Header Files

HeaderFile	Functions, Objects and Member Functions

`iostream`	**`Objects: cin,cout`**
	`Member functions:`
	`read(),write(),getline(),get(),put()`

`fstream`	**`Member functions:`**
	`open(),close(),read(),write(),getline(),get(), put(),eof()`

`stdio`	**`gets(),puts()`**
`stdlib`	**`randomize(),random(), itoa(), atoi()`**
`iomanip`	**`setw()`**
`string`	**`strlen(),strcpy(),strcat(),strcmp(),strcmpi(), strupr(),strlwr(),strrev()`**
`ctype`	**`isupper(),islower(),isalnum(),isdigit(), isalpha(),toupper(), tolower()`**
`math`	**`sin(),cos(),exp(),frexp(),log(),abs(),fabs(), sqrt(),pow()`**

Importance of main()

`main()` function is the first function to be executed in the program.
All the remaining functions in the program are either called from **`main()`** or from the functions, which are called by **`main()`** function.

C++ Data Type

Name	Description	Size*	Range*
`char`	Character or small integer	1 byte	signed: -128 to 127 unsigned: 0 to 255

`short`
`int (short)`
	Short Integer	2 bytes	signed: -32768 to 32767
			unsigned: 0 to 65535

`int`	Integer	2 bytes	signed: -32768 to 32767 unsigned: 0 to 65535
`long int (long)`	Long integer	4 bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
`float`	Floating-point number (Real Number)	4 bytes	-3.4e38 to +3.4e38
`double`	Double precision floating point number	8 bytes	-1.7e308 to +1.7e308
`long double`	Long double precision floating point number	10 bytes	-1.1e4932 to +1.1e4932

Type modifiers

Type modifiers are used to modify range and/or size of the data type.
short, long, unsigned, signed are type modifiers in C++

Example:

```
  long double Regno;//Regno occupies 2+8=10 bytes
  unsigned int RollNo;//Rollno occupies 2 bytes [range 0..65335]
  signed char Temp;//Temp occupies 1 byte[range -128..127]
```

To store temperature

Access Modifier

const is an access modifier in C++. It is used to declare an identifier, whose value will remain same throughout the program.

Example:

```
      const int MAX=90;
```

Run-time error

A run-time error occurs during the execution of the program, when the program performs an illegal/unexpected operation.

Example:

```
      int a,b,c;
      cin>>a>>b;
      c=a/b;//will result in Run-time error if b entered as 0
```

Syntax Error

A syntax error occurs when the compiler is unable to translate the program to machine language due to violation of rules of the programming language.

Example: (In C++, condition in a if statement not enclosed in brackets)

```
      if X>90
        cout<<X<<endl;
```

Logical Error

A logical error occurs when the program contains wrong formula or wrong calculation, which may be syntactically correct. The program having logical errors may give some output but not the expected one.

Example:

```
      //The formula used for calculating average
      //of five subject's marks as
      Ave= Eng+Math+Phy+Chem+Comp/5;
```

Preprocessor Directives #include and #define

The preprocessor is used to handle directives for source file inclusion (#include) or defining macro definitions (#define).

Example:

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

#define

It is used to define a macro. The macro substitution is done during compile time.
Example:

```
#define MAX 80
#define Area(L,B) L*B

void main()
{
  int a,b,ar;
  cin>>a>>b;
  (a<b)?a=MAX:b=MAX;
  ar=Area(a,b);
  cout<<ar<<endl;
}
```

Actual Parameter

A parameter that is used in the function call to send the actual values to the function is known as actual parameter.

Formal Parameter

A parameter that is used in the function definition to receive the values from actual parameter is known as formal parameter.
Example:

```
void Square(int A)//A is formal parameter
{
  cout<<2*A<<endl;
}
void main()
{
  int N=4;
  Square(N);//N is actual parameter
}
```

Call by Value

In call by value, actual parameter and formal parameter have different memory locations, so the changes done in the formal parameter are not reflected back in the actual parameter.

