Section 11 : Functions

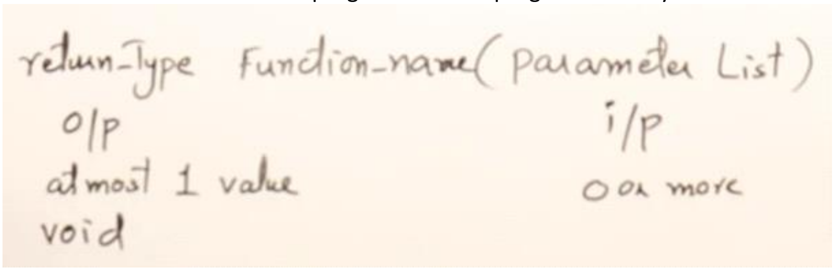
15 April 2022 16:49

© Rajat Kumar
https://www.linkedin.com/in/imRajat/
https://github.com/im-Rajat

Section 11 : Functions

Functions :

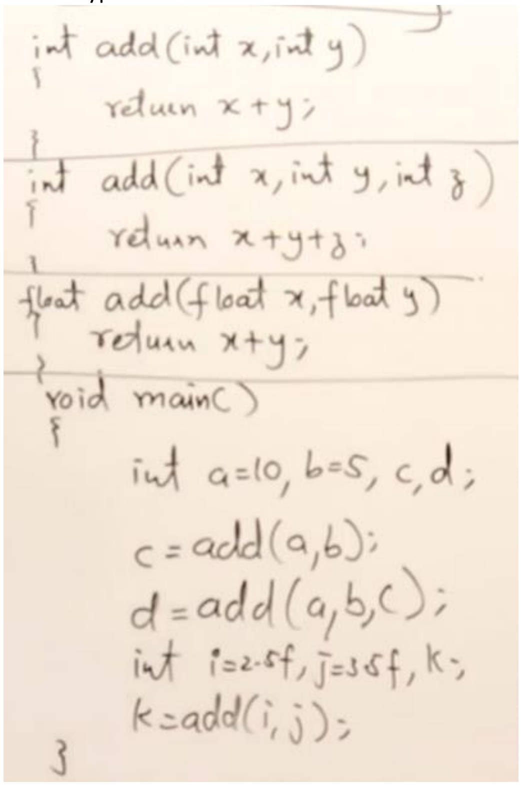
- It is peace of program which performs a specific task.
- It may take some inputs as parameter/arguments and return a result as return value.
- Functions are useful for procedural programming or modular programming.
- It can reuse in the same program or other program as many times.



It is got practice to don't have user interaction (cout and cin) inside the function. Do this only in main function if possible.

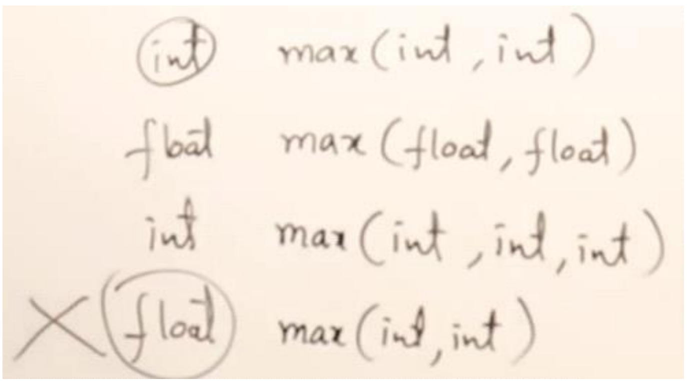
Function Overloading :

- Functions with same name but different argument list
- Return type is never consider in function overloading. Functions that differ only in their return type cannot be overloaded.



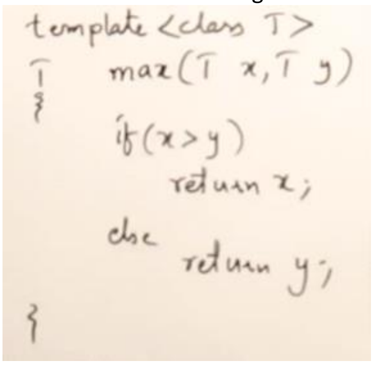
```
int add(int x,int y)
{
    return x+y;
}
int add(int x,int y,int z)
{
    return x+y+z;
}
float add(float x,float y)
{
    return x+y;
}
void main()
{
        int a=10, b=5, c,d;
        c=add(a,b);
        d=add(a,b,c);
        int i=2.5f, j=3.5f, k;
        k=add(i,j);
}
```

