Functions

Functions

- A group of statements that perform a specified operation is called a function.
- In a function we group several program statements into a unit and give that unit a name that is called function name.
- Function can be invoked any where in the program.
- Program statements that appear in the program more than once are suitable for creating a function.
- Function code is stored in only one place in the memory.
- Another reason for creating functions is that a complex or bigger program code is divided into different functions due to which it becomes easy to manage the program.

Functions

- There are 3 things important related to a function.

i) Function Declaration

ii) Function Calling

iii) Function Definition

```
#include<iostream.h>
#include<conio.h>
void line(void); //Function Declaration
void main(void)
{       clrscr();
        line(); //Function Calling
        cout<<"Hello"<<endl;
        line();
        cout<<"We are studying functions"<<endl;
        line();
        getch();
}
void line(void) //Function Declarator
{       for(int a=1;a<=20;a++)
                cout<<"*";
        cout<<endl;
}
```

Output

****

Hello

****

We are studying functions

****

Eliminating the Declaration

```
#include<iostream.h>
#include<conio.h>
    void line(void) //Function Definition without Declaration
    {
        for(int a=1;a<=20;a++)
          cout<<"\xCD";
          cout<<endl;
    }
void main(void)
    { clrscr();
    line();
    cout<<"Hello"<<endl;
    line();
    cout<<"We are studying functions"<<endl;
    line();
    getch();
}
```

Passing by Value

- An argument is a piece of data, i.e. a value passes from program to function
- These passed values etc can be used by the function according to the requirements.
- There are two ways in Passing by Value, through which arguments can be passed to the functions, i.e.

I. **Passing Constants to functions**
II. **Passing Variables to functions**

Passing Constants to functions

- As the name represents, In passing constants to functions, a character, integer or float constant is actually passed as argument to the function, i.e.
  - **line('*');**
  - **square(5);**

```
#include<iostream.h>
#include<conio.h>
void line(char); //Function Declaration
void main(void)
{       clrscr();
        line('*'); //Function Calling
        cout<<"Hello"<<endl;
        line('-');
        cout<<"We are studying functions"<<endl;
        line('*');
        getch();
}
void line(char ch) //Function Declarator
{       for(int a=1;a<=20;a++)
                cout<<ch;
        cout<<endl;
}
```

Output

****

Hello

We are studying functions

****

Passing Variables to functions

```
#include<iostream.h>
#include<conio.h>
void chline(char, int);

void main(void)
{     clrscr();
       char ch;
        int n;
       cout<<"Enter a character ";
       cin>>ch;
       cout<<"Enter a value ";
       cin>>n;
       chline(ch, n); //Character and Integer variables passed
       cout<<"Hello"<<endl;
       chline(ch, n);
       cout<<"We are studying functions"<<endl;
       chline(ch, n);
       getch();
}
void chline(char ch, int n)
{
     for(int a=1;a<=n;a++)
             cout<<ch;
     cout<<endl;
}
```

Output

```
Enter a character +
Enter a value 10
++++++++++
Hello
++++++++++
We are studying
functions
++++++++++
```

Passing Variables to functions

#include <iostream> using namespace std; int addition (int a, int b) { int r; r=a+b; return (r); } int main () { int z; z = addition (5,3); cout << "The result is " << z; }

```
#include<iostream.h>
#include<conio.h>
struct Distance
{      int feet;
       float inches;
};
void showDistance(Distance);

void main(void)
{      clrscr();
       Distance d1, d2;
       cout<<"Enter feet for 1st Distance ";
       cin>>d1.feet;
       cout<<"Enter inches for 1st Distance ";
       cin>>d1.inches;
       cout<<"Enter feet for 2nd Distance ";
       cin>>d2.feet;
       cout<<"Enter inches for 2nd Distance ";
       cin>>d2.inches;
       cout<<"\nFirst Distance is ";
       showDistance(d1);
       cout<<"\nSecond Distance is ";
       showDistance(d2);
       getch();
}

void showDistance(Distance dd)
{
       cout<<dd.feet<<"\'-"<<dd.inches<<"\"";
}
```

Output

Enter feet for 1st Distance 5
Enter inches for 1st Distance 6.5
Enter feet for 2nd Distance 7
Enter inches for 2nd Distance 8.5
First Distance is 5'-6.5"
Second Distance is 7'-8.5"

