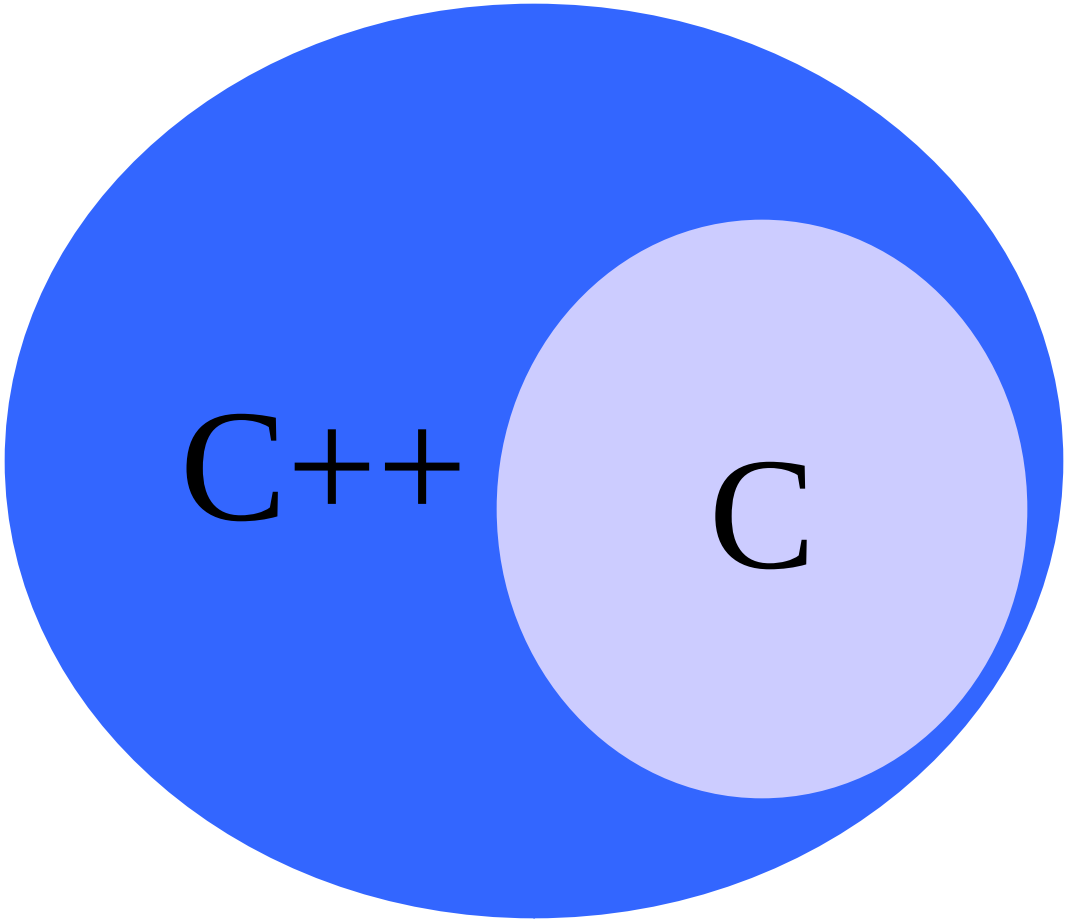
Object Oriented Programming through

C++

Introduction

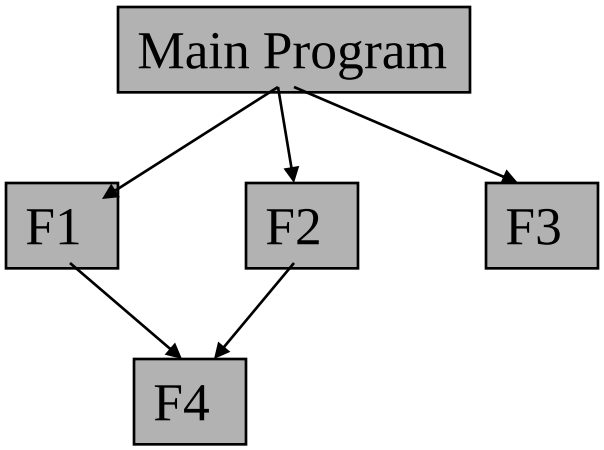
- C++ is an Object Oriented Programming language, developed by Bjarne Stroustrup in early 80’s at AT & T Lab.
- It is better than C
- More appropriate for real-life and commercial applications.

