BENNETT UNIVERSITY

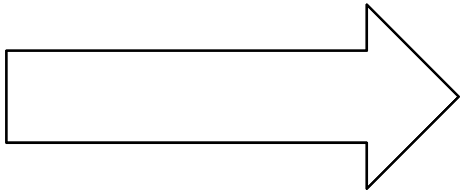
TIMES OF INDIA GROUP

CSET334 - Programming using C++

Module 2: FUNCTIONS

FUNCTIONS

```
Void main()
{
Statement 1;
Statement 2;
Statement 3;
.
.
.
.
Statement n;
}
```



```
Void main()
{
Statement 1;
Statement 2;
Sum1();
Statement 3;
Statement 4;
Sum2();
Statement 5;
Statement 6;
}
```

Advantages

- Support for modular programming
- Reduction in program size.
- Code duplication is avoided.
- Code reusability is provided.
- Functions can be called repetitively.
- A set of functions can be used to form libraries.

Types

1.Built in functions :-

are part of compiler package.

Part of standard library made available by compiler.

Can be used in any program by including respective header file.

2. User defined functions:-

Created by user or programmer.

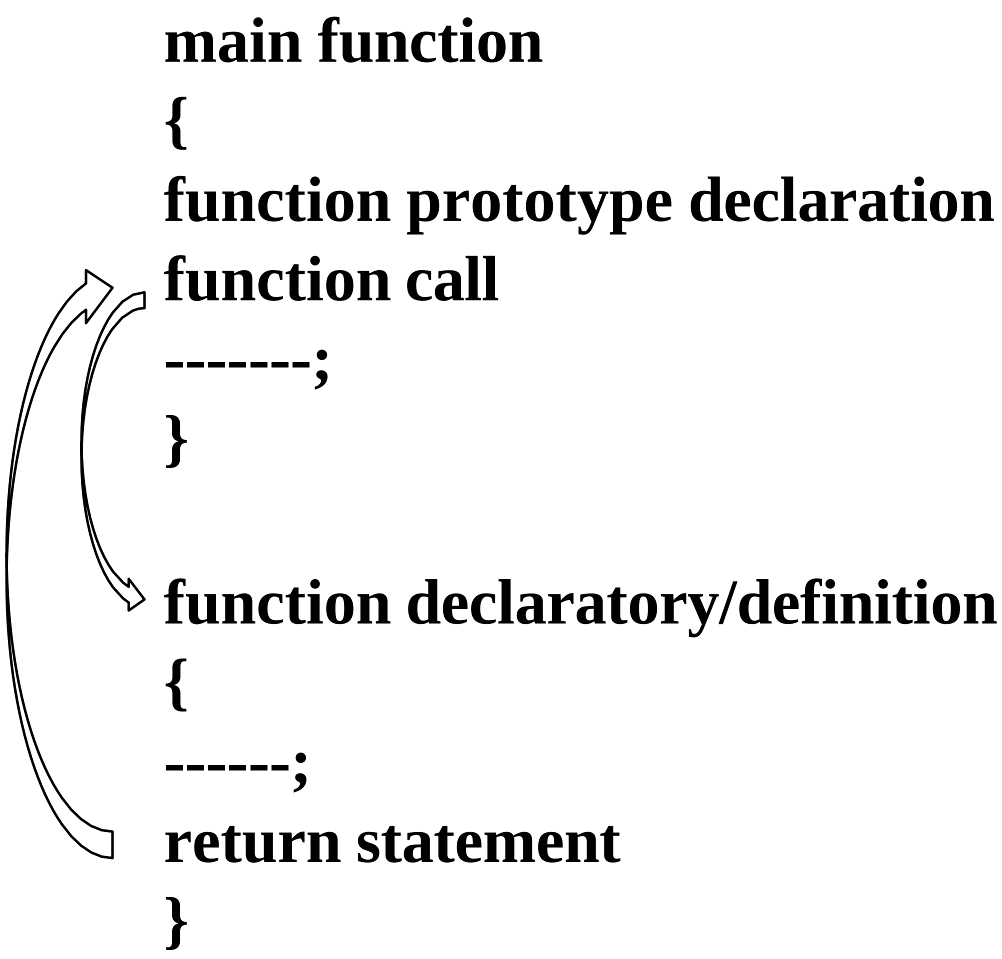
Created as per requirement of the program.

