⇒ **Tokens:-** A c++ Program is composed of tokens which are the smallest individual unit.

Tokens can be:
There are 6 types of tokens in c++.

1) keyword
2) Identifiers
3) Constants
4) Strings
5) Special symbols
6) Operators

⇒ **Keywords:-** In c++ keywords are reserved words that have a specific meaning in the language & can't be used as identifiers.

NOTE:- keywords are essential part of c++ lang & are used to perform specific tasks or operations.

⇒ 95 keywords in c++
⇒ Const continue default delete do else double enum etc.

Identifiers:- An identifier is a name given to a variable, fun or another object in the code.

They can consist of letters, digits & underscore.
Some rules must be followed when choosing an identifier.

① The first character must be a letter or an underscore.
② Identifiers cannot be the same as a keyword.
③ Identifiers cannot contain any spaces or special characters except for the underscore.
④ Identifiers are case sensitive meaning that a variable myVariable is different from myvariable.

(A) Valid Identifiers:-

my_variable
Student_name
balance_due.

(B) Invalid Identifiers:-

my variable (Contains a space)
Student# (Contains a special character)
int (Same as keyword.

NOTE It is important to chose meaningful & descriptive name for your identifiers to make your code easier to read & understand.

(3) Constants:- A Constant is a value that Cannot be changed during the execution of the Program.

⇒ Constants often represent fixed values. used frequently in code.
⇒ Constants can be of any of the basic data types & can be divided into integer, floating point Numerals, characters, string & boolean values
⇒ Const keyword is used.

Ex: Const double PI = 3.14159;
↓ ↓
datatype initialize

(4) Strings:- A string is a sequence of characters that represent text

⇒ Strings are Commonly used in C++ Pros. to store & manipulate text data.
⇒ To use string in C++ → <string> → Headerfile at the beginning of your Pros.
That will provide access to the String. class, which is used to create & manipulate string in C++

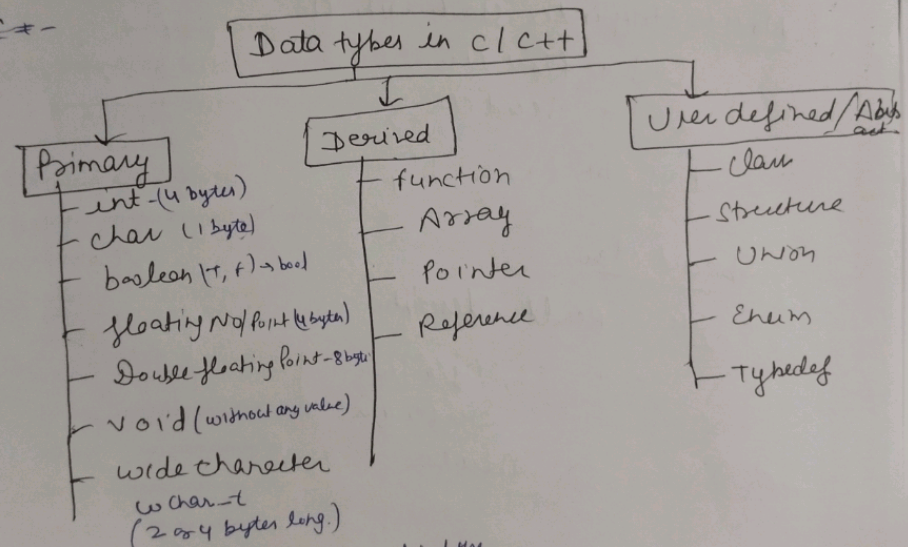
Data types in C++ :-

⇒ All variable use data type during declaration to restrict the type of data to be stored. Therefore we can say that data types are used to tell the variables the types of data they can store.

⇒ Whenever a variable is defined in C++, the compiler allocates some memory for that variable based on data type with which it is declared.

⇒ C++ support a wide variety of data types. Data types specify the size & types of values to be stored.

C++ -



⇒ Size of operator :- Used to find the No. of bytes occupied by a variable data type in computer.