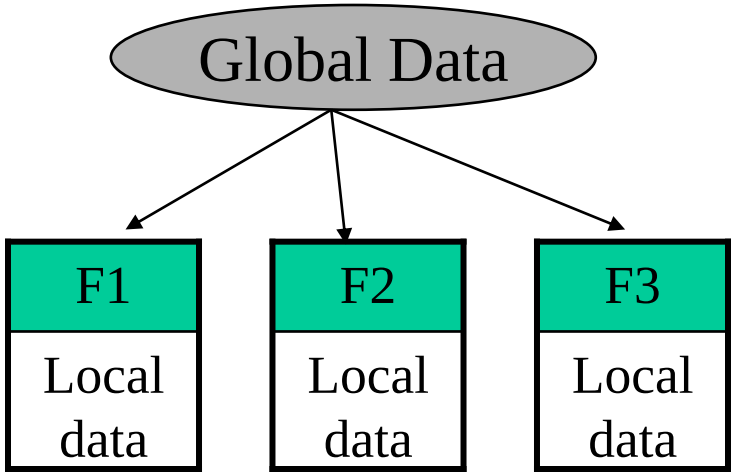
Venn diagram



Procedure Oriented Programming (POP)

- Emphasis is on algorithm
- Large programs are divided into a no of procedures or functions.

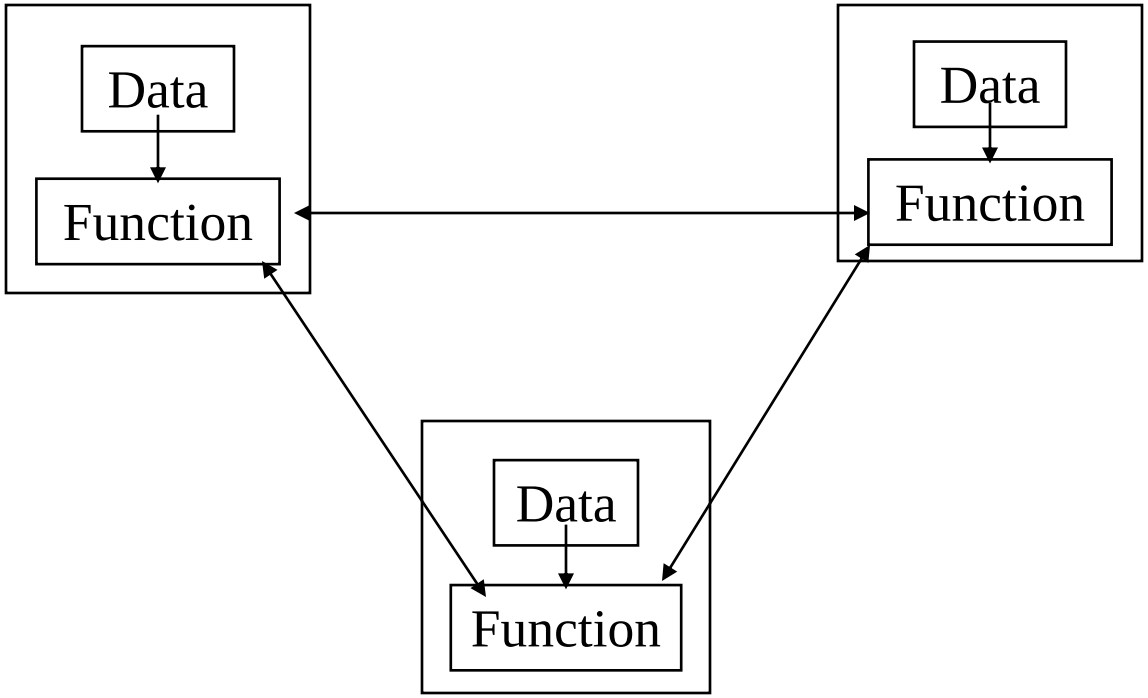




- Mainly the data in the programs are global.
- Data move freely inside the system.
- It follows ***top-down*** approach.

Object Oriented Programming (POP)

- Emphasis is on data
- Large programs are divided into a no objects.



- Data are tied with functions.
- It follows ***bottom-up*** approach.

