Inline functions

- **C++ provides a keyword *inline* that makes the function as inline function.**
- **Inline functions get replaced by compiler at its call statement. It ensures faster execution of function just like macros.**
- **Advantage of inline functions over macros: inline functions are type-safe.**
- **Inline is a request made to compiler.**
- **Every function may not be replace by complier , rather it avoids replacement in certain cases like function containing switch , loop or recursion may not be replaced**

Demo of Inline functions

```
#include<stdio.h>
#define sq(a) a*a
inline int square(int a);
int main()
{
    int x=5, y=0;
    y=sq(x); // y=x*x;
    printf("\n x=%d y=%d", x, y);

    y=square(x);
    printf("\n x=%d y=%d", x, y);
}
inline int square(int a)
{
    return a*a;
}
// for compile ->>  g++ -E -o demo5.i demo5.cpp
```

Function overloading :

Function having same name but differs either in different number of arguments or type of arguments or order of arguments such process of writing function is called function overloading .

Functions which is taking part in function overloading such functions are called as overloaded functions.

case 1:function having same name & diff no of arguments

```
#include<stdio.h>
int sum(int n1, int n2) // sum@@int, int
{
    return n1+n2;
}
int sum(int n1, int n2, int n3) // sum@@int, int, int
{
    return n1+n2+n3;
}
int main()
{
    int result;
    result=sum(10,20);  // sum@@int, int
    printf("result=%d", result);  // 30
    result=sum(10,20, 30);  // sum@@int, int, int
    printf("result=%d", result); // 60
}
```

case 2:function having same name & diff types of arguments

```
#include<stdio.h>
int sum(int n1, int n2) // sum@@int, int
{
    return n1+n2;
}
float sum(float n1, float n2) // sum@@float,float
{
    return n1+n2;
}
int main()
{
    int result;
    float result1;
    result=sum(10,20);  // sum@@int, int
    printf("result=%d", result);  // 30
    result1=sum(10.2f,20.1f);  // sum@@float, float
    printf("result=%.2f", result1); // 30.3
}
```

case 3:function having same name & diff order of arguments

```
#include<stdio.h>
float sum(int n1, float n2) // sum@@int, float
{
    return n1+n2;
}
float sum(float n1, int n2) // sum@@float,int
{
    return n1+n2;
}
int main()
{
    float result;
    result=sum(10.1f,20);  // sum@@int, float
    printf("result=%.2f", result);  // 30.1
    result=sum(10,20.2f);  // sum@@int, float
    printf("result=%.2f", result); // 30.2
    return 0;
}
```

Name mangling:

when we write function in c++ complier internally creates unique name for each and every function by looking towards name of the function and type of arguments pass to that function. Such process of creating unique name is called name mangling . That individual name is called mangled name.

eg:

```
        sum (int no1, int no2, it no3)
                                   sum@ int, int , int
        sum(int no1, int no2)
                                          sum@int,int
```

#include<stdio.h>

```
void f1(int a){
       printf("\n Inside int block");
}
void f1(float a){
      printf("\n Inside float block");
}
void f1(double a){
      printf("\n Inside double block");
}
int main()
{
        f1(1);
        f1(1.2);
        f1(1.2f);
        f1((int)1.2);
        f1(1.2);
        return 0;
}
```

Namespace

To prevent name conflicts/ collision / ambiguity in large projects
• to group/orgaize functionally equivalent / related types toghther.
• If we want to access value of global variable then we should use scope resolution operator (::)
• We can not instantiate namespace.
• It is designed to avoid name ambiguity and grouping related types.
• If we want to define namespace then we should use namespace keyword.
• We can not define namespace inside function/class.
• If name of the namespaces are same then name of members must be different.
• We can not define main function inside namespace.
• Namespace can contain:
1. Variable
2. Function
3. Types[structure/union/class]
4. Enum
5. Nested Namespace

```c
#include<stdio.h>
int no1=1000;
int main(void)
{
    int no1=100;
    printf("\n local variable no1=%d [%u]", no1, &no1);
                                            // no1=100

    return 0;
}
```

in c prog it will print local variable if name of local variable and global variable is same.

```
#include<stdio.h>
int no1=1000;
int main(void)
{
  int no1=100;
  printf("\nlocal variable no1=%d [%u]", no1, &no1); // no1=100
  printf("\nglobal variable::no1=%d [%u]",::no1,&::no1);//no1=1000
  return 0;
}
```

in cpp prog we can print both local and global variable.
We have to use scope resolution operator (::) to print global variable.

```
#include<stdio.h>
```

```
namespace N1
{
    int no1=50;
    int no2=60;
}
int no1=1000;
int main(void)
{
   int no1=100;
   printf("\n local variable no1=%d [%u]", no1, &no1); // no1=100
   printf("\n global variable::no1=%d [%u]",::no1,&::no1);
   // namespacename::objectname                       //no1=1000
   printf("\n  N1::no1=%d [%u]", N1::no1, &N1::no1); // no1=50
   printf("\n  N1::no2=%d [%u]", N1::no2, &N1::no2); // no2=60
   using namespace N1;
   printf("\n no2=%d [%u]", no2, &no2); // no2=60
   printf("\n local variable  no1=%d [%u]", no1, &no1); // no1=100
   printf("\n N1::no1=%d [%u]", N1::no1, &N1::no1); // no1=50
   return 0;
}
```

