PROGRAMMING FUNDAMENTALS

Let's explore technology
together to live in the future

Checkout more on
https://github.com/Sy-hash-collab

Sy-hash-collab

Functions

Definition :

A specific block of code that is used to perform some functionality / operation in the code.

- Functions provide modular approach to programming.
- Functions provide / divide a complete code into small meaningful components / parts / modules.
- Handling errors become easier when functions of modular programming approach is used.

Types :

1- Built in 2- User defined.

```
void display()
{
  cout << " Your Name ";
}
int main()
{
  display();
}
```

=> Function call is actually starting point of user-defined function. If missing control will never enter user-defined function.

=> Main is driver function.

Function decleration:

Function decleration means providing 3 informations to compiler of your function.

1- Function Name
2- IS Function returning a value.

As some functions perform mathematical operations, and we get some arithmetic value as a result. In this case, we will have to write the datatype of value before function name.

eg If it is returning integer value, we use int datatype.

3- Arguments / Parameters

When a function is called, in most cases it requires some values on which the function performs its functionality.

Suppose, we write a function which will add two values and returns their result. In this case, we use arguments parameters to tell compiler, that while calling this function you have to provide two values along with their datatype.

Returning Value from Function:

A function can return a single value. If function returns a value it is to be mentioned in function decleration. Datatype is mentioned in function decleration before function name, according to the returned value. We use int datatype if function return an integer value.

Keyword "return" is used in function to return a value.

Upon execution of "return" statement the control moves back to the calling functio along with returned value.

Using returned value in assignment:

The calling function can store the returned value in a variable and then use this variable in the program By Assigning a variable a value.

Function decleration:

We declare a function cube of int datatyp this means it will return integer value.

Function definition:

In this we use formal parameter num, this means that whenever function is called the value is passed to num.

Function call :

We get input from user as 'n' and assign the value which the function will return to another variable 'c' So that when cube is calculated its value is assigned to c.

```
int cube (int);
```

Function decleration

```
void main ()
{
  int n, c;
  cout << "Enter number: ";
  cin >> n;
  c = cube (n);
  cout << "Cube is = " << c;
}

int cube (int num)
{
  return num*num*num;
}
```

Function call

Actual Parameter

Formal parameter

Function definition

Using returned value in arithmetic expressio

```
int cube (int );
void main ()
{
  int n, c;
  cout << "Enter number: ";
  cin >> n;
  c = 5 * cube (n );
  cout << " Result is " << c ;
}
int cube (int num )
 {
   return num* num* num;
 }
```

Using returned value in output statement

```
int cube (int );
 void main ()
 {
    int n, c;
    cout << " Enter number: ";
    cin >> n;
    cout << "Cube is: " << cube (n);
 }
int cube (int num)
{
  return num * num * num;
}
```

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

char grade (int m);
int main()
{
   int marks;
   char g;
   cout << "Enter marks: ";
   cin >> marks;

   g = grade (marks);

   cout << "Your grade is: " << g;
}

char grade (int m).
{
   if (m > 80)
      return 'A';

   else if (m>60)
      return 'B';

   else if (m>40)
      return 'C';

   else
      return 'F';
}
```

#

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

 int sqr (int n );
 int cube (int n );

 int main ()
 { int a, b, r;
    cout << " Enter an integer: ";
    cin >> a;
    cout << " Enter an integer : ";
    cin >> b;
 }
 int sqr (int n)
 { return n*n ;
 }
 int cube n*n*n;
  { return n*n*n;
  }
```

Parameters:

Parameters are the values provided to a function when a function is called.

```
function_name (pm);
function_name (pm1, pm2);
function_name ();
```

Ways to provide parameters:

1- Using constant values.

```
function_name (10, 20);
```

2- Store values in some variables and then pass them to parameters.

```
int a = 10;
int b = 20;
function_name (a, b),
```

Sequence and types of parameters in function call must be similar to the sequence of datatypes in function dec.

```
void calc (int a, float b);
calc (10, 15.5);
int a = 10;
float b = 15.5;
calc (a, b);
```

Formal Parameters:

Parameters used in the header of function definition are called formal parameters. These parameters are used to receive values from calling function.

```
void calc (int x, int y)
```

Actual Parameters:

Parameter used in the function call are called the actual parameters.

```
void calc (10,20);
calc (a,b);
```

Passing Parameters to Function:

Providing parameters to a function is called "passing parameters to function".

