User Defined Functions

Introduction

- Large programs is broken down into smaller units known as functions.
- **A function is a sub program that acts on the data passed into the function and often returns a value.**
- **Function helps reduce the program size and avoid ambiguity.**

Types of Functions

- Built in functions
- User defined functions

Built in functions

- These functions are part of the compiler package, these are part of the library made available by the compiler.
- Eg exit(), pow(),sqrt()

User-defined functions

- These are created by you i.e, the programmer, as per requirement of your program.

All function must have the following:

- Function prototype / function declaration
- Function definition
- Function call

Function Prototype

A function prototype is a declaration of the function that tells the program about the type of the value returned by the function and the number and type of arguments(parameter) passed.

Eg: int sum() ;

int sum(int , int);

General form of function prototype is:

Type function-name (parameter list)

Eg *float volume(int , float);*

return type – float (a float value will be returned)

function name- volume

parameter – 2 (one int and one float)

All Functions has 3 part

- Return type
- Function name
- Parameter
- Eg void sum ( )

return type – void - no return type

function name- sum

parameter - Nil

Function call

- How will you call the function created by you in the program?
- It can be done using function call

Eg assume prototype as

```
void sum (int ,int );
```

Function call: sum(a,b);

Write the function prototype and function call for the following

1. Return type is void ,function name add, 1 float parameter.

Ans: void add(float);

Call: add(a); //assume a is declared

Write the function prototype and function call for the following

1. Return type is float ,function name even, 2 float parameter.

Prototype: float even(float,float);

Call: float sum=even(a,b);

//assume a,b is already declared.

Write the function prototype and function call for the following

1. Return type is int ,function name sum, 2 float parameter and one int

Prototype: int sum(float,float,int);

Call: int a=sum(c,d,e);

Function definition (shoud be written outside main)

```
void sum()
{ int a,b,sum;
 cin>>a>>b;
sum=a+b;
cout<<sum;
}
```

```
int sum()
{ int a,b,sum;
 cin>>a>>b;
sum=a+b;
cout<<sum;
return sum;
}
```

Function definition

```
void sum(int a,int b)
{ int a,b,sum;
 cin>>a>>b;
int   sum=a+b;
cout<<sum;
}
```

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

Function prototype & function definition

- Function prototype

```
float volume(float ) ;     // prototype ends with semi colons
```

- Function call:

```
Float ans=volume(4.5);
```

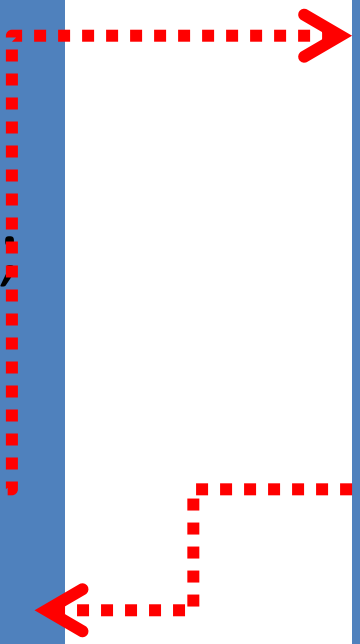
- Function definition

```
float volume (float a)
{  float n;
   n =a*a*a;
   return n; }
```

Program to print cube of a given number using function

```
void main()
{ float a;
 void cube(float a);
 cin>>a;
cube(a);
cout<<" Job Done:";
}
```

```
void cube(float b)
{ float c
 c=b*b*b;
 cout<<"cube"<<c;
}
```