#include<stdio.h>

```
namespace N1
{
    int no1=50;
    int no2=60;

    namespace N2
    {
        int no1=250;
        int no3=300;
    }
}
int no1=1000;
int main(void)
{
 int no1=100;
 printf("\nlocal variable no1=%d [%u]", no1, &no1); // no1=100
 printf("\nglobal variable::no1=%d [%u]",::no1,&::no1);//no1=1000
```

```
  // namespacename::objectname
  printf("\n  N1::no1=%d [%u]", N1::no1, &N1::no1); // no1=50
  printf("\n  N1::no2=%d [%u]", N1::no2, &N1::no2); // no2=60

  // outernamespacename::innernamespacename::objectname

 printf("\nN1::N2::no1=%d[%u]",N1::N2::no1,&N1::N2::no1);//no1=250
 printf("\nN1::N2::no3=%d[%u]",N1::N2::no3,&N1::N2::no3);//no3=300

  using namespace N1;
  //using namespace N2;  // or
  using namespace N1::N2;
  printf("\n variable from N1 no2=%d [%u]", no2, &no2); // no2=60
  printf("\n local variable  no1=%d [%u]", no1, &no1); // no1=100
  printf("\n N1::no1=%d [%u]", N1::no1, &N1::no1); // no1=50
  printf("\n N1:N3 no3=%d [%u]", no3, &no3); // no3=300
  printf("\n local variable no1=%d [%u]", no1, &no1); // no1=100

  return 0;
}
```

this pointer

- **When we call member function by using object implicitly one argument is passed to that function such argument is called this pointer**
- ***this* is a keyword in C++.**
- **this pointer always stores address of current object or calling object.**
- **Thus every member function of the class receives *this* pointer which is first argument of that function.**
- **This pointer is constant pointer.**

For Cmplex class member function type of this pointer is

Complex * const this

classname * const this

Structure in cpp with out this pointer

```
#include<stdio.h>
#pragma pack(1) // slack bytes
struct student
{
    private:  // variable // data member  // field
        int rollno;
        char name[10];
        float per;

    public:
       void accept_stud_info()
       {
          printf("\n enter rollno::");
          scanf("%d", &rollno);
          printf("\n enter name::");
          scanf("%s", name);
          printf("\n enter per::");
          scanf("%f", &per);
          return;
       }
```

Structure in cpp with out this pointer

```
    void display_stud_info()
    {
        printf("\n rollno  name  per \n");
        printf("%-5d%-10s%6.2f", rollno, name, per);
        printf("\n\n\n");
       return;
    }
};
int main(void)
{
    student s1;//struct student s1;
    printf("\n enter student info::");
    s1.accept_stud_info();  //accept_stud_info(&s1);
    //s1.per=45;
    printf("student info :: \n");
    s1.display_stud_info(); //display_stud_info(&s1);
    return 0;
}
```

Structure in cpp with this pointer

```
#include<stdio.h>
#pragma pack(1) // slack bytes
struct student
{
    private:  // variable // data member  // field
        int rollno;
        char name[10];
        float per;
    public:
      //void accept_stud_info(structname * const this)
      //void accept_stud_info(student *const this)
      void accept_stud_info()
      {
         printf("\n enter rollno::");
         scanf("%d", &this->rollno);
         printf("\n enter name::");
         scanf("%s", this->name);
         printf("\n enter per::");
         scanf("%f", &this->per);
         return;
      }
```

Structure in cpp with this pointer

```
 //void display_stud_info(student * const this)
   void display_stud_info()
   {
      printf("\n rollno  name  per \n");
      printf("%-5d%-10s%6.2f",this->rollno,this->name,this->per);
      return;
    }
};
int main(void)
{
    student s1;//struct student s1;
    printf("\n enter student info::");
    s1.accept_stud_info();  //accept_stud_info(&s1);
    //s1.per=45;  //error as per is private

    printf("student info :: \n");
    s1.display_stud_info(); //display_stud_info(&s1);
    return 0;
}
```

What is class?

- **Building blocks that binds together data & code**
- **Program is divided into different classes**
- **Class has**
  - **Variables (data members)**
  - **Functions(member functions or methods)**

