C++ Function

→ The function in C language is the block of code
→ To perform any task we can create function.
→ A function can be called many times.

#* Advantages of function in C.
i. Code reusability
ii. Code optimization.

#* Types of functions.
→ There are two types of function in C programming.
i. Library function → The funct which are already defined in C++ language.
ii. User-defined funtions → The function which are ~~decided~~ ~~by~~ created by the user, to use many times.
→ It reduces complexity of a big problem of optimizes the code.

#* Declaration of function.

declaration
```
return type  funtion name(data type parameter)
{
    //code to be created
};
```

```
int main
{
    function name( ); // funct^n call
}
```
initialization

There are two ways to pass value or data to a function

$$\begin{tikzcd} & \text{There are two ways to pass value or data to a function} \arrow[dl] \arrow[dr] & \\ \text{i } \boxed{\text{Call by Value}} & & \text{ii } \boxed{\text{Call by Reference}} \end{tikzcd}$$

Key point: Original value is not modified in call by value, but it is modified in call by reference

Call by Value

Original value is not modified.
Value being pass to the function is locally stored by the function parameter in $\boxed{\text{stack memory}}$ location.

* STACK is based on LIFO

If you change the value of function parameter, it is changed for the current function only.
It's value is not change in the main() functⁿ.

Ex:
```
#include<iostream>
void func(int data);
int main()
{

    int data=3
    func(data);
    cout<<data;
    return 0
}
```

```
void func(int data)
{
    data=5;
}
```

Output = 3 [illegible]

#Call by Reference.

*→ In call by value, origional value will modified because we pass reference (address).

→ Here, address of the value is passed in function. So actual cf formal arguments share the same address. Hence, value changed inside the function, is reflect inside as well as outside the function.

ex:
```
#include<iostream>
using namespace std;
  void swap(int *x, int *y)
  {
      int swap;
      swap = *x.
      *x = *y
      *y = swap.
  }
int main()
 {
   int x=500, y=100;
   swap(&x, &y);
   cout<<"Value of x is:"<<x<<endl;
              "      y
   return 0;
```
output:
```
 x = 100
 y = 500
```

Date			
Page No.			

Difference call by value & call by reference.

Call by Value	Call by reference.
i. Value passed to the funct?	i. Address is passed to the function
ii. change made inside the functⁿ is not reflected on other function	ii. change made inside the function is reflected outside the functⁿ also.
* iii Actual & formal arguments will created in <u>different</u> memory allocation	iii. Actual & formal arguments will be created in same memory location.

C++ Recursion

When the function is called within the same function, it is known as recursion. functⁿ which calls the same function is known as recursive function.

<u>Tail Recursion</u> → A function that calls itself, & doesn't perform any task after function calls, is known as tail recursion.

<u>ex</u>:
```
recursionfunction() {
        recursionfunction(); '
    }.
```
$\hookrightarrow$ calling self function

C++ Storage Classes.

→ storage class is used to define the lifetime of visibility of a variables of /or function within program.

→ lifetime refers to the period during which the variable remains active of visibility refers to the module of a program in which the variable is accessible.

# There are five types of storage classes.

1. Automatic
2. Register
3. External
4. Mutable
5. Static

Automatic storage class.

The auto keyword provides type inference capabilities, using which automatic deduct^n of the data type of an expression in a programming language can be done.

This consume less time having to write out things the compiler already knows.

As all the types are deduced in compiler phase only.

The time for compilation increase slightly but it does not affect the run time of program.

$\boxed{\text{It is default storage for all local Variable.}}$

Register Storage Class

The register variable allocates memory in ~~mem~~ register than RAM.
Its size is same of register size.
It has faster access than other variable.

Note: We can't get the address of register variable

ex: register int a = 0;
↳ keyword.

its store in ~~or~~ register only if only if space is free, if not then in memory

always faster as compared to memory.

Static

→ Static variables have a property to preserving thier value even after they are out of their scope.

→ Static variables preserve the value of their last use in their scope.

→ We can say that they are initialized only once & exist until the terminat of the program.

→ Thus, no ~~new~~ new memory is allocated because they are not re-declared.

→ Their Scope is local to the function which they defined.

Note: By default, they are assigned the value 0 by the compiler.

Extern

→ Extern storage class simply tell us that the variable is defined elsewhere & not within the same block.

→ Basically, the value is ~~defined~~ assigned to it in a different block & this can be overwritten/changed in a different block as well.

ex: extern int Counter=0;

Date
Page No.

Mutable

→ Sometimes there is a requirement to modify one or more data members of class/struct through const function even though you don't want to the funct^n to update other members of class/struct.

When we declare a function as const, the pointer passed to funct^n becomes const. Adding mutable to a variable allows a const pointer to change member.

ex:
```
#include <iostream>
using namespace std;

class Test {
Public:
    int x;
  mutable int y
  → keyword

  Test()
  { x = 4;
    y = 10;
  }
};
 int main()
{
 const Test t1;
 t1.y = 20
 cout << t1.y;
 return 0;
```