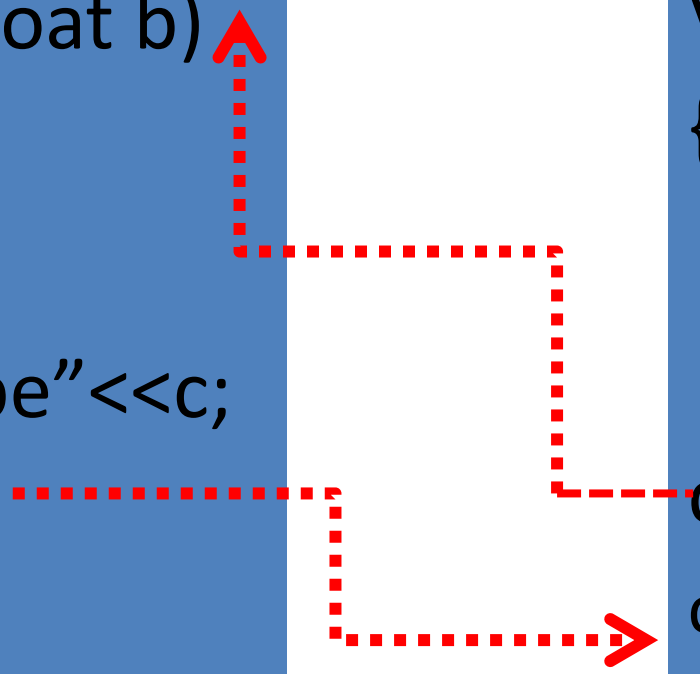


- Note a function prototype is not needed if the definition of the function is given before the function call (ie before void main).

Program to print cube of a given number using function

```
void cube(float b)
{ float c
 c=b*b*b;
 cout<<“cube”<<c;
}
```

```
void main()
{ float a;
 void cube(float a);
 cin>>a;
cube(a);
cout<<“ Job Done:”;
}
```



Multiplication of 2 numbers

// without parameter and no return type

```
 FP : void multiply();
FC : multiply();
FD : void multiply()
{ int a,b,p=0;
 cout<<“ENTER THE NUMBER :”;  cin>> a>>b;
 p=a*b;
cout<<“\n The product is :”<<p;
}
```

// with parameter & without return type

FP – void multiply(int,int);

FC: multiply(a,b);

FD : void multiply(int a,int b)

```
 {
   int p=a*b;
    cout<<“ Product =“<<p;
}
```

// without parameter & with return type

FP – int multiply();

FC: int p = multiply();

FD : int multiply()

```
 { int a,b;
   cin>>a>>b;
   int p=a*b;
    return(p);  // or return(a*b);
}
```

// with parameter & with return type

FP – int multiply(int,int);

FC: int p=multiply(a,b);

FD : int multiply(int a,int b)

```
 {
   int p=a*b;
    return p;
}
```

1 . W.A.P Using function called sumEven to return the sum of even numbers and accept the limit as the parameter.

```
void main()
{ int n,se=0;
 int sumEven(int); //Prototype
 cin>>n;
se=sumEven(n);  //Function call
cout<<“sum of even=“<<se;
}
```

```
float sumEven(int a);
{ int c=0,sum=0;
  while(c<=a)
 { if(c%2==0)
    sum+=c;
    c++;
  }
return (sum);
}
```

QUESTIONS

1. W.A.P using functions (with parameter & return type) to calculate the area of a circle.

2. W.A.P using functions (with parameter & no return type) for converting cm to feet & inches [1 ft=12 inches, 1 inch=2.54cm]

3. W.A.P to accept 2 nos M,n return the value M^n (using with only one parameter and return type)

4. W.A.P to accept X from a user & return square of X.

5. W.A.P using function to find the average of 3 given numbers.

6. W.A.P using function to accept a number from the user and find the factorial of a number.

7. W.A.P using function to accept a number from the user and returns 1 if the number is positive else 0 if the number is negative.

8. W.A.P using function to accept a number from the user and print the stars in the ascending order.

9. W.A.P using function to accept a number N from the user and find the Fibannaci series till the term N.

10. W.A.P using function to accept a string from the user and pass each character into the function and count the number of digits in the string.

11. W.A.P using function to enter the mark of a student return the grade back.

(100-90 ‘A’,
75-90 ‘B’,
50-75 ‘C’,
ELSE ‘D’)

FORMAL &ACTUAL PARAMETER

- Actual parameter is a parameter that is used in function call statement to send the value to the function.
- Formal parameter is a parameter which is used in the function header of the called function to receive the value from the actual parameter.

```
void main()
{ void multiply(int);
  int a;
 cin>>a;
multiply(a);
// a is a actual parameter sent to
                    //multiply function
}
void multiply( int c)     // c is the formal parameter

{ cout<<c*c; }
```

Consider the following code and find the output

```
void main()
{ for(int i=0;i<3;i++)
  { for(int j=0;j<3;j++);
  }
cout<<“i=“<<i<<“j=“<<j;
 }
```