User defined function

```
Void main()
{
Statement 1;
Statement 2;
multiply();
Statement3;
----------;
----------;
Sum();
return0;
}

multiply()
{
----;
}
```

Parts of a function



```
main function
{
function prototype declaration
function call
-------;
}

function declaratory/definition
{
------;
return statement
}
```

Function prototype

A function prototype is a declaration of a function that tells the program about the type of value returned by the function, name of function, number and type of arguments.

```
Syntax:      Return_type    function_name (parameter list/argument);

                            int     add(int,int);
                            void   add(void);
                             int      add(float,int);
```

4 parts

i. Return type
ii. Function name
iii. Argument list
iv. Terminating semicolon

Variable declaration

Data_type variable_name ;

int x=5;

float marks;

int price;

Function call

A function must be called by its name followed by argument list enclosed in semicolon.

Syntax: function_name (parameter list/argument);

add(x,y);

add(40,60);

add(void); or add();

Note: data type not to be mentioned.

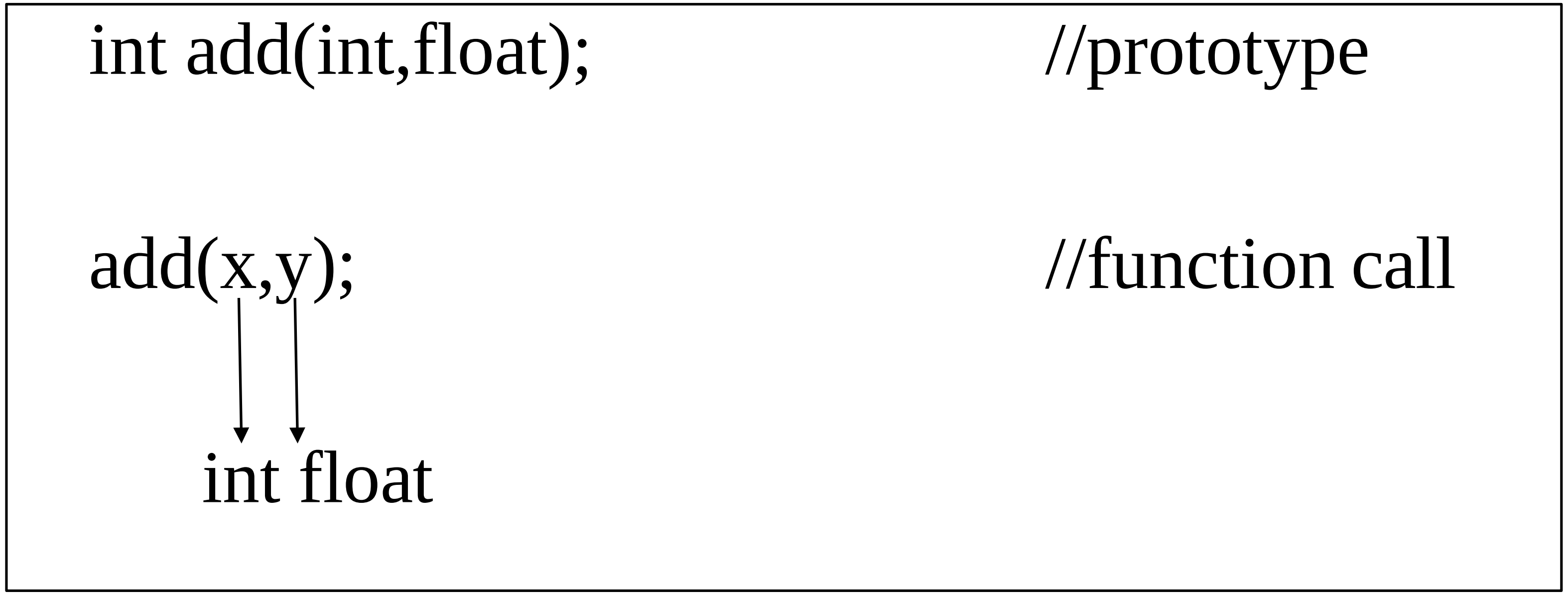
Suppose int add(int,int); //prototype

Now to this function add(x,y); //function call

or

add(40,60);

Suppose



```
int add(int,float);                    //prototype

add(x,y);                              //function call

   int float
```

Now suppose the function return some integer value, you can use a variable to store that value.

i.e   z=add(x,y);

Parts of a function

```
main function
{
function prototype declaration
function call


}
function declaratory/definition
{
return statement
}
```