⇒ Data type Modifiers :- There are used with built in data types to modify the length of data that a particular data type can hold

#

⑤ Special Symbol :- several special symbols are used for 19 various purpose.

⇒ & (ampersand) → Address

⇒ tilde (~) → bitwise Not

⇒ Asterisk (*) → multiplication & pointer operator

⇒ Pound (#) → Pound sign → Preprocessor directives (Special type of instn Processed by the Preprocessor before the code is compiled

⇒ Percentage (%) → Modulus operator (Used to find Remainder)

⇒ Vertical bar (|) → bitwise OR operator

⇒ Caret (^) → bitwise XOR

⇒ Exclamation Point (!) → Logical Not

⑥ Operators :- Operators are special symbols that are used to perform operations on one or more Operands. int c = a+b;

⇒ An operand is a value on which an operator acts.

Including – Arithmetic operator, Relational, Logical & others

⇒ Arithmetic operators :- +, −, *, /

⇒ Relational operators :- == (equality), ! (inequality) > (greater) ; < (less)

⇒ Logical operators :- Used

⇒ Used to combine 2 or more relational expression & determine their logical relationship.

⇒ Some examples of logical operators – && (Logical And) || (Logical OR) ! (Used for Logical Not

Bitwise operator :- Bitwise operators are used to perform bitwise operations on individual bits of a value & the assignment operators are used to assign a value to a variable

Operators in C++

	Operator	Type
Unary operator →	++ , --	Unary operator
	+ , - , * , / , %	Arithmetic operator
Binary operator	< , <= , > , >= , == , !=	Relational operator
	&& , || , !	Logical operator
	& , | , << , >> , ~ , ^	Bitwise operator
	= , +∓ , -= , *= , /= , %=	Assignment "
Ternary operator →	? :	Ternary / Conditional

A) Unary operator :- There operator operate or work on a single operand.

Ex:- Increment operator

```
int a=5;
a++;
return 6
```

Decrement operator

```
int a=5;
a--;
return 4
```

⇒ ++a → The value of variable is incremented first & then it is used in Prog.

⇒ a++ → The value of the variable is assigned first & then it is incremented.

NOTE :- C++ uses the + operator for both addition & concatenation.
No- are added & Strings are concatenated.

B) Binary operators:- There operators operate or work with 2 operands. for eg. Addition, Subtraction

```
int a=3, b=6;
int c= a+b; // c=9
```

NOTE Modulo operator (%) operator should only be used with integers.

C) Relational operator:-

```
int a=3,
    b=6;
```

== → checks if both operands are equal.

a==b; → Return false.

d) Logical operator:- There operators are used to combine 2 or more conditions or constraints or to complement the evaluation of the original condition in consideration. The result returns a boolean value i.e t&f

i) Logical AND (&&) →

⇒ Return the true only if all the operands are true or Non zero.

Ex:- int a= 3,
b = 6;
a && b; → Return true

(b) Logical OR → || Retu

Returns true if either of the operands is true or Non zero.

int a=3, b=6;
a||b; — return true

(c) Logical NOT → ! Return true if the operand is false or zero

! a; → Return false

4) Bitwise operators:- These operators are used to perform bit level operations on the operands.

NOTE - only char & int data types can be used with bitwise operators.

5) Assignment operators:- These operators are used to assign value to the variable.

⇒ The value on the right side must be of the same data type as the variable on the left side. otherwise the Compiler will raise an error.

Variables in C++ :-

⇒ variables in c++ is a name given to a memory location. It is the basic unit of storage in a Program.

⇒ The value stored in a variable can be changed during Program execution.

⇒ A variable is only a name given to a memory location all the operations done on the variable effects that memory location.

⇒ In C++, all the variables must be declared before use.

How to declare a variable :-

(1) Single variable :- type variable_name;

(2) Multible variable :- type variable1_name, variable2_name,

⇒ A variable name can consist of alphabets, numbers & underscore character.

⇒ However the name must not start with a no.

int age = 15; ← value
data type ↑ variable

data type :- Type of data that can be stored in this variable.

⇒ Variable_name :- Name given to the variable.

Value :- It is the initial value stored in the variable.