Consider the following code and find the output

```
void main()
{ for(int i=0;i<3;i++) // var i has scope or life only
                                    inside main program
  for(int j=0;j<3;j++); // var j is a local variable for that
                        // particular compound stmt or block
{……….}

cout<<“i=“<<i<<“j=“<<j; //Will give you an error

 }
```

Will the following code execute ? Justify your answer

```
void main()
{ int a,x=9;
 cin>>a;
 void multiply(a);
}
void multiply(int c)
{ cout<<c*x; }
```

Will the following code execute ? Justify your answer

```
void main()
{ int a,x=9;
 cin>>a;
 void multiply(a);
}
void multiply(int x)
{ int c=20;
cout<<c*x; }
```

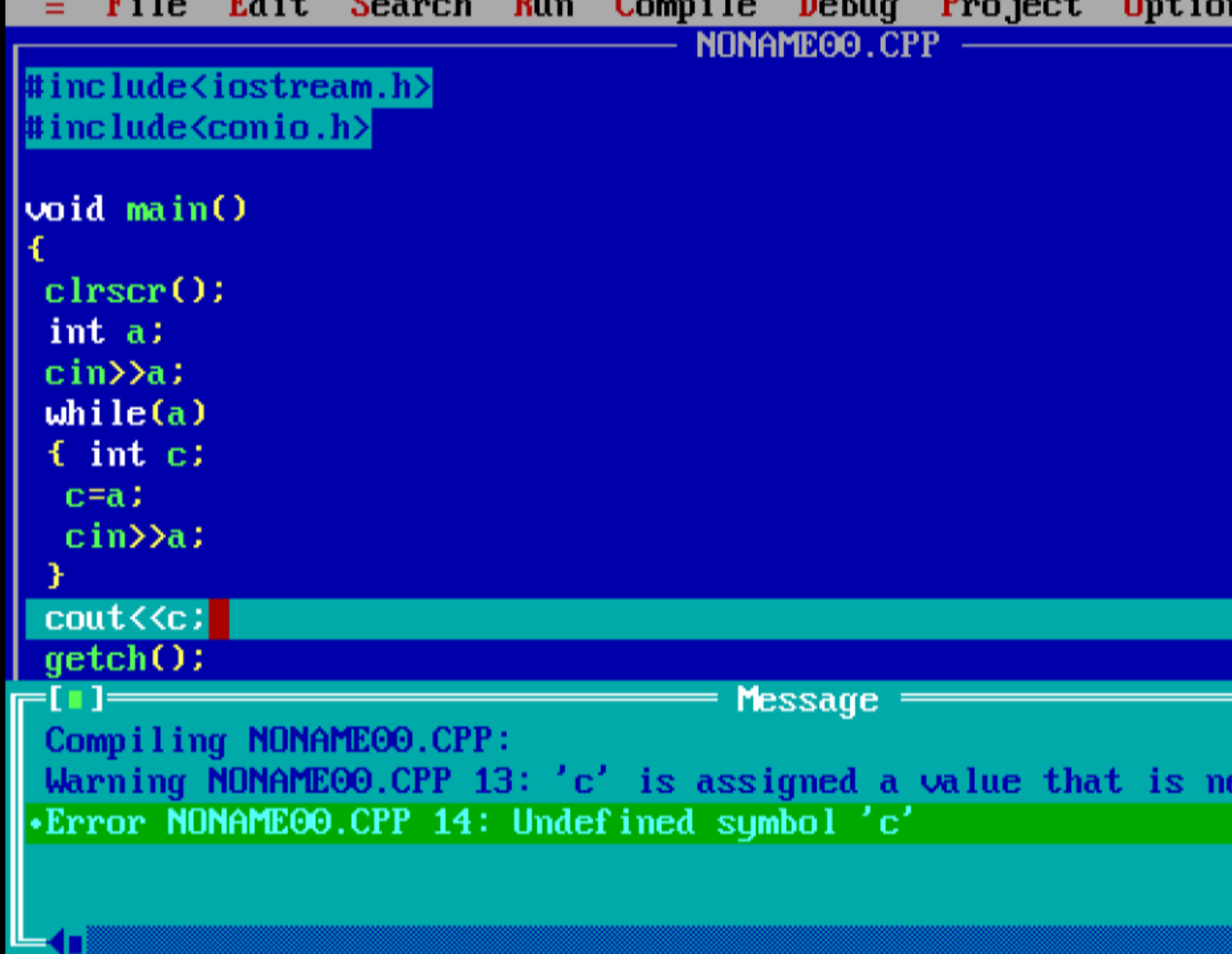
Variable scope

- A variable scope : The program part(s) in which a particular piece of code or a variable can be accessed is known as variable scope.