- If function name and parameters are exactly same but return type is different means they are not overloaded. It's a name conflict. We are redefining same function again.



```
int   max(int, int)
float max(float, float)
int   max(int, int, int)
X float max(int, int)
```

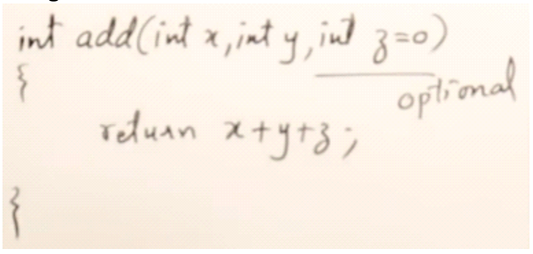
Function Template :
- The functions which are generic. Generalized.



```
template <class T>
T   max(T x, T y)
{
    if(x>y)
        return x;
    else
        return y;
}
```

- Instead of having different functions for different data type, we can use template T.

Default Arguments :

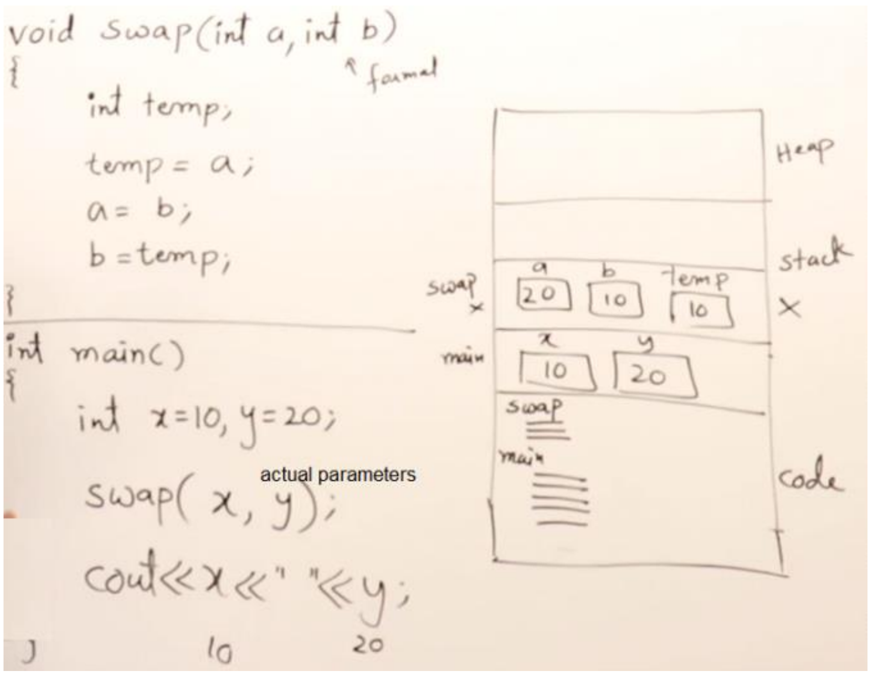


```
int add(int x, int y, int z=0)
{                     optional
    return x+y+z;
}
```

- Making default argument should start from right side to left only without skipping any.

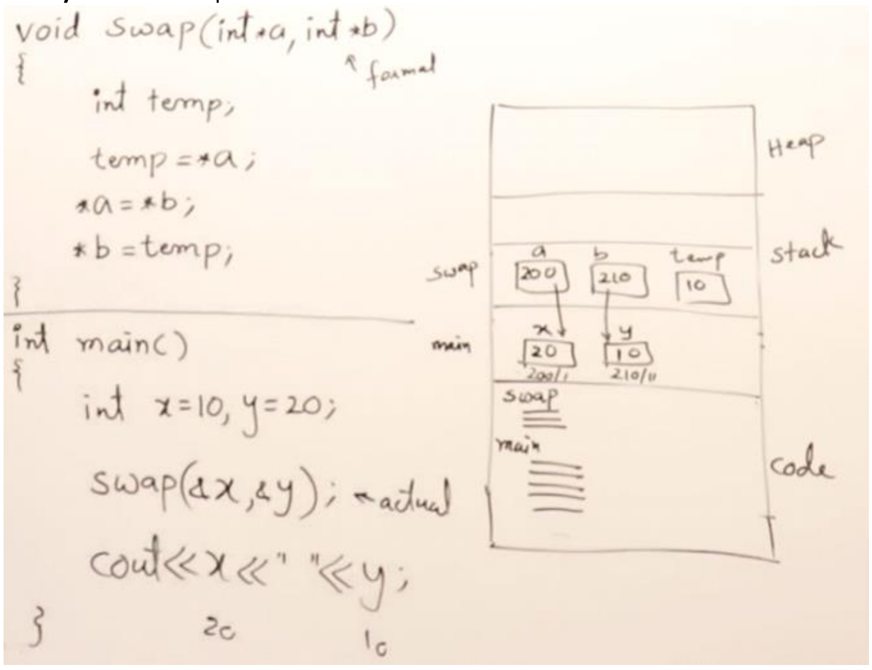
Parameter Passing Methods :(Pass by can be called as call by)
- Pass by Value
- Pass by Address
- Pass by Reference (not available in c)