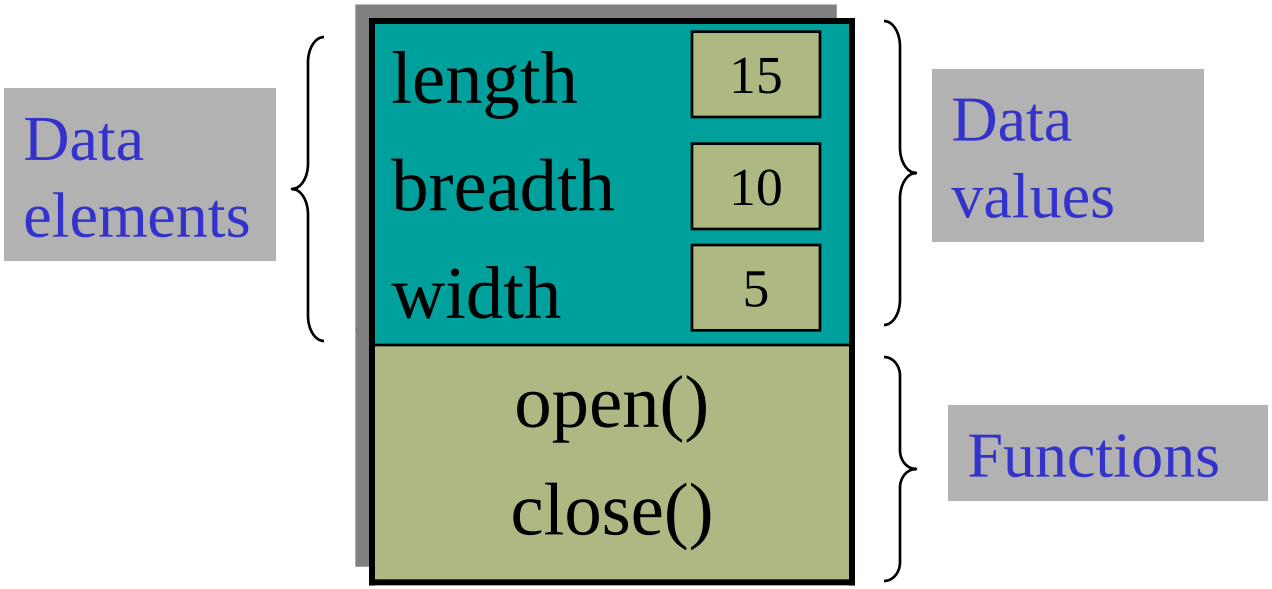
Object

- An object is a collection of some properties, behavior with some existence.
- Box object:

  Properties:- *length* , *breadth*, *width*. (data elements)

  Behavior:- *open*, *close*. (functions)

  Existence:- *length=15*, *breadth=10*, *width=5* (data values)



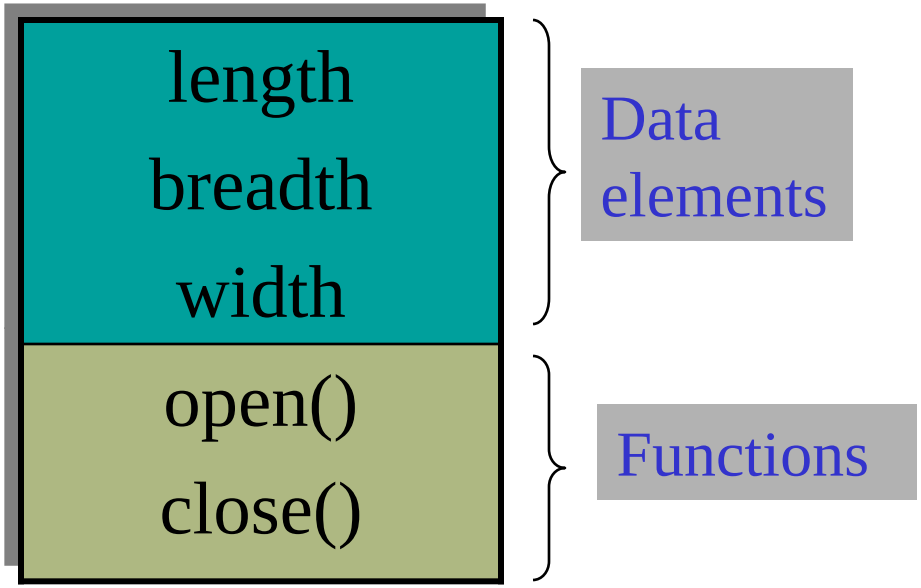
A Box object

Class

- Class is a collection of some properties, behavior
- BOX class:

  Properties:- *length* , *breadth*, *width*. (data elements)

  Behavior:- *open*, *close*. (functions)



A Box class

- Class is a logical structure or prototype where as Object has physical existence.
- Object is an instance of a Class.
- Class is a collection of similar types of objects where the data values may not be the same.