Function definition

2 parts

Syntax:

```
function_type    function_name(parameter list)
{
Local variable declaration;
Function body statement;
Return statement;
}
```

Function header

Note: no semicolon in header

Function body

Example:

```
int   add(int,int);                 //prototype
Z=add(x,y);                           //function call

int add(int a,int b)                 //function definition
{
body statement;
Return(a+b);
}
```

```
#include<iostream.h>
using namespace std;
int main()
{
 Print();
return 0;
}
void print()
{
cout<<“2 is even  no.”<<endl;
}
```

Error message since print was not declared in the scope.

```
#include<iostream.h>
using namespace std;
void print()
{
cout<<“2 is even  no.”<<endl;
}
int main()
{
Print();
return 0;
}
```

Why prototyping?????

```
#include<iostream.h>
using namespace std;
void print();
```

Return_type   function_name (parameter list/argument);

```
int main()
{
print();
```

function_name(parameter list/argument);

```
return 0;
}
```

```
void print()
```

function_type   function_name(parameter list)

```
{
cout<<"2 is even  no."<<endl;
}
```

Parts of a function

main function

{

function prototype declaration

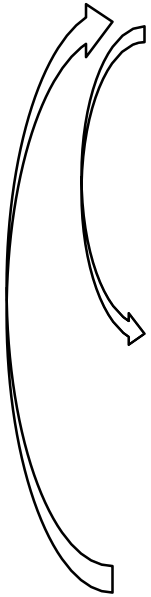
Return_type function_name(arguments); eg: int add(int);

function call

function_name(actual arguments); eg: add(a);

-------;

}



function declaratory/definition

Return_type function_name(formal arguments) eg: int add(int X);

{

------;

return statement

}

Function categories

i) Function with no return value and no argument.

```
void add(void);
```

ii) Function with arguments passed and no return value.

```
void add(int,int);
```

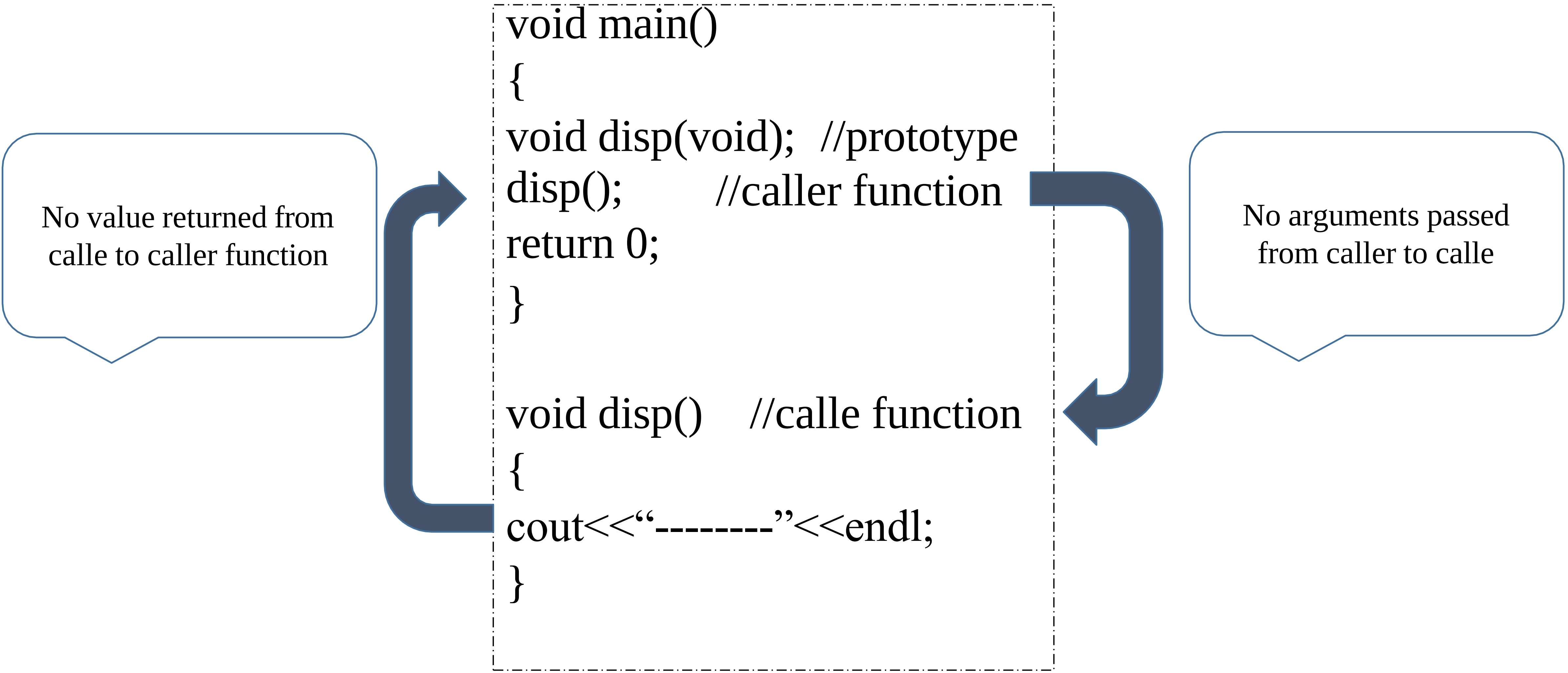
iii) Function with no arguments but returns a value.

```
int add(void);
```