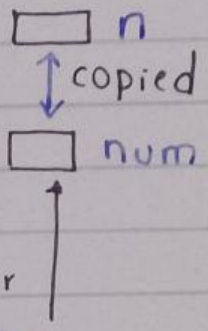
1- Pass by Value:

A mechanism in which the value of actual parameter is copied to formal parameter of called function.

If function makes any change in formal parameters it doesnot affect the values of actual parameters.

We store user input in var. 'n' and then call the func. 'show', we pass var 'n' as parameter, perform functionality., copied to variable 'num'.

```
void show (int);  ]→ Function decleration
 void main ()
{
   int n;
   cout << "Enter number: ";
    cin >> n;
    show (n); → Actual parameter
    cout << "End of Program ";
}
  void show (int num)
                    ↓
                Formal parameter
{
  cout << "The number is" <<num;
}
```




```
void cal (int a, int b, char op);
 int main ()
 {
   int x, y;
   char c;
   cout << " Enter first number , operator, sec.
   cin >> x >> c >> y;                    number: ";
   cal( x , y, c);
}
  void cal (int a, int b, char op)
  {
    switch (op)
    {  case '+':
         cout <<a<< "+" <<b << " = " <<a+b;
         break;
```

```
case '-':
    cout << a << "-" << b << "=" << a-b;
    break;
case '*':
    cout << a << "*" << b << "=" << a*b;
    break;
case '/':
    cout << a << "/" << b << "=" << a/b;
    break;
default:
    cout << "Invalid operator.";
}
```

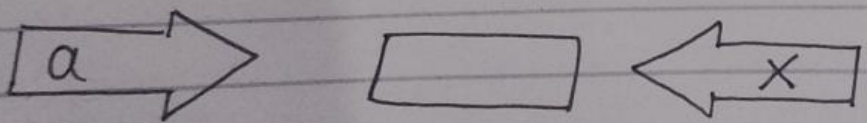
Parameters Pass by Reference:

A parameter passing mechanism in which adress of actual parameter is passed.

The ampersand (&) sign is used with formal parameters to pass parameters by reference.

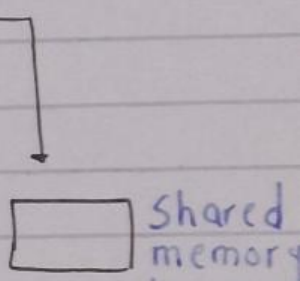
No separated memory locations are reserved for formal parameters if parameters pass by reference mechanism is used.

Formal parameters become second name of actual parameters.



If values are changed in formal parameters during function execution will change the values in actual parameters also.

```
void show (int &
void main ()
{
  int n;
  cout << "Enter number:";
  cin >> n;
  show (n);
  cout << " End of Program
}
void show (int & num)
{
  cout << "The number is "<<num;
}
```




```
void swap (int&x , int & y);
int main ()
{
  int a, b;
  cout << " Enter integer"; cin >> a;
  cout << "Enter integer"; cin >> b;

  cout << " Values before swapping:";
  cout<<" a= "<<a<<endl;
  cout << " b = " << b <<endl;
```

```
cout << "Values Swapping:" <<endl;
 swap (a,b);

 cout << " Values after swapping:";
 cout << "a=" <<a <<endl;
 coul- << "b=" << b <<endl;

} void swap (int &x , int &y)
 {  int t;
    t = x;
    x = y;
    y = t;
 }
```