Function Return

```
#include <iostream>
int  addition (int a, int b)
{
      int  r;
      r=a+b;
      return r;
}
int  main ()
{
      int  z;
       z = addition (5,3);
       cout << "The result is " << z;
}
```

```
#include <iostream>
int subtraction (int a, int b)
{
    int  r;
     r=a-b;
    return r;
}
int main ()
{
int x=5, y=3, z;
z = subtraction (7,2);
cout << "The first result is " << z << '\n';
 cout << "The second result is " << subtraction (7,2) << '\n';
cout << "The third result is " << subtraction (x,y) << '\n';
z= 4 + subtraction (x,y);
cout << "The fourth result is " << z << '\n'; }
```

```
The first result is 5
The second result is 5
The third result is 2
The fourth result is 6
```

Function Return

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

float p2k(float);

void main(void)
{ clrscr();
      float pounds, kilograms;
      cout<<"Enter weight in Pounds ";
      cin>>pounds;
      kilograms = p2k(pounds);
      cout<<pounds<<" Pounds = "<<kilograms<<" Kilograms";
      getch();
}
    float p2k(float pounds)
    {
      float kilograms = 0.453592 * pounds;
      return kilograms
      }
```

Output

Enter weight in Pounds 200
200 Pounds = 90.718399 Kilograms

```
return 0.453592 * pounds ;

or

return (0.453592 * pounds) ;
```

Function Return

- When we call a function than one or more arguments can be sent to a function, but a function can only return one argument.
- Multiple arguments can not be returned from a function using return statement, but can be made to do so if our return variable is of structure type.
- Functions return type should always be included in the function declaration.
- If function doesn't return anything then we use **void** to indicate that function will not return a value.
- If a functions return type is not specified in the function declaration then the compiler assumes that functions return type is **int**.
- So, if a function is declared as, i.e. **test();** then it means that the function will return an integer type of value on completion, as in its declaration, no return type was specified.
- It is better that we always specify the return type even if it is **int**. This habit makes the program listing consistent and readable.

Returning Structure Variables

```
#include<iostream.h>
#include<conio.h>
struct Distance
{ int feet;
float inches;
};
Distance sumDistance(Distance, Distance);
void showDistance(Distance);
void main(void)
{ clrscr();
    Distance d1, d2, d3;
    cout<<"Enter feet for 1st Distance ";
    cin>>d1.feet;
    cout<<"Enter inches for 1st Distance ";
    cin>>d1.inches;
    cout<<"Enter feet for 2nd Distance ";
    cin>>d2.feet;
    cout<<"Enter inches for 2nd Distance";
    cin>>d2.inches;
d3 = sumDistance(d1, d2);
cout<<"\nFirst Distance is ";
showDistance(d1);
cout<<"\nSecond Distance is ;
showDistance(d2);
cout<<"\nSum of Distance is”;
showDistance(d3);
getch();
}
```

Returning Structure Variables

```
Distance sumDistance(Distance dd1, Distance dd2)
{    Distance dd3;
     dd3.inches = dd1.inches + dd2.inches;
     dd3.feet = 0;
        if(dd3.inches >= 12.0)
        {
          dd3.inches -=12.0;
          dd3.feet++;
        }
     dd3.feet += dd1.feet + dd2.feet;
     return dd3;
}
void showDistance(Distance dd)
{
    cout<<dd.feet<<"'-"<<dd.inches<<"\"";
    cout<<endl;
}
```

Output

```
Enter feet for 1st Distance 5
Enter inches for 1st Distance 6.5
Enter feet for 2nd Distance 8
Enter inches for 2nd Distance 9.5
First Distance is 5'-6.5"
Second Distance is 8'-9.5"
Sum of Distance is 14'-4"
```

Passing by Reference

In passing arguments by reference, instead of passing a value to the function, its reference that is the address of that variable is passed.

Passing by reference has two main advantages, i.e.

1. Function can access the actual variables of the calling function.
2. Provides a mechanism for returning more than one value from the called function to its calling function.

Passing by Reference