⇒ Ex :-

int a=50, b=100 → integer type.

float = 50.8 → declaring a variable of float type.

char c='z'; → declaring a variable of char type.

Rules for declaring variable :-

1) The name of the variable contains letters, digits & underscore.
2) The name of the variable is case sensitive.
3) The name of the variable does not contain any white space and special characters. (ex → #, $, %, * etc.)
4) All the variables names must begin with a letter of the alphabet or an underscore (_).
5) We cannot use C++ keyword (ex. float, double, class) as a variable name.

int a; → declaration of variable.

a=10; → initialisation

definition = declaration + initialisation.

⇒ Types of variables :-

① Local variable
② Instance variables
③ Static variables

```
class CF {
 Public:
 static int a; → static variable
    int b; → instance variable
 Public:
 func()
 { int c; → local variable
 };
};
```

① Local variable :- A variable defined within a block or method or constructor is called a local variable.

The Scope of these variables exists only within the block in which the variable is declared.

Initialisation of local variable is Mandatory.

② Instance variables :- Instance variables are non static variables and are declared in a class outside any method, constructor or block.

Initialisation of instance variable is not mandatory.
Instance variable can be accessed only by creating objects.

③ Static variables :- Static variables are also known as Class variables.

Static variables are declared using the static keyword within a class outside any method constructor or block.

Type conversion :-

⇒ Type conversion is the process that convert the predefined data type of one variable into an appropriate data type.

⇒ The main aim behind type conversion is to convert two different data type variables into a single data type to solve mathematical & logical expression easily without any data loss.

⇒ Type conversion can be done into two ways.

1.) Implicit type conversion
2) Explicit type conversion

① Implicit type Conversion :- This is the type of conversion done automatically by the Compiler without any human effort.

It means an implicit conversion automatically convert one data type into another type based on some predefined rules of the c++ Compiler. Hence it is also known as the automatic type conversion.

⇒ For example :-

```
int x = 20;
Short int y = 5;
int z = x + y;
```

Here x, y → short int integer type.

Int type

NOTE It avoids the data loss, overflow or sign loss in implicit type conversion of c++.

② Explicit type Conversion :-

Conversion that require user intervention to change the datatype of one variable to another, is called the explicit type conversion.

In other words, an explicit conversion allows the programmer to manually changes or typecasts the datatype from one variable to another type. Hence it is also known as typecasting.

Operator Precedence :-

It specify the order of operations in expressions that contain more than one operator.

operator precedence determines the grouping of terms in an expression. The associativity of an operator is a property

that determines how operators of the same precedence are grouped in the absence of Parenthesis. 25

for example the multiplication operator has higher precedence then the addition operator.

for example :- $x = 7 + 3 * 2$

$x = 13$ not 20.

because operator * has higher precedence then +.
So it first determine / multiplied with 3*2 then add with 7.

Postfix	() [] -> . ++ --	Left to Right
Unary	+ - ! ~ ++ -- (type)* & sizeof	Right to Left
Multiplicative	* / %	Left to Right
Additive	+ -	Left to Right
Bshift	<< >>	Left to Right
Relational	< <= > >=	Left to Right
Equality	== !=	Left to Right
Bitwise AND	&	Left to Right
Bitwise XOR	^	Left to Right
Bitwise OR	|	Left to Right
Logical AND	&&	Left to Right
Logical OR	||	Left to Right
Conditional	?:	Right to Left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to Left
Comma	,	Left to Right

Implicit type conversion :- This is the type ... automatically by the

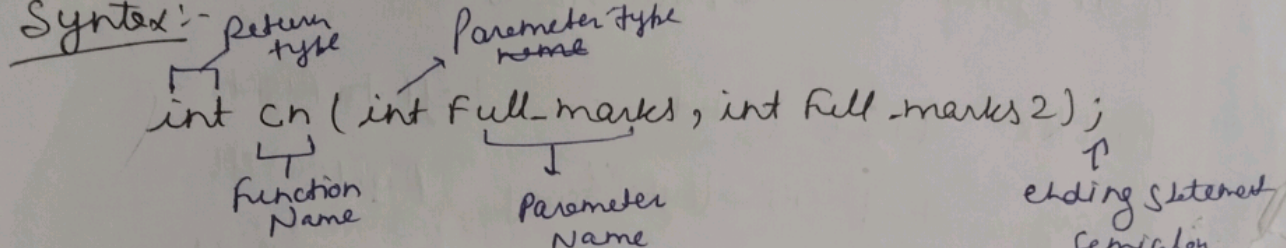
Main function :-

⇒ A function is a set of statement that takes input, does some specific computation & produce output.