Call by Value: Value of variables are copied and different space is allocated to them



```
void swap(int a, int b)
{
      int temp;
      temp = a;
      a = b;
      b = temp;
}
int main()
{
     int x=10, y=20;
     swap( x, y);
     cout<<x<<" "<<y;
}
```

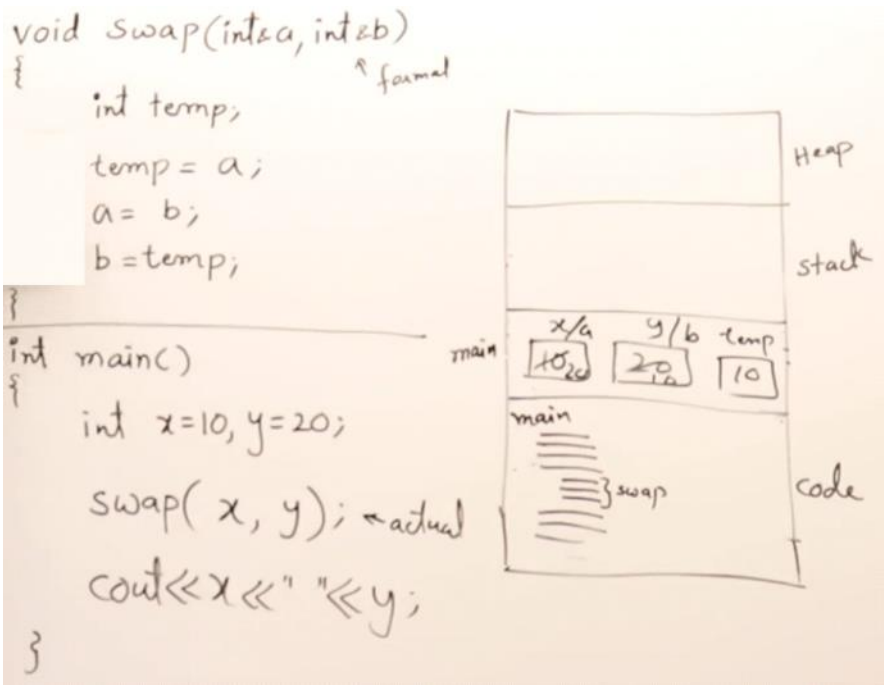
Call by Address: We pass the addresses of the variables.



```
void swap(int *a, int *b)
{
      int temp;
      temp =*a;
     *a = *b;
     *b = temp;
}
int main()
{
     int x=10, y=20;
     swap(&x,&y);
     cout<<x<<" "<<y;
}
```

Call by Reference:

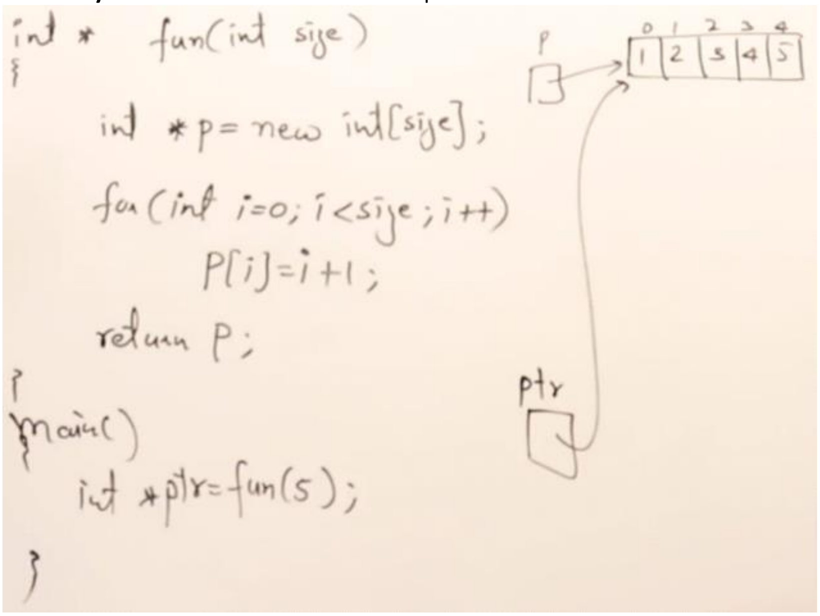
- References are just a nickname of a variable. Syntax is very similar to call by value. It just we can & in variable in functi on definition. But it worked similar to call by address as it can modify the value of actual parameters.
- Whenever we use call by reference mechanism it will not generate separate piece of machine code. It will copy the machine cod e at the place of function call. The function will not be a separate function, it will part of main function only.
- When use call by reference avoid using loops (may get warnings).
- If the piece of machine code of a function is copied at the place of function call like below then such functions are called as inline functions in C++.
- When we use a call by reference function automatically becomes in-line function.



```
void swap(int&a, int&b)
{
     int temp;
     temp = a;
     a = b;
     b = temp;
}
int main()
{
     int x=10, y=20;
     swap( x, y); ← actual
     cout<<x<<" "<<y;
}
```

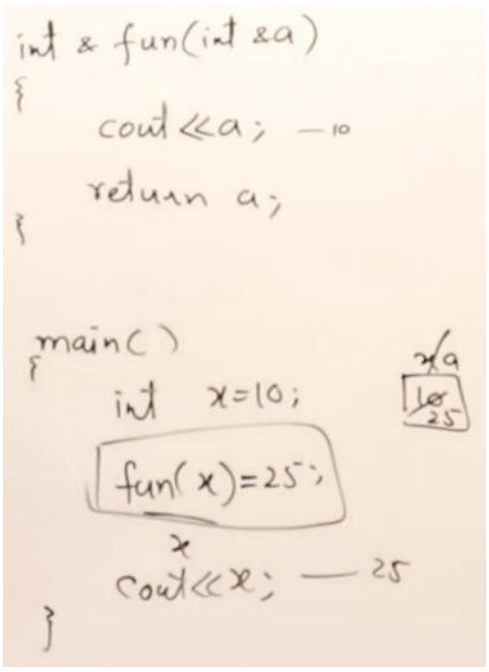
- If I write any complex code like loops or something this function will no longer be using call by reference. It may become ca ll by address automatically compilers will change it. So therefore we should not write any complex code inside the functions which are using call by reference.

Return by Address : We can also return a pointer



```
int *   fun(int size)
{
     int *p= new int[size];
     for(int i=0; i<size; i++)
          P[i]=i+1;
     return P;
}
main()
{
   int *ptr=fun(5);
}
```

Return by Reference :



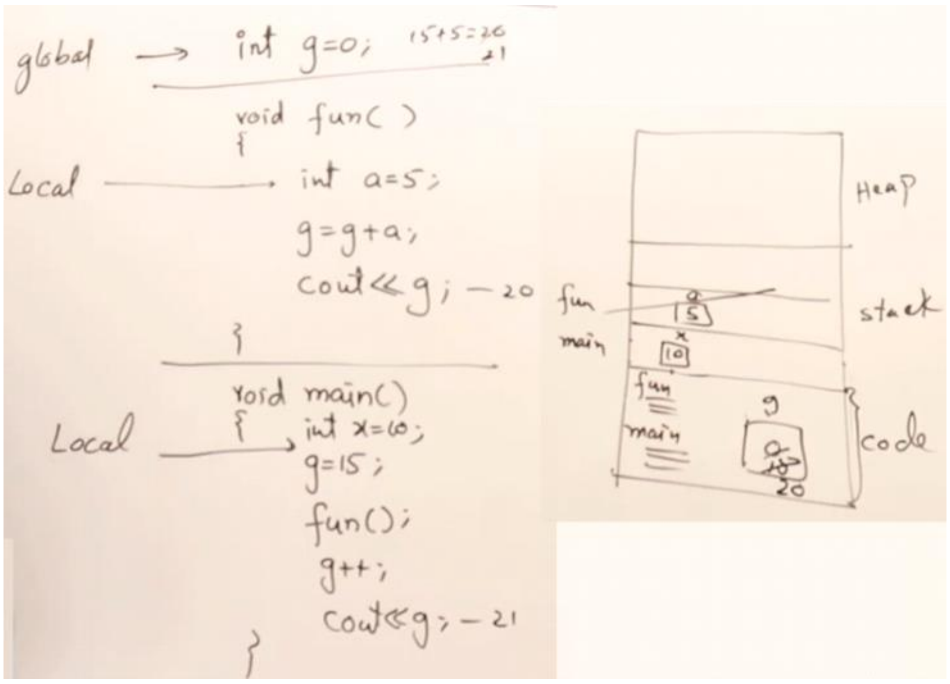
```
int & fun(int &a)
{
    cout<<a; —10
    return a;
}

main()
{
    int  x=10;
    fun(x)=25;
    cout<<x; — 25
}
```