iv) Function with arguments and returns a value.

```
int add(int,int);
```

I. Function with no return value and no argument



```
void main()
{
void disp(void);  //prototype
disp();        //caller function
return 0;
}

void disp()    //calle function
{
cout<<“--------”<<endl;
}
```

No value returned from calle to caller function

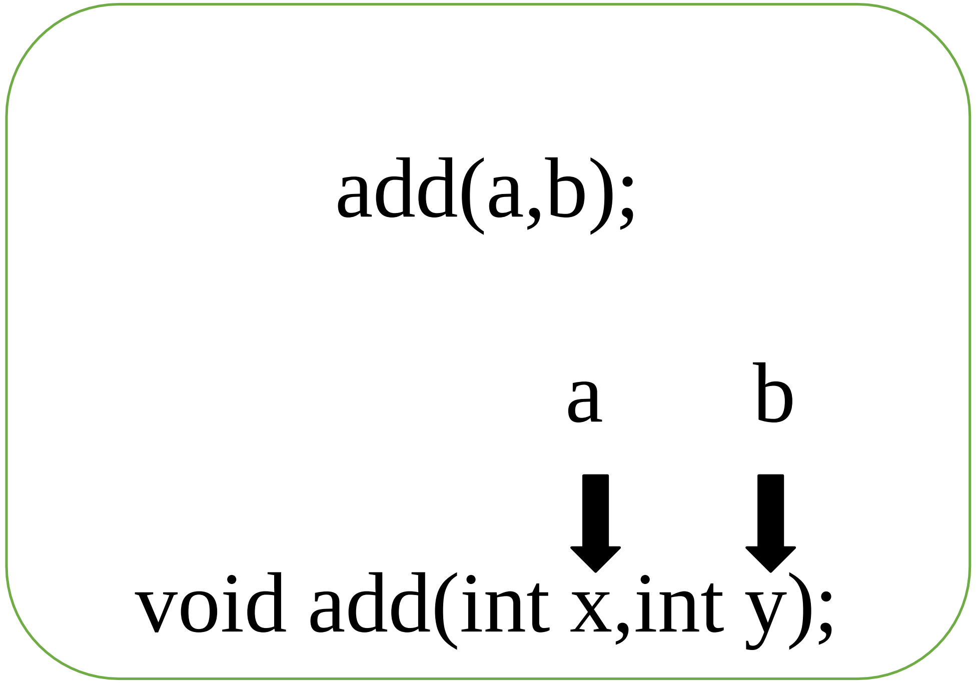
No arguments passed from caller to calle

```
//program to print square of a number using functions.
void main()
{
void sqr(void);
sqr();
getch();
return 0;
}

void sqr()
{
int no;
cout<<“enter a no.”;
cin>>no;
cout<<“square of”<<no<<“is”<<no*no;
}
```

ii. Function will not return any value but passes argument

```
#include<iostream.h>
#include<conio.h>
void add(int,int);
int main()
{
int a,b;
cout<<“enter values of a and b”<<endl;
cin>>a>>b;
add(a,b);
getch();
return 0;
}
void add(int x,int y)
{
int c;
c=x+y;
cout<<“addition is”<<c;
}
```