```
#include<iostream.h>
#include<conio.h>
void swap(int&, int&);

void main(void)
{ clrscr();
int a, b;
cout<<"Enter value for a ";
cin>>a;
cout<<"Enter value for b ";
cin>>b;
cout<<"\nBefore Swapping"<<endl;
cout<<"A is "<<a<<" and B is "<<b<<endl;
swap(a, b);
cout<<"\nAfter Swapping"<<endl;
cout<<"A is "<<a<<" and B is "<<b<<endl;
getch();
}
```

Output

```
Enter value for a 10
Enter value for b 20
Before Swapping
A is 10 and B is 20
After Swapping
A is 20 and B is 10
```

Function has not returned any value but even then the variables in the main function have changed value

```
void swap(int& aa, int& bb)
{
    int t = aa;
    aa = bb;
    bb = t;
}
```

Passing Structures by Reference

```
#include<iostream.h>
#include<conio.h>
struct Distance
      {    int feet;
           float inches;
      };
void scale(Distance&, float);
void show(Distance);
void main(void)
{    clrscr();
     Distance d1;
     cout<<"Enter feet for distance ";
     cin>>d1.feet;
     cout<<"Enter inches for distance";
     cin>>d1.inches;
     cout<<"Distance is ";
     show(d1);
     scale(d1,0.5);
     cout<<"Distance now is ";
     show(d1);
     getch();
}
```

```
void scale(Distance& d, float factor)
{ float inches = (d.feet*12 + d.inches ) * factor;
   d.feet = inches / 12;
   d.inches = inches - d.feet * 12;
}

void show(Distance d)
{
 cout<<d.feet<<"\'-"<<d.inches<<"\"";
 cout<<endl;
}
```

Output

```
Enter feet for distance 10
Enter inches for distance 6.6
Distance is 10'-6.6"
Distance now is 5'-3.299999"
```

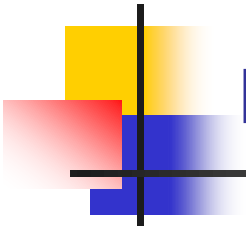
Overloaded Functions
Or
Function Overloading

- Overloaded function or Function overloading means that more than one function with the same name exists in the program but differing in the number of arguments.
- When the function will be called, then number of arguments will decide that which function will be actually called, i.e.

Function Overloading Cont…

```
void line();
void line(int);
void line(char);
void line(int, char);
void line(char, int);
```

- We can see the above mentioned declarations that all five functions have the same name, i.e. **line**, but every functions prototype is different from one another.
- Similarly, when we'll call the function line than its number of arguments will decide, which function to execute.



Function Overloading Cont…

- Function definition doesn’t need to be in sequence the way functions are declared,
- but only requirement is that the number of function definitions should be equal to the number of function declarations.
- In overloaded functions, the compiler can distinguish even if we provide different types of arguments in the functions.

Function Overloading

```
#include<iostream.h>
#include<conio.h>
void line(void);
void line(int);
void line(char);
void line(int, char);
void line(char, int);

void main(void)
{     clrscr();
      line(10);
      line();
      line('=',15);
      line('*');
      line(20,'-');
      getch();
}
void line(void)
{ for(int a=1;a<=10;a++)
cout<<"*";
cout<<endl;
}
void line(int n)
{ for(int a=1;a<=n;a++)
            cout<<"*";
            cout<<endl;
}
void line(char c)
{ for(int a=1;a<=10;a++)
            cout<<c;
            cout<<endl;
}
void line(int n, char c)
{ for(int a=1;a<=n;a++)
            cout<<c;
            cout<<endl;
}
void line(char c, int n)
{ for(int a=1;a<=n;a++)
            cout<<c;
            cout<<endl;
}
```

Default Arguments in Function Declaration and Calling

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

void line(char='*', int=20);

void main(void)
{
    clrscr();
    line();
    line('=');
    line('-',10);
    getch();
}
void line(char ch, int n)
{
    for(int a=1;a<=n;a++)
    cout<<ch;
    cout<<endl;
}
```

Inline Functions

```
#include<iostream.h>
#include<conio.h>
inline float p2k(float pounds) //inline function
{ return 0.453592 * pounds;
}
void main(void)
{
      clrscr();
       int pounds;
       cout<<"Enter weight in pounds ";
       cin>>pounds;
       cout<<pounds<<" Pounds = "<<p2k(pounds);
       cout<<" Kilograms";
       getch();
}
```

Output

Enter weight in pounds 180
180 Pounds = 81.646561 Kilograms