There are 4 kinds of scopes in c++

1. Local scope : A variable declared in a block is local to that block and can be used only in it and other blocks contained under it.

```
void main()
{ int a;
 cin>>a;
 while(a!=0)
 { int c;
    cin>>a;       //c has local scope only inside while
   c=a;}   // a has a local scope even inside file
  cout<<c; }   //gives you error as scope of c is only
                  inside while block
```


File Edit Search Run Compile Debug Project Optio
NONAME00.CPP
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
int a;
cin>>a;
while(a)
{ int c;
c=a;
cin>>a;
}
cout<<c;
getch();
Message
Compiling NONAME00.CPP:
Warning NONAME00.CPP 13: 'c' is assigned a value that is n
•Error NONAME00.CPP 14: Undefined symbol 'c'

2. Function scope : The variables declared in the outermost block of a function have function scope i.e they can be accesssed only in the function that declares them. (Label in goto also have function scope).

```
void main()
{  int add;
  void sum(int ,int);
  int a=3,b=5;
 sum(a,b);
 }
```

```
 void sum(int c ,int d)
 { add=c+d;
   cout<<add;
}
```

≡ File Edit Search Run Compile Debug Project Options Window He

NONAME00.CPP 1

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

void main()
{
 int add;
 void sum(int,int);
 int a=3,b=5;
 sum(a,b);
 getch();
}

void sum(int c, int d)
{ add=c+d;
cout<<add;
```

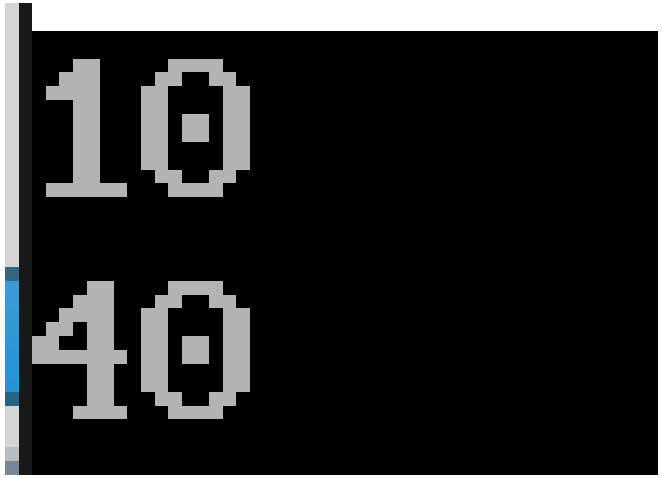
[■] Message 2

```
 Compiling NONAME00.CPP:
•Error NONAME00.CPP 14: Undefined symbol 'add'
 Warning NONAME00.CPP 16: Parameter 'c' is never used
 Warning NONAME00.CPP 16: Parameter 'd' is never used
```

3. File scope : A name declared outside all blocks and function has file scope, i.e it can be used in all the blocks and functions written inside the file. (also known as global variables).