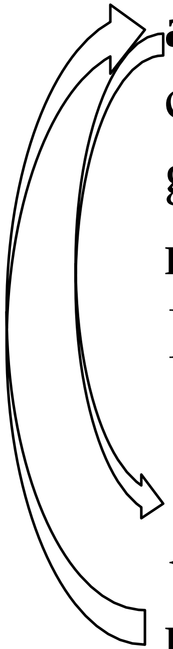


add(a,b);

a b

void add(int x,int y);

iii) Function with arguments and return value



```
main function
{
int sqr(int);                              //function prototype
int a,ans;
cout<<“enter a number”;
cin>>a;
ans=sqr(a);                                //function call
cout<<“square of number is”<<ans;
getch();
return 0;
}

 int sqr(int X)                            //function declaratory/definition
{
return(X*X);
}
```

iv) Function with no arguments but returns a value

```
int main()
{
int add(void);
int z;
z=add();
cout<<sum of 2 nos is”<<z;
getch();
return 0;
}

int add(void);
{
int a,b;
cout<<“enter 2 nos”;
cin>>a>>b;
return(a+b);
}
```

Function call

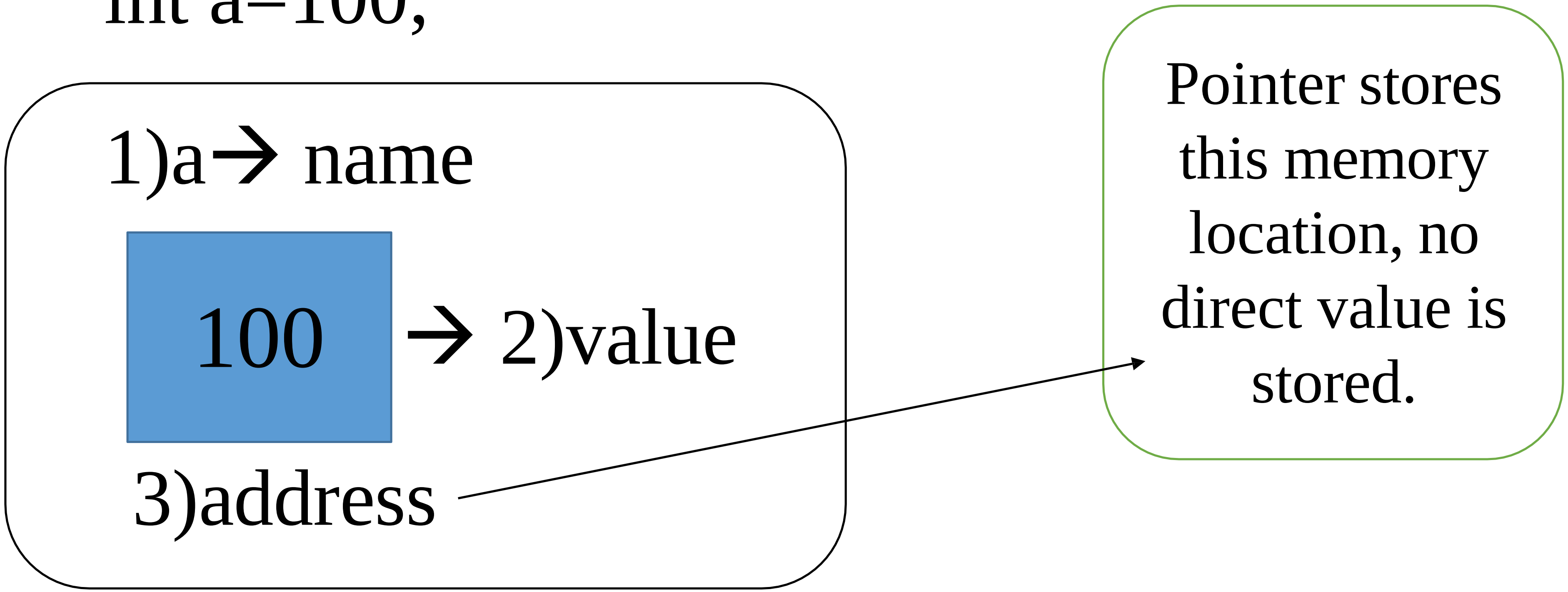
add(x,y);

i.e z=add(x,y);

Pointers

- Special type of variables which hold the address of another variable i.e no values or datas are stored but it points to another variable where data is stored.

int a=100;



Pointer stores this memory location, no direct value is stored.