Definition of OOP

OOP is an approach which provides a way to make modularized program by allocating separate memory locations for both data and functions which acts as prototype for creating some more object as per demand.

Insertion and Extraction operators

```
cout << "NITR";
```

- **cout** is an object of class ostream.
- << is called the insertion or put-to operator.
- The value present right to << operator would be put into the object cout which is connected to VDU to display it.

```
int a;

cin >> a;
```

- **cin** is an object of class istream.
- >> is called the extraction or get-from operator.
- Extract the value stored in cin object got from keyboard and assign it to the integer variable a;

A simple program

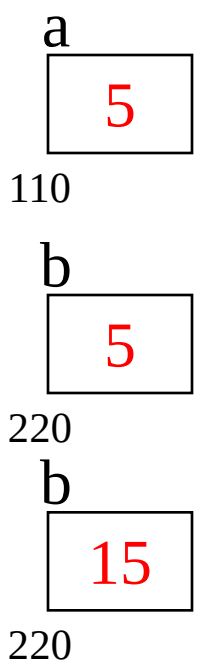
```
#include<iostream>
using namespace std;            // namespace defines a scope for
int main(){                     // global identifiers.
    int a,b,sum;
    cout<<"Enter the value of a and b \n";
    cin>>a;
    cin>>b;                     // cin>>a>>b;
    sum=a+b;
    cout<<"The addition result is: "<<sum;
    return 0;
}
```

Executing the program in linux environment

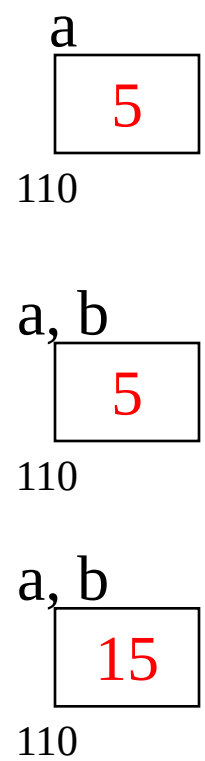
- Write the program in VI or VIM editor.
- Save the file with .cpp extension.
- Compile the file in the $ prompt with the command ***g++***. e.g. g++ add.cpp
- After the successful compilation, to see the output by ***./a.out***

Reference Variable

```
int a=5;

int b=a;

b=b+10;
cout<<a;  // 5
cout<<b;  // 15
```




```
int a=5;

int &b=a;

b=b+10;
cout<<a;  // 15
cout<<b;  // 15
```



‘b’ acts as an alias to ‘a’ and points to the same memory location 110

Scope resolution operator

```
#include<iostream.h>
using namespace std;
int m=10;                          // global variable m.
int main(){
        int m=20;                  // local m in outer block
                {
                        int k=m;           // local m in inner block
                        int m=30;
                        cout<<k;           // 20
                        cout<<m;           // 30
                        cout<< ::m;        // 10
                }
        cout<<m;                           // 20
        cout<< ::m;                        // 10
        return 0;
}
                    // ::m directly accesses the global m
```

Memory Management Operators

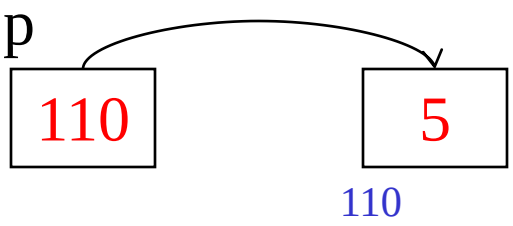
1. new operator: It allocates memory

Syntax : new datatype

e.g. **new int** allocates a memory block of size 2 bytes and returns the base address of the memory block;