Call By reference

In call by reference, actual parameter and formal parameter share the same memory location, so the changes done in the formal parameter are reflected back in the actual parameter. Requires **&** sign in formal parameter.
Example:

```
void Calc(float Sal,float &Itax)
{       //Sal - Call by value, Itax - Call by reference
  Sal=1.1*Sal;
  Itax=0.3*Sal;
}
```

Default Parameter

It is used to provide a default value to a parameter. If no value is sent from the actual parameter, the formal parameter automatically gets this default value. The default parameter cannot be referenced and cannot be placed before a non-default parameter.
Example:

```
void PrintLine(int N=20)
{
  for (int C=0;C<N;C++) cout<<"-";
}
void main()
{
  PrintLine(40);
  PrintLine();
}
```

Function Prototype

A function prototype in C++ is a declaration of a function that does not require the function body but does specify the function's name, parameter types and return type. While a function definition specifies what a function does, a function prototype can be thought of as specifying its interface. In the function prototype, argument names are optional, however, the type is necessary along with **&** or **[]** or default value (if required).
Example:

```
void Disp(char []);
void main()
{ Disp("Hello");}
void Disp(char Msg[])
{ cout<<Msg<<endl;}
```

Global Variable

A variable, which is declared outside all the functions in the program, is known as global variable. A global variable can be accessed and modified in any part of the program (i.e. in any function). If local variable carries identical name as global variable, to access the global variable scope resolution operator (::) is required.

Local Variable

A variable, which is declared inside a function or a compound statement in the program, is known as local variable. A local variable can be accessed and modified in the function or the compound statement in which it is declared.
Example:

```
int Num1=100,Num2=200;//Global Variables
void main()
{
  int Num2=20,Num3=30;//Local Variables
  Num1+=10;Num2+=20;::Num2+=30;Num3+=40;
  cout<<Num1<<Num2<<::Num2<<Num3<<endl;//1104023070
}
```

Type Casting

It is an explicit process of type conversion from a data type to another.
Example:

```
int A=1,B=2;
float C=(float)A/B;cout<<C;//Output:0.5
OR
int P=65;
cout<<(char)P<<endl; //Output:A
```

(Automatic) Type Conversion

It is an implicit process of type conversion from a data type to another.
Example:

```
int P=65; char CH;
CH=P;//Type Conversion
cout<<CH<<endl;//Output: A
```

Ternary Operator/Conditional Operator

It is an operator **(?)** with three operands.

Example:

```
int A=10,B=20,C;
C=(A>B)?A:B; //? As an expression
OR
int A=10,B=20;
(A>B)?cout<<A:cout<<B;//? As statement
```

Use of typedef :

typedef is a keyword in C++. It is used to provide an alternative name to existing data types.

Example:

```
typedef float Real;
typedef char STR[80];
typedef int MAT[2][3];
void main()
{
   STR S;//will mean same as char S[80];
   MAT M;// will mean same as int M[2][3];
   :
}
```

Extraction and Insertion operators

“>>” is known as extraction operator in C++ and is used with cin.
“<<” is known as insertion operator in C++ and is used with cout.
Example:

```
cin>>A>>B;
cout<<A<<"+"<<B<<"="<<A+B<<endl;
```

random() and randomize()

randomize() - A function of **stdlib.h**, used to initialise random number generator. randomize() function initialises the random number generator with a random value, which allows random() function to generate different set of random values in every execution.

random() - A function of **stdlib.h** that returns a random integer between **0** and **UpperLimit-1** (both inclusive).

Syntax: **`<IntegerVar>= random(<UpperLimit>);`**

Example 1

```
#include <iostream.h>>
#include <stdlib.h>
void main()
{
   randomize();
   int MAX=random(4)+1;
   for (int C=1;C<=MAX;C++)
       cout<<C<<”:”;
}
```

This example will generate values ranging from 0 to 3, which in turn added to 1 will result in value of MAX to be from 1 to 4. Hence, the possible outputs for the above program would be **any** of the following, **(i)** to **(iv)**:

(i)	`1:`
(ii)	`1:2:`
(iii)	`1:2:3:`
(iv)	`1:2:3:4:`

Example 2

```
int Number,MagicNumber;
Number=4;
for (int I=1;I<=5;I++)
{
   MagicNumber=random(Number);
   cout<<MagicNumber;
}
cout<<endl;
```

If the first time execution of this program generates output as
23122
Every execution of this program will generate **same** set of numbers in the output

