs some specific task.
- For Example,
  - clrscr() function clears the screen
- Purpose of using functions :
  - To divide a large and complex program into small blocks so that it becomes more readable.
  - Increase the reusability of program code. Once a function is defined, we can use it number of times without rewriting codes.

Working of function : (Part of Function)

- Function Declaration

  (Also known as Function Prototype)
  - Function declaration specifies the structure of the function. It tells the compiler the name of the function, number and type of arguments and return type of function.
- Function Definition
  - Function definition defines what the function will be called.
- Function Call
  - Function call is the statement which calls the function.

The main() function :

- Same as the main() function of C, the main() function in C++ is the entry point of the program execution.
- Whenever a program is going to run, C++ calls the main() function and the statements written in it will be started to execute.
- Generally we can define return type of main().
- Whenever return type is defined at that time return statement must used, else it will generate an error :

  Function Should Return a Value

The Function Prototype :

- In C++, the function must be declared before it's called.
- The function declaration is also known as the function prototyping.
- The function prototype specifies the structure of function. It tells the compiler about function name, function return type and number of types of function argument.
- Syntax :

  Return_Type *Function Name*(Argument List)
- Example : void test();

Call by Reference :

- Whenever a function is called, its arguments are passed to the function definition.
- In C++ we can use concept of reference variables.
- We can use reference variable to call a function by reference.
- When we pass arguments by reference, its formal arguments in the function definition become the aliases of the actual argument of the calling function. So any changes made on the formal arguments will reflect on the actual arguments.

Call by Reference :

- Call by reference is necessary when a function returns multiple values.
- For example a function swapping two variables' values.

```
#include<iostream.h>
void swap(int &, int &); //Function Declaration
void main()
{ int a=10, b=20;
  cout<<"\nValue Before SWAPPING\n";
  cout<<"A="<<a<<"\nB="<<b;
  swap(a,b);
  cout<<"\nValue AFTER SWAPPING\n";
  cout<<"A="<<a<<"\nB="<<b; }
void swap(int &x, int &y)
{ int temp;     temp=x;    x=y;     y=temp; }
```

Difference : Call by [Value / Reference]

Call by Value	Call by Reference
Changes made on arguments will not affect to the original values.	Changes made on arguments will affect the original values.
Cannot return multiple values	Multiple values can be returned
Normal variables are used	Reference variables are used.
Example : Any normal function	**Example :** Swapping, Bubble sort

Inline Functions :

- Whenever a function is called; the control is transferred to the function definition, saves the registers, inserts the arguments into the stack and finally after executing all the functions statements, returns to the calling function.
- A function definitely saves time but just imaging what happens if a small function is called so many times…?
- All the functions calling time will be spent on such overheads resulting reduction in performance.

Inline Functions :

- The inline function is a solution of repeating of small size of udf.
- Inline functions is a small function that is expanded in line when it is called.
- Syntax :

  Inline return_type function_name(arguments)

  { //statements }
- Inline functions increase the speed of program execution as the function call and return is not there but the program occupies more memory because the inline function definition is expanded each time when the function is called.

Inline Functions : Limitations

- There are some limitations to use inline function where it may not work as listed below:

1. If the function contains a loop
2. If the function contains goto or switch
3. If the function contains static variable
4. If the function is not returning value and a return statement exists
5. If the function is recursive

Default Arguments :

- In a function we have some arguments with values.
- When we have not argument values at that time it generates error. We can have some default argument values.
- These arguments are known as default arguments and the function is known as the default argument function.
- The default arguments are specified at the time of function declaration.
- When the compiler sees a function prototype it checks for any default arguments and prepare itself for default value.

Default Argument :

- Rules for default arguments :
  - We can use any number of arguments.
  - The default arguments must be the trailing argument in the argument list. (Means we cannot have default argument at starting or at middle in the argument list.
  - Example

    void sum(int x, int y, int z=10); Valid

    void sum(int x, int y=5, int z); Invalid

Const Arguments :

- If we do not want to allow the function to modify its argument, then we can have constant arguments in our function.
- So when we declare a function's argument by const keyword, we will not be able to modify its argument in function definition.
- If we try to change value of constant argument then the compiler will give an error…

  Connot modify a const object

Function Overloading :

- A same function name can be used for performing different task that method is known as function overloading.
- In a class if there is more than one function with same name but different type of arguments then this function is known as overloaded function and this process is known as **function overloading.**
- For example :

```
void test(int x, int y);
void test(int x, int y, int z);
void test(int x, float y, char z);
void test(int a, float b);
```

Function Overloading :

- When an overloaded function is called, the compiler matches the prototype matching the same number and type of arguments.
- If exact match is not found, the compiler tries to convert the arguments implicitly. For example, int is converted to float; float is converted to double.
- If compiler finds multiple matches for a function call, it will give an error message.
- If all of the above steps fail, the compiler will try the user-defined conversions to find a suitable match.

Adding C functions in Turbo C++

- In Turbo C++, you can use C functions also.
- We should have the knowledge of which header file consists the function.
- Example...
  - To use functions like **printf()** and **scanf()** you have to include **stdio.h** header file.

```
int main()
{       printf("This is C function");
        cout<<"\nThis is C++ function";
        return 0;
}
```