```
#include<iostream.h>
int a=20;
void main()
{ a=10;
   cout<<a<<endl;
  void sum();
sum();
}
```

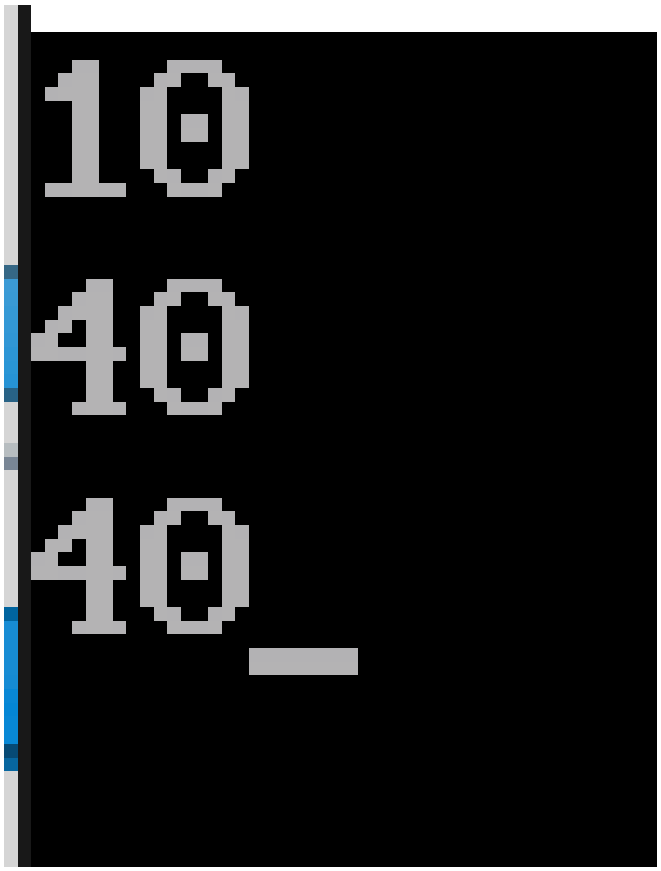
```
void sum()
{ a=a+30;
   cout<<a<<endl;
}
```


10
40

3. File scope : A name declared outside all blocks and function has file scope, i.e it can be used in all the blocks and functions written inside the file. (also known as global variables).

```
#include<iostream.h>
int a;
void main()
{ a=10;
   cout<<a<<endl;
  void sum();
sum();
cout<<a;

}
```

```
void sum()
{ a=a+30;
   cout<<a<<endl;
}
```


10
40
40_

Global variable/ Prototype

If a function/variable is declared inside the main function then the function/variable can be used only by the main program. Whereas if you declare the function/variable globally then it can be used by all the functions in the program,

```
#include<iostream.h>
int a=10;   //Global Variable
void main()
{ int b=15;   //local variable
  cout<<a<<“\t”<<b; }
```

output

10 15

```
#include<iostream.h>
int a=10;
void main()
{ int a=15;
  cout<<a<<"\t"<<a; }

output
  15   15

//Actually which variable is being called
 global or local????
```

This will display the local variable. This is because the local and global variable has the same name. Thus the local variable a hides the global variable a.

To unhide global variable :: (scope resolution) operator has to be used.

```
#include<iostream.h>
int a=10;
void main()
{ int a=15;
  cout<<::a<<"\t"<<a; }
```

output

10 15

LOCAL VARIABLE	GLOBAL VARIABLE
1. It is a variable that is declared with in a function or within a block	1. It is variable which is declared outside all the functions
2. It is accessible only within a function/block in which it is declared	2. It is accessible throughout the program

```
#include<iostream.h>
float val=10.0;
void value(int t)
{ int res=2;
  return(res*t); }
void main()
{  cout<<val;
    int b=value(2);
    cout<<b;
}
```

local variable ??? Global variables ????

How to declare a constant?

```
void main()
{
int a=10;

cout<<a;   // 10 will be displayed
a=a+10;
cout<<a;
}
```

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

void main()
{
 clrscr();
 const int a=10;
 cout<<a<<endl;
 a=a+10;
 cout<<a;
 getch();
}
```

9:4

Message

Compiling NONAME00.CPP:

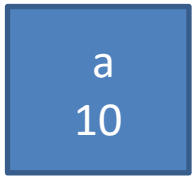
•Error NONAME00.CPP 9: Cannot modify a const object

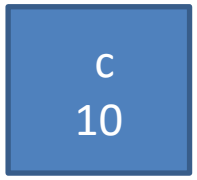
A function can be called invoked in 2 ways call by value & call by reference

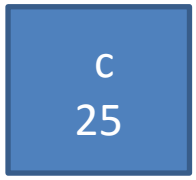
During call by value, any change made to the formal parameter is not reflected back to the actual parameter.

```
void main()
{ void change( int );
  int a=10;
  change(a);
  cout<<a;
}
```

```
void change( int c);
  { cout<<c<<endl
 c=25;
cout<<c<<endl;
}
```