Example 3

Now, let us see the same example with randomize() function

```
randomize();
int Number,MagicNumber;
Number=4;
for (int I=1;I<=5;I++)
{
   MagicNumber=random(Number);
   cout<<MagicNumber;
}
cout<<endl;
```

If the first time execution of this program generates output as
23122
Every execution of this program will generate **different** set of numbers in the output
12322…12320…01231…

Example 4	

`randomize();`
`char Guess[]={'A','E','I','O'};`
`int GN,N=4;`
`for (int I=1;I<=N;I++)`
`{GN=random(I);cout<<Guess[GN];}`
`cout<<endl;`
	Which of the following is/are not possible outputs from the C++ code
	(a) AIEE (b) AEAO
	(b) AAEI (d) AEIO

Expressions to generate numbers
 	
(a) between 10 to 20 (inclusive of 10 and 20) is going to be
`N=10+random(11);`
 	
(b) between 35 to 64 (inclusive of 35 and 64) is going to be
`N=35+random(30);`	

Pre/Post Increment/Decrement Operators

++ is an increment Operator to increment the value of a variable by one, when used before the operand known as pre-increment and when used after the operand known as post-increment operator.

-- is an decrement Operator to decrement the value of a variable by one, when used before the operand known as pre-decrement and when used after the operand known as post-decrement operator.

```
int A=100,B=150;
A++;
cout<<A<<endl;//101
++A;
cout<<A<<endl;//102
A+=++B;
cout<<A<<B<<endl;//253151
A+=B++;
cout<<A<<B<<endl;//404152
cout<<A+B<<A++<<++B<<endl;//558404153
cout<<++A<<B++;
cout<<A+B<<++A<<++B<<endl;//406153562407155
```

Commonly used ASCII Values for characters

CHARACTER RANGE	ASCII Value Range
'A' to 'Z'	65 to 90
'a' to 'z'	97 to 122

CHARACTER RANGE	ASCII Value Range
'0' to '9'	48 to 57

Structure – `struct`

Array of structure

```
sruct Member
{
  int Mn;     //Member Number
  char Nm[20];//Name
};
void Enter(Member M[],int N)
{
  for (int I=0;I<N;I++)
  {
    cin>>M[I].Mn;
    gets(M[I].Nm);
  }
}
void Show(Member M[],int N)
{
  for (int I=0;I<N;I++)
    cout<<setw(5)<<M[I].Mn
    <<setw(20)<<M[I].Nm
    <<endl;
}

//To search on the basis of Eno
void SearchEn(Member M[],int N)
{
  int Sen,Found=0;
  cout<<"Eno to search:";cin>.Sen;
  for (int I=0;I<N-1;I++)
    if (M[I].En==Sen)
    {
      cout<<"Name:"<<M[I].Nm<<endl;
      Found++;
    }
  if (!Found)
    cout<<"Sorry Not Found…"<<endl;
}

//To search on the basis of Name
void SearchNm(Member M[], int N)
{
  char SNm[20];int Found=0;
  cout<<"Nm to search:";gets(SNm);
  for (int I=0;I<N-1;I++)
    if (strcmp(M[I].Nm,SNm)==0)
    {
      cout<<"Mno:"<<M[I].Mn<<endl;
      Found++;
    }
  if (!Found)
    cout<<"Sorry Not Found…"<<endl;
}
```

```
//To sort on the basis of Eno
void SortEn(Member M[],
                      int N)
{
  for (int I=0;I<N-1;I++)
  for (int J=0;J<N-I-1;J++)
    if (M[J].Mn>M[J+1].Mn)
    {
      Member T=M[J];
      M[J]=M[J+1];
      M[J+1]=T;
    }
}
//To sort on the basis of Name
void SortNm(Member M[],
                      int N)
{
  for (int I=0;I<N-1;I++)
  for (int J=0;J<N-I-1;J++)
    if
     (strcmp(M[J].Nm,M[J+1].Nm)>0)
    {
      Member T=M[J];
      M[J]=M[J+1];
      M[J+1]=T;
    }
}

void main()
{
  Member Mem[5];
  Enter(Mem,5);
  Show(Mem,5);
  SearchEn(Mem,5);
  SearchNm(Mem,5);
  SortEn(Mem,5);
  Show(Mem,5);
  SortNm(Mem,5);
  Show(Mem,5);
}
```