⇒ The main aim to use / make a function so that instead of writting the same code again & again for different inputs, we can call this function.

⇒ In simple term a function is a block of code that runs only when it is called.

Syntax :-



Why do we need functions :-

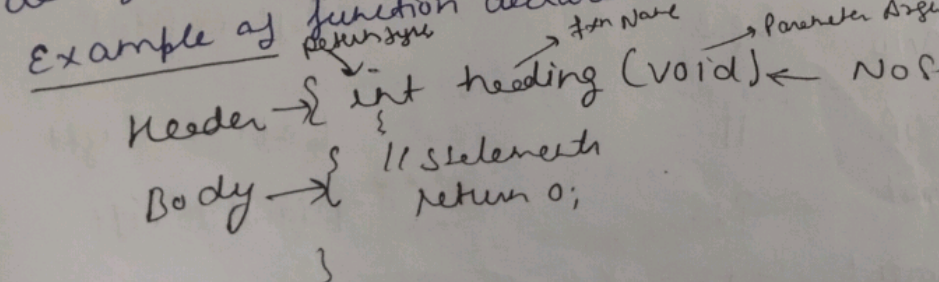
⇒ Functions helps us in reducing code redundancy.

⇒ Functions make code modular.
Consider a big file having many lines of code. It becomes really simple to read & use the code, if the code is divided into functions.

Function Declaration :-

⇒ A function declaration tells the compiler about the no of Parameter & data types of Parameter & Return type of function.

Example of function declarations.



Structure of a function:-

```
function_return_type function_name
(data_types arg1,
 data_types 2 arg2 ---- )
{
  body of function
  return (value);
}
```

Fxn name should have defined with a name.

⇒ Function_Return_type:- In place of this we write the type of value which returns by the function to the calling portion of the program for example int, float, char etc.

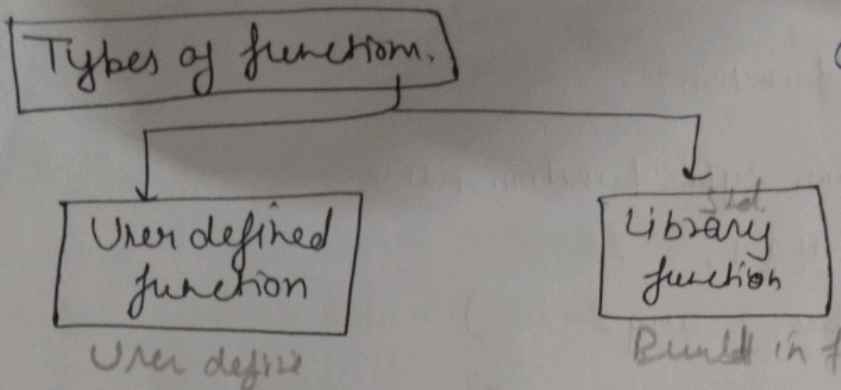
NOTE:- If a fun. return type is void then there is no need of return statement within the fun body.

⇒ Function_name:- Function name can be any name conformity to the syntax rules of identifier.

Ex:- display ();
square_of_a ();

⇒ Datatype and Arg:- Arg means Argument.
An argument is a data passed from a program to a fxn.
The data-type of argument tells about the type of argument
Note that the each argument variable must be declared independently inside the parenthesis.

a=20



① return_type function_name (Parameter 1, Parameter 2) {
statement;
}

Q Write a fxn which calculates sum of 2 integer type no.

```
int sum (int a, int b)
{
 int sum;
 sum = a+b;
 return (sum);
}
```

```
int main () {
 return 0;
}
```
↓ Return type integer

```
void print (int num)
{
 cout << num;
 return;
}
```
→ fun ended

(iv) Body of function :- The body of fxn is same like the body of main() program. Statement or a block of statements enclosed b/w two braces; {and}

(v) Return statement :- The return is a reserved word. It is used to terminate function & return a value to its caller.