- Mostly we make the functions as R values, but we can make it as L value using references.

Global vs Local Variables :

- Variable inside the function is local and the variable outside all the functions is global.
- Global variable are accessible in all the functions
- Small piece of memory as belonging to the section only, which is meant for keeping global variables. So we should look at thi s and the initial size zero.



```
global  →  int g=0;   15+5=20
                          21
           void fun( )
           {
Local  ——→   int a=5;
             g=g+a;
             cout<<g; — 20
           }

           void main()
Local  ——→ {  int x=10;
              g=15;
              fun();
              g++;
              cout<<g; — 21
           }
```

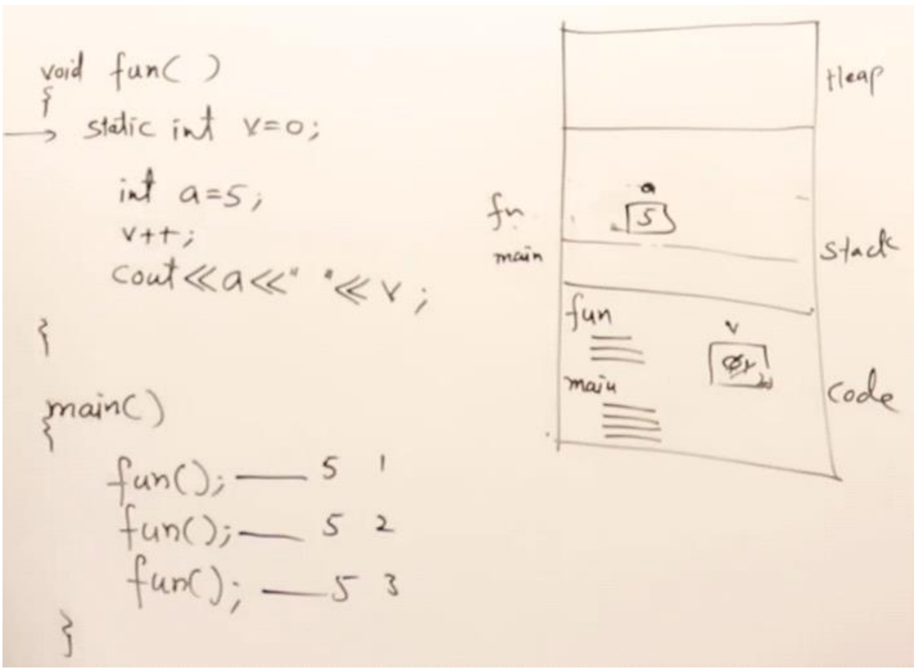
Scoping Rule :C++ have block level scope.

```
int x=10;
int main()
{
    int x=20;

    {
        int x=30;
        cout<<x<<endl; // Output : 30
    }
    cout<<x<<endl; // Output : 20
    cout<<::x<<endl; // Output : 10
}
```

Static Variables :

- There are two points about **Global Variable**. It can be accessible everywhere, and it will remain always in the memory.
- Static Variable will remain always in the memory but not accessible everywhere.
- Static variables are the variables which remains always in the memory. Always in the memory. They are just like a global Vari able. Only the difference between global and static variable is global variable can be access in any function, but static variable are accessible only inside the function in which they are declared.
- Think of a static variable imagine that they are global. But their scope visibility is limited to a function.
- Static variables are not available in Java.



```
void fun( )
{
    static int v=0;
     int a=5;
     v++;
     cout<<a<<" "<<v;
}

main()
{
    fun(); —— 5  1
    fun(); —— 5  2
    fun(); —— 5  3
}
```