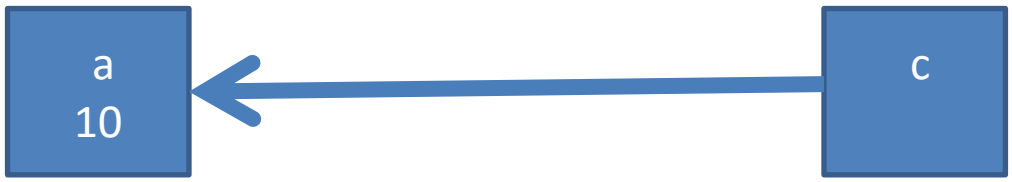




- During call by reference, changes made in the formal parameter are reflected back to the actual paramter.

```
void main()
{ void change( int & );
  int a=10;
  change(a);
  cout<<a;
}
```

```
void change( int &c)
  { cout<<c<<endl
 c=25;
cout<<c<<endl;
}
```



```
void main()
{ void swap( int , int );
  int a,b;
  cin>>a>>b;
  swap(a,b);
  cout<<a<<b;
}
```

```
void swap( int c, int d);
  { int temp;
temp=c;
c=d;
d=temp;
cout<<c<<d;
}
```

```
void main()
{ void swap( int &, int &);
  int a,b;
  cin>>a>>b;
  swap(a,b);
  cout<<a<<b;
}
```

```
void swap( int &c, int &d);
  { int temp;
temp=c;
c=d;
d=temp;
cout<<c<<d;
}
```

There are 3 types of functions

- Computational function –that calculate and compute some value, they always returns a value.
- Manipulative function- That return a success(1) or a failure(0) eg: isupper, islower
- Procedural function- That performs an action and have no explicit return value eg: exit(0), getch()

DEFAULT ARGUMENTS

- C++ allows default values to a function parameters which is useful in case a matching argument is not passed in the function call statement.

- Eg void sum(int a,int b=5) //in definition
- void interest(float prin=10, int time=5,int rate=9)

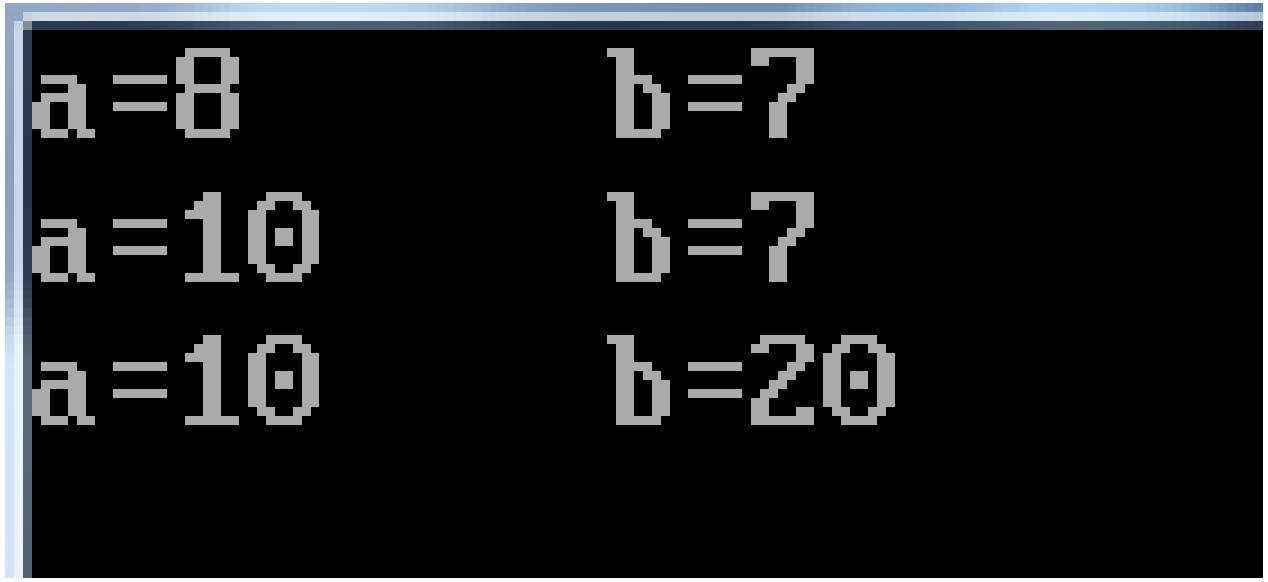
≡ File Edit Search Run Compile Debug Project Options Win