⇒ The syntax of return statement is —

return (value); or return value;

The value can be expression also. We can skip the value part i.e return;

Ex!- return (a-b);
return (j--);
return;

In above prog within function body we can write the following statement in place of

```
int temb = a * b;
return (temb);
⇓
return (a*b);
```

both are equivalent.

Structure of function :-

```
return-type function-name (Parameter-type variable.name)
{
    // Logic
    return value;
}
```

Argument :-

Actual Argument

```
c = add (10, 15);
```

Actual argument

Formal Arguments-

```
int add (int a, int b);
```

Formal Argument

⇒ Passing argument to function :-

⇒ Passing by value —

```
int add (int a, int b)
{
return a+b;
}
```

a = 20
it will not change original value of x.

NOTE:- A fxn can also have more than one return statement

Example:

```
int smaller (int a, int b)
{
if (a<b) return (a);
  else return (b);
  }
```

Ex Write a fxn which calculates the multiplication of 2 no, call this fxn in main Prog.

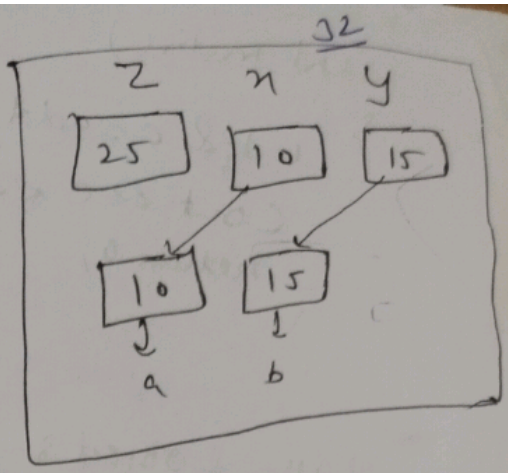
```
#include <iostream.h>
#include <conio.h>
Void main ()
{ int x, y;
int mul (int a , int b); // function Prototype
clrscr();
cout << " Enter the value of first no -: ";
cin >> x;
Cout << " Enter the value of second no. : ";
cin >> y;
Cout << " The multiplication is : " << endl;
int m = mul (x,y); // Calling fxn.
Cout << m << endl;
getch ();
}    // end of main Program
Int mul (int a, int b)
{ int temp = a * b;
return (temp);
}
```

O/P

Enter the value of first No: -5
Second : -6
The multiplication is
30

```
int main()
{
int z, x=10, y=15;
z = add(x, y);
cout << z << endl;
return 0;
}
```

→ Passing by value.



⇒ Pass by Reference :-

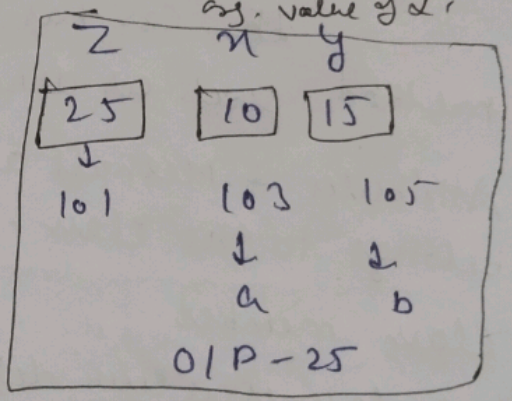
```
int add (int &a, int &b)
{
    return a+b;
}
// return by value

int main()
{
int z, x=10, y=15
z = add(x, y);
cout << z << endl;
return 0;
}
```

Adv :- No extra m/m allocation takes ~~takes~~ in Pass by value.

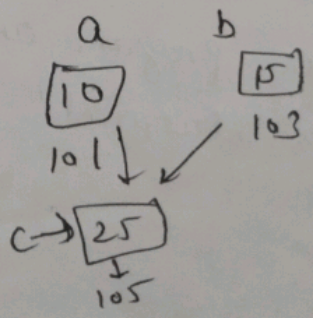
Disadv

If a=20 defined in fun def then it will change the original value If we will do a=20 then it will change org. value y &.