Declaration: Data_type *variable_name;

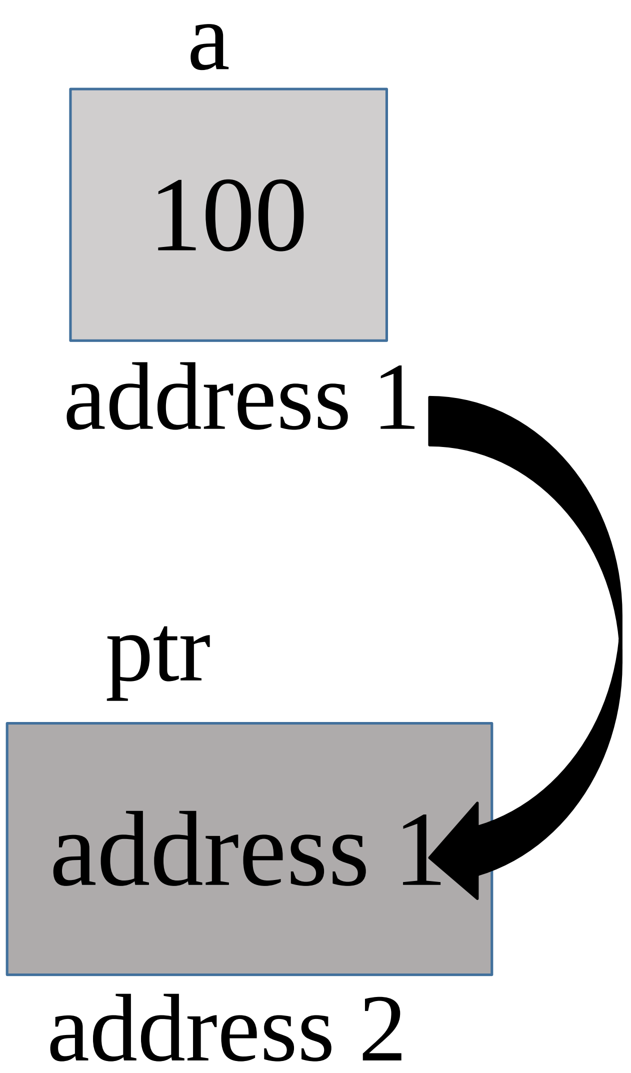
Intialisation:

```
int a=100;
int *ptr;
ptr=&a; //referencing
```

“Address of” operator

```
or
int a=100;
int *ptr=&a;
```

Pointer initialisation and declaration



a

100

address 1

ptr

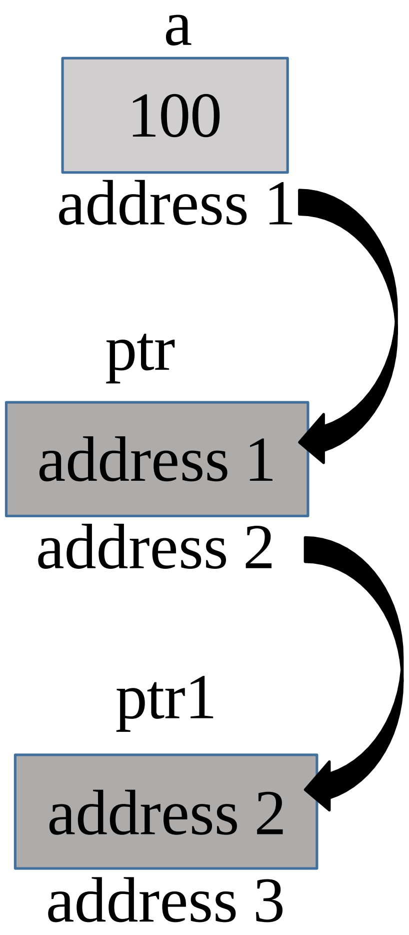
address 1

address 2

Note: a pointer can be used to point to another pointer also i.e it can store the address of another pointer.

```
int a=100;
int *ptr=&a;
int **ptr1=&ptr;
```

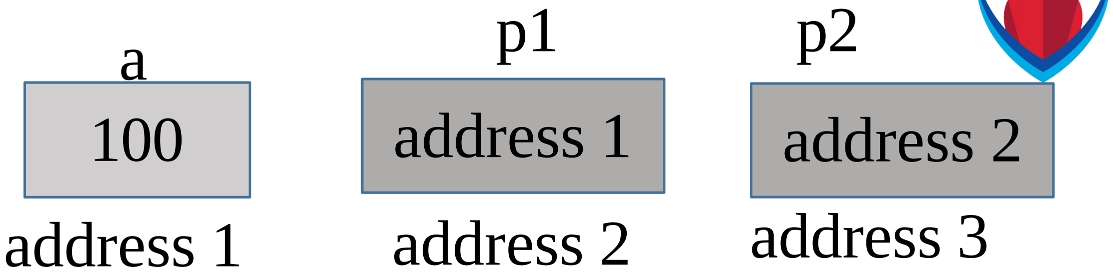
Note: pointer 1 points to address of ptr.



```
cout<<a; //100
cout<<*ptr;//100
cout<< **ptr1; //100
```