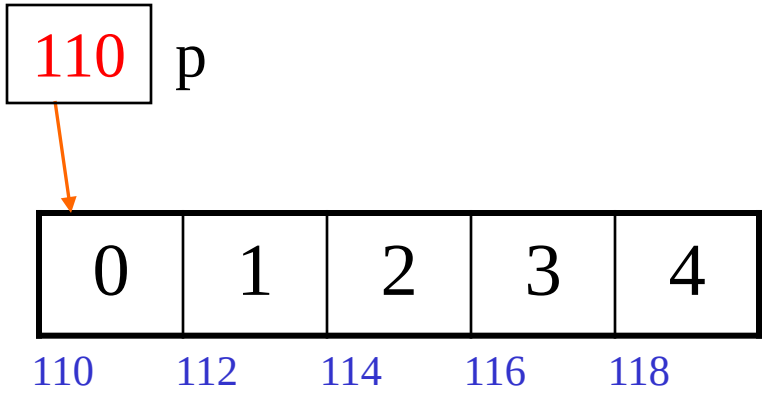
use:

```
int *p;
p=new int;
*p=5;
```



For an array:

```
int *p;
p=new int[5];
*p=0;
*(p+1)=1;
*(p+2)=2;
*(p+3)=3;
*(p+4)=4;
```



2. delete operator: It deallocates memory

use:

```
int *p;
p=new int;
*p=5;
delete p;
```

For an array:

```
int *p;
p=new int[5];
delete [5]p;
```

or

```
delete []p;
```

or

```
delete p;
```

Advantages of *new and delete* operators over *malloc, calloc and free*

- Don't have to specify the *sizeof* operator while using new.
- new operator will return correct address so there is no need of type casting.
- While allocating memory dynamically we can initialize the block.
- new and delete operators can be overloaded.

```
int *p=(int *)malloc(sizeof(int));
```

```
int *p=new int(5);
```

Passing Arguments to Function

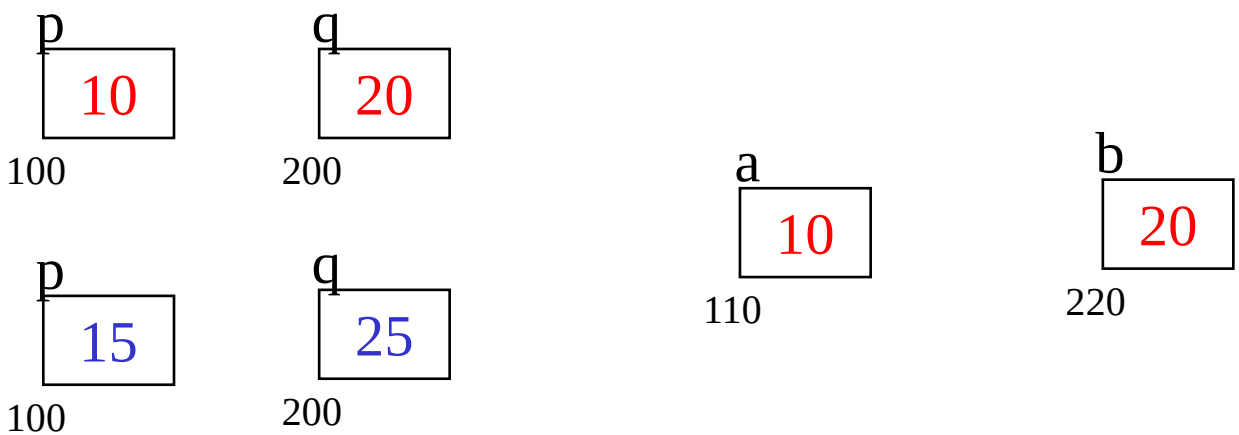
Arguments can be passed to a function in 3 ways.

- Pass by Value.
- Pass by Address or Pointer.
- Pass by reference.

Pass by Value

```
void add(int p, int q){
        cout<<(p+q);     // 30
        p=p+5;
        q=q+5;
}
```