⇒ Return by Reference :-

```
int &add (int a, int b)
{
    int c;
    c = a+b;
    return c;
}
```



Note:- A fun can also have more than one return statement

```
int main()
{ int &c = add(10,15);
  cout << c << endl;      //25
  return 0;               //&c
}
```

class & object :-

class is a template for objects and an object is an instance of a class.

When the individual objects are created they inherit all the variables & functions from the class.

Everything in c++ is associated with classes & objects along with its attributes & methods.

For example :-

Car → is an object

The car has attributes such as weight & color & method, such as drive & brake.

Attributes & Methods are basically variables & functions that belongs to the class. There are often referred to as class member.

A class is a user defined data type that we can use in our program, and it works as an object constructor, or a "blueprint" for creating objects.

Access specifiers (Modifiers) in C++, Use & Accessibility :-

1. Private
2. Protected
3. Public

Ex: class

```
{ private :  ----> 1. Member fun
                   2. friend
  int a;  ----> (class's data variable is called Data member.
```

In Private :- class itself access its class member fun.
② Friend of class.

```
Protected :   Use in case of
int a;        // Inheritance
```

In Protected :- Datamember written in base class & also derived class which inherit it can access its member fun.

```
Public :
int a;        // Class inside m/m fun & outside of the class can access it.
```

Difference among Private, Protected and Public access specifier ?

Private	Protected	Public
① If the mode of access specifier is private then it can only accessible for the class & friend.	① If the mode of access specifier is protected then it can be accessible for the class itself and in case of inheritance also accessible for its derived class.	① If the mode of access specifier is Public then we can access the public member with in the class & outside of the class. (Anywhere or Anywhere)

② It Cannot be inherited

③ It Provide High Security for its own data member.

② It is inherited for derived class with in range.

3. It Provides less Security than Private data member

The Public member is inherited for derived class.

3. It does not Provide any Security for its data members.

Access Specifier	Class	Derive class	Friend functions.
Private	Accessible	Not Accessible	Accessible
Protected	"	Accessible	"
Public	"	"	"

Difference between class & object?

Class

① class is a collection of data member and member function (object).

② class always declared with keyword class.

③ we can't pass value directly in class.

④ class is declared at once

Object

① object is Instance of a class.

② At the time of object creation we doesn't require any keyword.

3 we can pass values directly through the objects

4. One or more object created for a Particular class

```
myfunction ("Jenny");
myfunction ("Anja");
return 0;
}
```

⇒ When a Parameter is Paned to the function, it is called an argument.

fname → Parameter

While Liam, Jenny & Anja are arguments

Default Parameter value:

You can also use a default Parameter value by using the equal sign(=).

If we call the fun without an argument, it uses the default value ("Norway");

A parameter with a default value is often known as an "optional Parameter".

```
# include <iostream>
# include <string>          Using namespace std;
void myfunction(String Country = "Norway")
    {
Cout << Country <<"\n";
  }
int main() {
myfunction ("sweden");          // Sweden
myfunction("India");            // India
myfunction();                   // Norway
myfunction ("USA");             // USA
return 0;
  }
```

Example:- Create a class called "MyClass";

```
class MyClass // class
{
  Public:      // Access specifier → Specify that membe
                  (attr Method)
int myNum;     // Attribute (int variable)
string myString; //(String variable)
  };
```

Attribute :-

When variables are declared within a class they are called attributes.

end with semicolon (;)

Parameter & Arguments :-

Info. can be Passed to function as a Parameter.

Parameters act as variable inside the function.

Parameter are specified after the fun name, inside the ~~Parantheses~~.

You can add as many Parameters as you want, just separate by a comma.

```
void functionName (Parameter 1, Parameter 2, Parameter 3)
    {  ____  //Code to be executed
    }

void myFunction (string fname)
{
Cout << fname << "Refines \n";
}
int main() {
myFunction("Liam");
```