Chain of pointers

```
#include<iosream.h>
#include<conio.h>
int main()
{
int a=100;
int *p1;
int * *p2;
p1=&a;
p2=&p1;
cout<<“address of a”<<&a;
cout<<“address of a”<<p1;
cout<<“value of a”<< *p1;
cout<<“value of a”<< * *p2;
cout<<p2;
cout<< *p2;
getch();
}
```



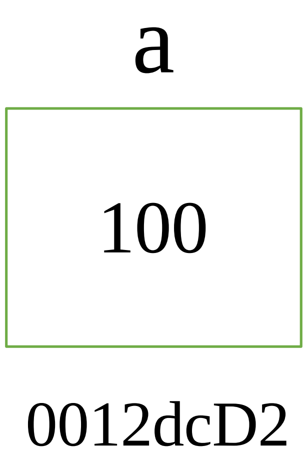
//p1 points to address of a

// p2 points to address of p1

Reference variable in C++

When a variable is declared as reference, it becomes an alternative name for an existing variable. A variable can be declared as reference by putting ‘&’ in the declaration.

```
int a=100;
```



a

100

0012dcD2

Now we will use a reference variable i.e two names for same memory location.

```
int a=100;
int &ref=a;          //initialization and declaration
```

Now in program we can use either “a” or an alternative name “ref”

```
C=a+b;          //same output
C=ref+b;
```

Program using reference variable

```
#include<iostream.h>
#include<conio.h>
int main()
{
int a=100;
int &ref=a;
cout<<“value of a is”<<a;
cout<<“value of ref is”<<ref;
cout<<address of a is”<<&a;
cout<<address of a is”<<&ref;
getch();
}
```

```
100
100
0012dcD2
0012dcD2
```

Both “a” and “ref” are used for same memory location as alternative name

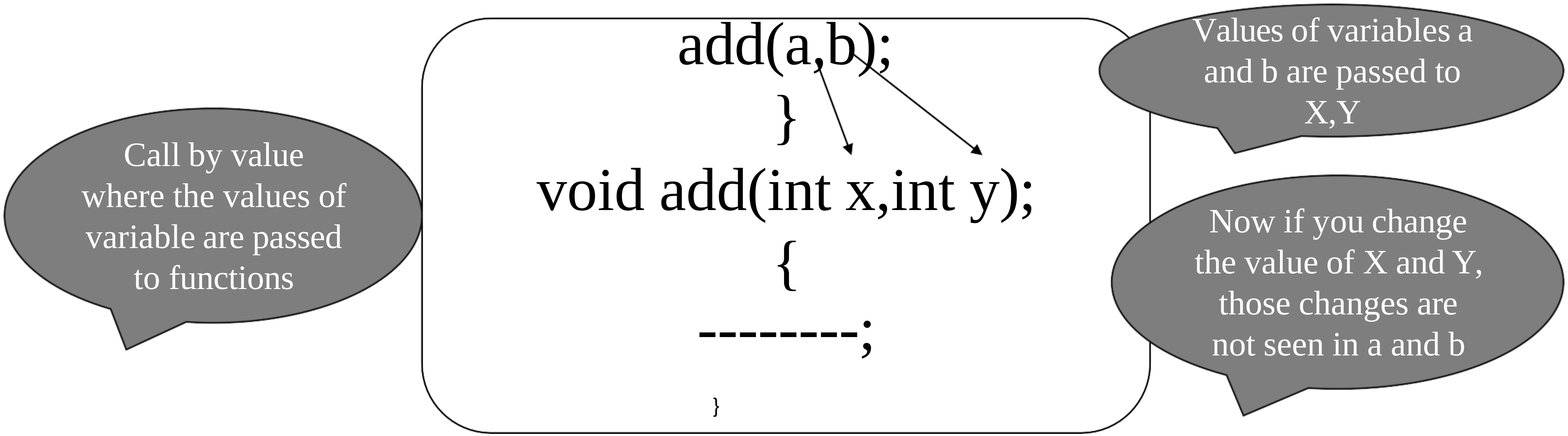
Call by value

A function can be invoked in two manners

(i)call by value

(ii)call by reference

The call by value method copies the value of actual parameters into formal parameters i.e the function creates its own copy of arguments and uses them.