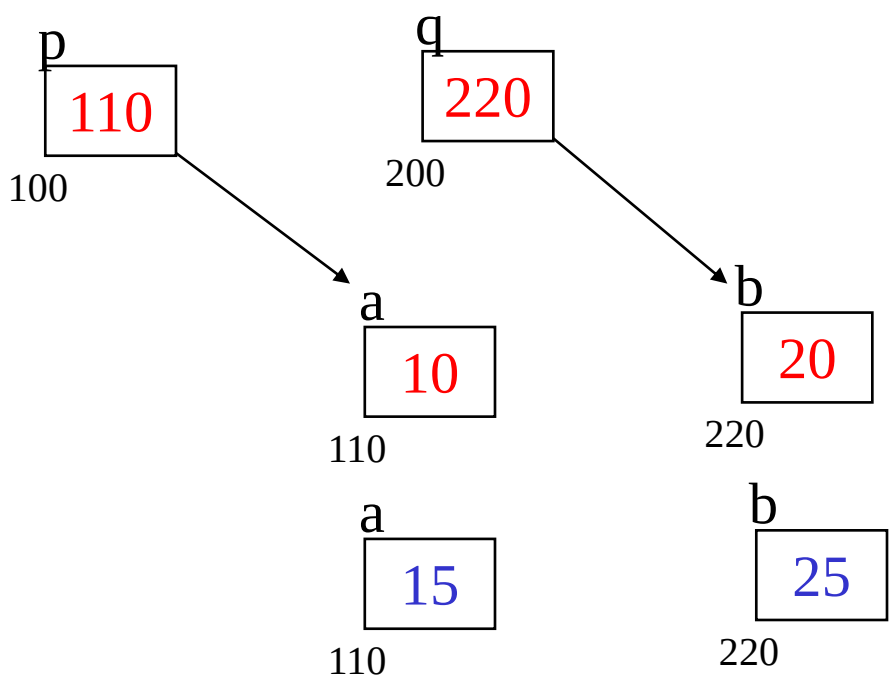
```
int main(){
        int a=10,b=20;
        add(a,b);
        cout<<a;   // 10
        cout<<b;   // 20
}
```



Pass by Address or Pointer

```
void add(int *p, int *q){
        cout<<(*p+*q);   // 30
        *p=*p+5;
        *q=*q+5;
}
```

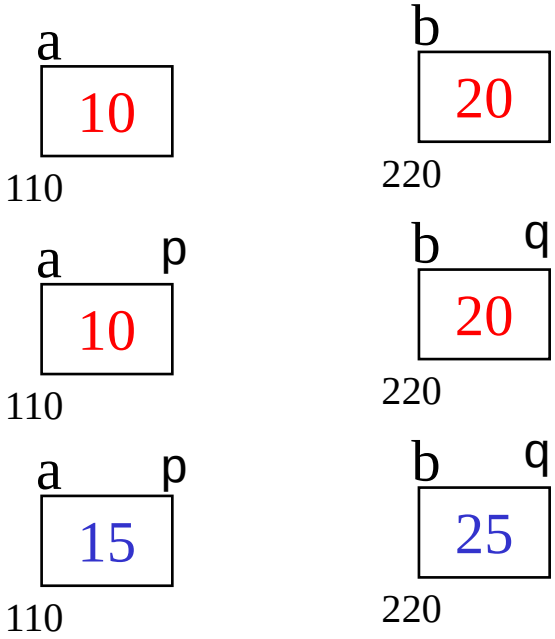
```
int main(){
        int a=10,b=20;
        add(&a,&b);
        cout<<a;   // 15
        cout<<b;   // 25
        return 0;
}
```



Pass by Reference

```
void add(int &p, int &q){
        cout<<(p+q);      // 30
        p=p+5;
        q=q+5;
}
```

```
int main(){
        int a=10,b=20;
        add(a,b);
        cout<<a;   // 15
        cout<<b;   // 25
        return 0;
}
```



Returning values from Function

Values can be returned from a function in 3 ways.

- Return by Value.
- Return by Address or Pointer.
- Return by reference.

Return by Value

```
int  max(int p, int q){
        if(p > q)
                return p;
        else
                return q;
}
```

```
int main(){
        int a=50,b=20,c;
        c=max(a,b);
        cout<<c;   // 50
}
```

Return by Address or Pointer

```
int *  max(int *p, int *q){
        if(*p > *q)
                return p;
        else
                return q;
}
```

```
int main(){
        int a=50,b=20;
        int *c;
        c=max(&a,&b);
        cout<< *c; // 50
}
```

Return by Reference

```
int &  max(int p, int q){
        if(p > q)
                return p;
        else
                return q;
}
```

```
int main(){
        int a=50,b=20,c;
        c=max(a,b);
        cout<< c;  // 50
}
```

Default Argument in Function

```
void add(int a,int b){
        cout<<a+b;
}
```

```
int main(){
        add(5,7);   // 12
        add(2);     // error-insufficient no of arg
}
```

```
void add(int a,int b=9){
        cout<<a+b;
}
```

```
int main(){
        add(5,7);   // 12
        add(2);     // 11
}
```