Recursive Function :

- A function calling itself is called as recursion.

```
void fun(int n)
{
    if(n>0)
    {
        cout<<n<<endl;
        fun(n-1);
    }
}
```

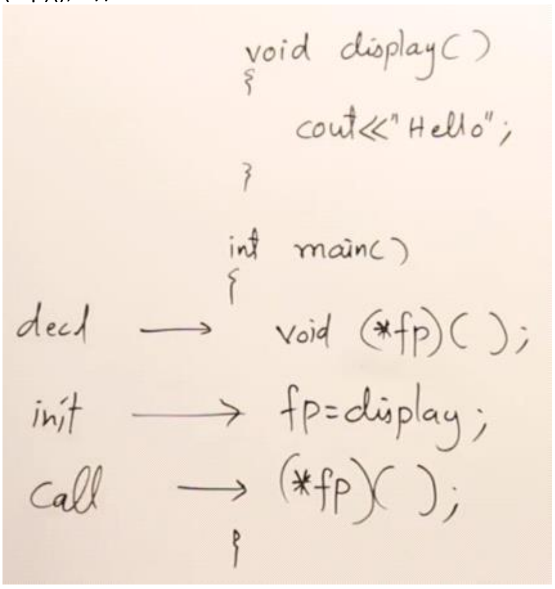
```
void fun(int n)
{
    if(n>0)
    {
        fun(n-1);
        cout<<n<<endl;
    }
}
```

- Output as : 5 4 3 2 1, and 1 2 3 4 5 when 5 is passed as n.

Function Pointer / Pointer to a Function :

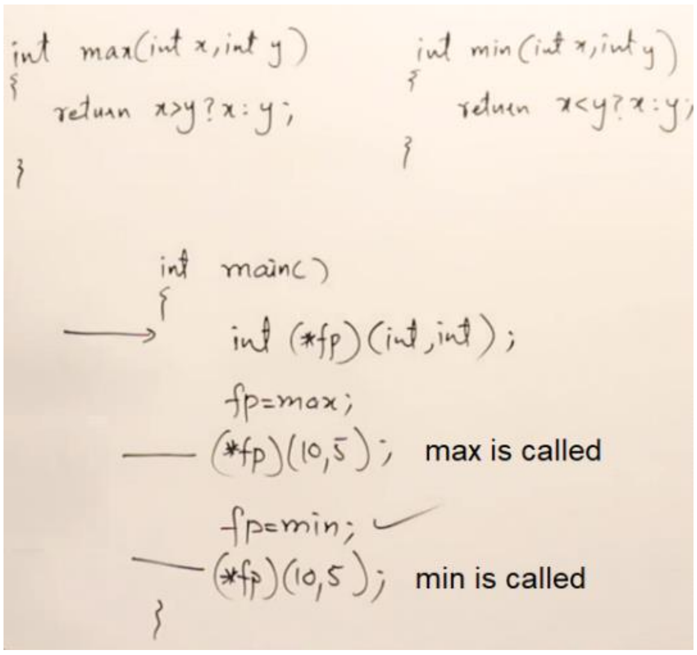
- When we declared a pointer to a function it must be inside the brackets otherwise it will not be a pointer to a function. It must be enclosed in a bracket

- Void (*fp)();   // declaration of pointer
  fp = functionName;   // initialization of pointer
  (*fp)();   // function call



```
                void display( )
                {
                    cout<<"Hello";
                }

                int main( )
                {
decl  ------>      void (*fp)( );
init  ------>      fp=display;
Call  ------>      (*fp)( );
                }
```



```
int max(int x,int y)            int min(int x,int y)
{                               {
  return x>y?x:y;                 return x<y?x:y;
}                               }

          int main( )
 ------>  {
            int (*fp)(int,int);
            fp=max;
     ----   (*fp)(10,5);   max is called
            fp=min; ✓
     ----   (*fp)(10,5);   min is called
          }
```

- **Polymorphism** :
  - Different functions are called because the pointer is pointing on different functions. So this is same name different functions different operations.
  - So this is just like polymorphism. Yes yes in function overwriting internally.
  - Function pointer are used for achieving a runtime polymorphism using function overwriting.
  - So this means that one function pointer can point on any function which is having same signature.
  - Yes a function pointer can point on all those functions which are having same signature.

- If a function is not returning any value then its return type should be void.
- Call by value will pass just values of actual parameters, they cannot be modified.
- Which type of functions can take datatype as parameters? - Template