[■] NONAME00.CPP

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

void sum(int a=8,int b=7)
{ cout<<"a="<<a<<"\t"<<"b="<<b<<endl; }

void main()
{clrscr();
 int a=10,n=20;
 sum();
 getch();
}
```

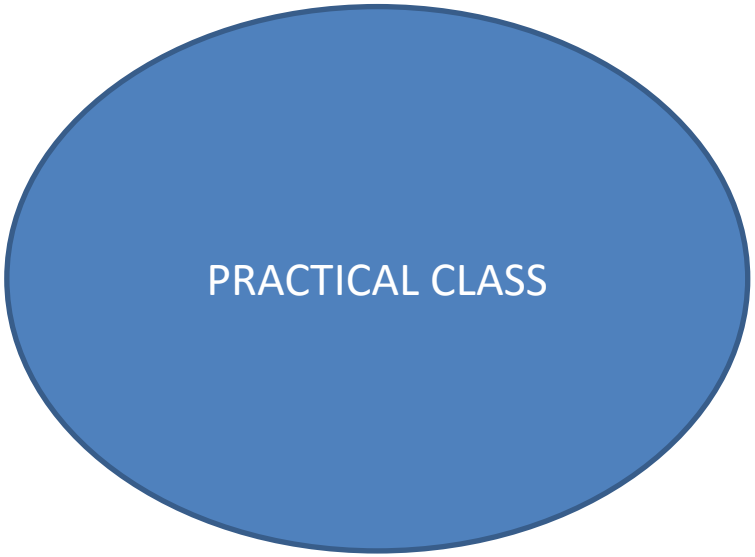

a=8
b=7

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

void sum(int a=8,int b=7)
{ cout<<"a="<<a<<"\t"<<"b="<<b<<endl; }

void main()
{clrscr();
 int a=10,n=20;
 sum();
 sum(a);
 sum(a,n);
 getch();
}
```


a=8 b=7
a=10 b=7
a=10 b=20

Predict the output

```
Void main()
{ cout<<interes(6100,1)<<endl;
```


PRACTICAL CLASS

IDENTIFIERS

- Can be any variables, keywords, literals, punctuators, operators

Keywords

- They are the words having special meanings or reserved words by programming language.
- eg: if, while, int, short, void, char…..

Literals : are often referred to as constants that never change their value during execution.

How to declare a constant?

```
void main()
{ const int a=10;
cout<<a;   // 10 will be displayed
a=a+10;
cout<<a;
}
```

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

void main()
{
 clrscr();
 const int a=10;
 cout<<a<<endl;
 a=a+10;
 cout<<a;
 getch();
}
```

9:4

Message

Compiling NONAME00.CPP:

•Error NONAME00.CPP 9: Cannot modify a const object

Punctuators

- eg ( ) // parameters in a function
- ; //end of a stmt
- = // assignment
- [ ] //array
- { } //compound stmt
- : // for label goto
- # // pre-processor directives

operators

- Arithmetic
- shift operator (<< , >>)
- logical operator
- Assignment operator (= ,+=,-=)
- Relational operator
- Conditional operator

UNARY, BINARY & TERENARY OPERATORS

- Unary (operators that act on one operand are referred to as unary operators)

  eg :++ , --, +,-

- Binary (operators that act upon two operands are referred to as unary operators)

  eg: a*b, a/8

- Terenary

  eg Conditional operator

What do you mean by cascading of I/O operators

- The multiple use of input(>>) or output(<<) in one statement is known as cascading of I/O operators.
- eg cin>>a>>b>>c;

cout<<“a=“<<a<<“b=“<<b<<“c=“<<c;

ctype.h

We can use inbuilt functions like

toupper() ,tolower(), isupper,islower

to change and see if a particular character is in upper case or not.

```
void main()
{
 clrscr();
 char a;
 cin>>a;
 if(isupper(a))
  a=tolower(a);
 else
  a=toupper(a);
  cout<<"\n"<<a;
 getch();
}
```

Program to exchange(swap 2 values using functions) such that

```
void main()
{ int a,b;
 cin>>a>>b;
swap(a,b)
cout<<a<<b;
}
```

```
void swap()
{   …….
  …..
cout<<a<<b;
}
```

Type Conversion –

The process of converting one predefined type into another is called as type conversion

i. Implicit type conversion :

An implicit type conversion is a conversion performed by the compiler without programmer's intervention.

ii. Explicit type conversion:

The implicit conversion of an operand to a specific type is called type conversion

eg int a,b;

float c;

c= (float)(a+b/3);