Call by value where the values of variable are passed to functions

```
add(a,b);
}
void add(int x,int y);
{
--------;
}
```

Values of variables a and b are passed to X,Y

Now if you change the value of X and Y, those changes are not seen in a and b

```
/* program to illustrate the concept of call by value */
#include<iostream.h>
#include<conio.h>
void add(int,int);
int main()
{
int a,b;
cout<<“enter values of a and b”<<endl;
cin>>a>>b;
add(a,b);
getch();
return 0;
}
void add(int x,int y)
{
int c;
c=x+y;
cout<<“addition is”<<c;
```

Call by reference

In call by reference method in place of calling a value to the function being called , a reference to the original variable is passed .i.e the same variable value can be accessed by any of the two names.

```
add(a,b);
}

void add(int &x,int &y);
{
--------;
}
```

In function call
We write reference variable for formal arguments

No need for return statement

&X,&Y will be the reference variable for a and b. if we change X and Y, Value of a and b are changed accordingly

Program to illustrate call by reference

```
#include<iostream.h>
#include<conio.h>
void swap(int &,int &);
int main()
{
int a,b;
cout<<“enter the values of a and b”;
cin>>a>>b;
cout<<“before swaping”;
cout<<“A”<<a;
cout<<“B”<<b;
swap(a,b);
cout<<“after swaping”;
cout<<“A”<<a;
cout<<“B”<<b;
getch();
}

void swap(int &X,&Y)
{
int temp;
temp=X;
X=Y;
Y=X;
}
```

Return by reference

- A function can also return a reference.
- Example:

```
#include<iostream.h>
#include<conio.h>
int &max(int &x,int &y)
{
    if(x>y)
    return x;
    else
    return y;
}
int main()
{
    int m=1,n=2;
    max(m,n)=4;
    cout<<"Value of  m"<<m<<endl;
     cout<<"value of n"<<n<<endl;
     getch();
     return 0;
}
```

Subtopics-

- Inline functions
- Default arguments
- Function overloading

Inline Functions

- An inline function is a function that expanded in line when it is invoked.
- That is the compiler replaces the function call with the corresponding function code .
- **Syntax:**

```
inline function-header
        {
  Function body
          }
```

Example:

```
#include <iostream.h>
#include<conio.h>
int multiply(int);
int main( )
{
int x;
cout<< "\n Enter the Input Value: ";
cin>>x;
cout<<"\n The Output is: " << multiply(x);
getch();
}
inline int multiply(int x1)
{
return 5*x1;
}
```

Default Arguments

- Default values are specified when the function is declared.
- Compier looks at the prototype to see how many arguments function uses.
- Default arguments are useful in situations where some arguments always have the same value.

Example

```
#include<iostream.h>
#include<conio.h>  int main()
{
    float amount;
    float value(float p,int n,float r=0.15); //prototype void
    printline(char ch='*',int len=40); //prototype
    printline();                    //uses default values for argumennts amount =
    value(5000.00,5);         //default for 3rd argument
```

```
cout<<"\n Final value = "<<amount<<"\n\n";
  println('='); //use default value for 2nd argument
  return 0;
  }
  float value(float p, int n, float r)
  {
      int year =1;
       float sum = p;
      while(year <= n)

{
            sum = sum*(1+r);
            year = year+1;
       }
```

```
getch();
     return(sum);
       }
  void printline(char ch, int len)
  {
     for(int i=1;i<=len;i++)
      printf("%ch",ch);
      printf("\n");
      }
```

Function Overloading

- A function is overloaded when same name is given to different function.
- The two functions with the same name will differ at least in one of the following.
  a) The number of parameters
  b) The data type of parameters
  c) The order of appearance

Example

```
#include <iostream.h>
#include<conio.h>
class arithmetic {
public:
    void calc(int num1)
{
cout<<"Square of a given number: "
<<num1*num1 <<endl;
}
 void calc(int num1, int num2 )
     {
cout<<"Product of two whole numbers: "
    <<num1*num2 <<endl;
}
};
int main() //begin of main function
{
      arithmetic a;
      a.calc(4);
      a.calc(6,7
      ); getch();
}
```

Thank You