Default argument

- always assigned from right to left of the arg list
- Default value is overridden if new value is passed

Inline function

- If the overhead time to call a function is greater than the execution time of the function code then the function should be made as *inline* and the function calls are replaced by the function code.

```
int main(){
        void disp(int);
        // prototype of disp()
        disp(5);
        disp(10);
        disp(15);
        return 0;
}
```

```
void disp(int a){
        cout<<a;
}
```

```
int main(){
        void disp(int);
        // no inline keyword in the prototype
        disp(5); //replaced by cout<<5;
        disp(10); //replaced by cout<<10;
        disp(15); //replaced by cout<<15;
        return 0;
}
```

```
inline void disp(int a){
        cout<<a;
}
```

- The inline keyword precedes the function definition but not the function declaration.
- Inline function acts as macro.
- Inline is a request but macro is a command.
- Incase the function contains more statements and can’t be expanded inline then the compiler shows a warning message and the function is treated as a normal c++ function.
- Inline function does not work in the following situations.
  - If the function contains a loop, switch or goto statement.
  - If it is a recursive function.
  - If it contains static variable.

Function Overloading

```
void test() { ……..}
void test(int a) {…….}
void test(char b) {…….}
void test(int a,char b) {…….}
void test(char b,int a) {…….}
```

```
test(); //calls the 1st test
test(5); //calls the 2nd test
test('m'); //calls the 3rd test
test(5,'m'); //calls the 4th test
test('m',5); //calls the 5th test
```

- Function overloading is a mechanism of using the same function name to create multiple functions performing different tasks and those can be differentiated by the compiler at the compile time(compile time polymorphism).
- It provides static binding.
- The functions are differentiated by the no of arguments, type of arguments or the sequence of arguments.

```
void test(char a) { ……..}
void test(long b) {…….}
```

```
test('A'); //calls the 1st test
test(25L); //calls the 2nd test
test(10); //calls the 3rd test
```

The argument (10) which is of int type is implicitly converted to long and so calls the 3rd test

The compiler chooses a particular overloaded from the set as follows.

- Search for *exact match*.
- Follow *integral promotion* *(int to char, char to int, float to double, double to float)*
- Follow *implicit type conversion*.

```
void test(double a) { ……..}

void test(long double b) {…….}
```

```
test(5.5f);

/* Shows an error message because float can be converted to both double and long double */
```

```
void test() { ……..}

int test() {…….}
```

→ No function overloading

OOP Features

1. Abstraction: It is a process of highlighting the important features of a class or an object without going into much detail.

2. Encapsulation: It is a process of putting data and functions into a single unit.

3. Polymorphism: means *"same name multiple forms"* e.g. **area** of a triangle.

   area of a rectangle.

4. Inheritance: The process of acquiring the properties of one class by another class is called inheritance.

(Thief + Intelligent) (Thief + Lazy)

OOP Features Contd…

5. Dynamic Binding:

Binding is a process of associating the function call with the code to be executed.

binding → Static binding (Compile time)
binding → Dynamic binding (Run time)

6. Message Passing: Message to an object is a request to execute a piece of code.

e.g. ob.area(15,10,5);

where ob is an object of class BOX.

Advantages of OOP

- Inheritance facilitates reusability.
- The principle of data hiding helps the programmer to build secure programs.
- It is easy to partition the project work based on objects.
- Object oriented systems can be easily upgraded from small to large systems.

Assignments

1. WAP in C++ to print all the prime no.s between 1 to 1000.
2. WAP in C++ to calculate the area of a square.
3. WAP in C++ to calculate compound interest.
4. WAP in C++ to accept name, mark, rollno, of 10 students and print average mark of the students using array of structure.
5. WAP in C++ to sort an array in ascending and descending order.
6. WAP in C++ to find the gross salary of 10 employees, given their BASIC, DA and HRA. The default HRA for each employee is 15% of basic and those who have basic more than 15,000 will get HRA of 25%.
7. WAP in C++ to implement swap_n_max() function to swap 2 nos, where the arguments are passed by pointer and the function returns a pointer to the maximum of the 2 nos.
8. WAP in C++ to implement swap_n_max() function to swap 2 nos, where the arguments are passed by reference and the function returns a reference to the maximum of the 2 nos.
9. Overload the area() function to calculate the area of a rectangle and a circle.