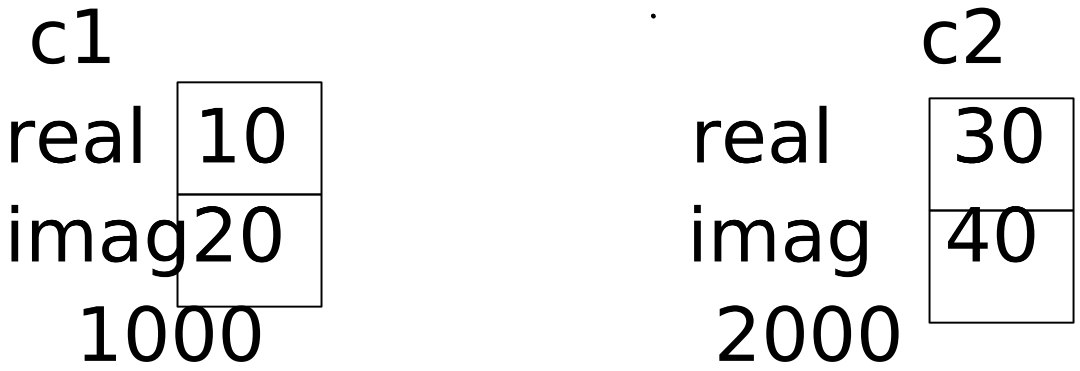
What is object?

- Object is an instance of class
- Entity that has physical existence, can store data, send and receive message to communicate with other objects
- Object has
  - Data members (***state*** of object)
  - Member function (***behavior*** of object)
  - Unique address(***identity*** of object)

What is object?

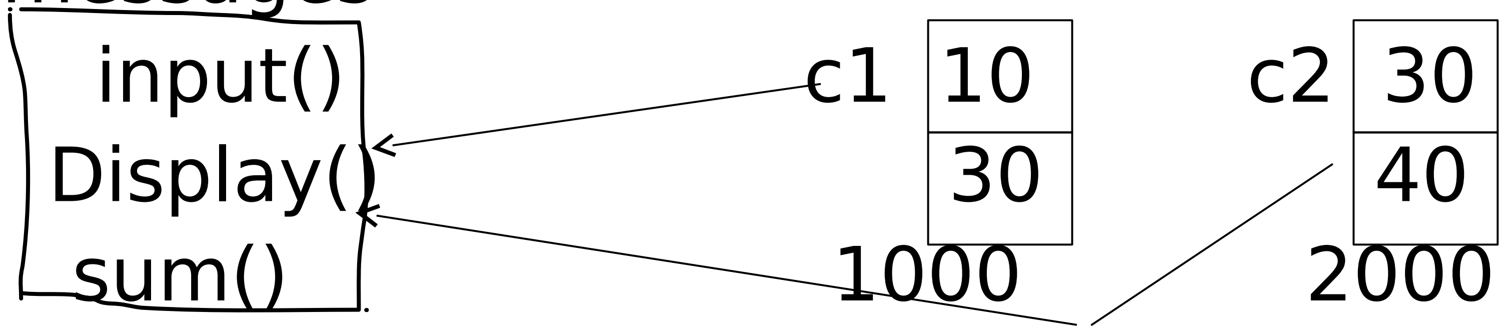
- The values stored in data members of the object called as ‘state’ of object.

Data members of class represent state of object.   Complex c1, c2;



- Behavior is how object acts & reacts, when its state is changed & operations are done
- Behavior is decided by the member functions.

Operations performed are also known as messages



Identity : Every object has a characteristics that makes object unique. Every object has unique address.

- *Identity of c1 1000 & c2 is 2000*

Class

it is template or blue print for an object.

object is always created by looking towards class, that’s why it is template for class.

class is a logical entity.

memory is not allocated to that class.

class is collection of data members and member functions.

Object

it is an instance of class.

object is physical entity.

memory is always allocated to object.

Data members

- **Data members of the class are generally made as private to provide the data security.**
- **The private members cannot be accessed outside the class.**
- **So these members are always accessed by the member functions.**

Access specifiers

- **By default all members in structure are accessible everywhere in the program by dot(.) or arrow(→) operators.**

- **But such access can be restricted by applying access specifiers**

  - **private: Accessible only within the struct**
  - **public: Accessible within & outside struct**

class in cpp with this pointer

```
#include<stdio.h>
#pragma pack(1) // slack bytes
class student
{
    private:  // variable // data member  // field
        int rollno;
        char name[10];
        float per;
    public:
       //void accept_stud_info(structname * const this
       //void accept_stud_info(student *const this)
       void accept_stud_info()
       {
          printf("\n enter rollno::");
          scanf("%d", &this->rollno);
          printf("\n enter name::");
          scanf("%s", this->name);
          printf("\n enter per::");
          scanf("%f", &this->per);
          return;
       }
```

Struct in cpp with this pointer

```
 //void display_stud_info(student * const this)
  void display_stud_info()
  {
     printf("\n rollno  name  per \n");
     printf("%-5d%-10s%6.2f",this->rollno,this->name,this->per);
     return;
  }
};
int main(void)
{
    student s1;//class student s1;
    printf("\n enter student info::");
    s1.accept_stud_info();

  //s1.per=45;  //error as per is private
    printf("student info :: \n");
    s1.display_stud_info();
    return 0;
}
```

cin and cout

- C++ provides an easier way for input and output.
- The output:
  - cout << “Hello C++”;
- The input:
  - cin >> var;

```
#include<stdio.h>
int main()
{
    printf(“ hellow world”);
}
```

```
#include<iostream.h>
int main()
{
    cout<<“hellow world”;
}
```

printf is a function & declared in stdio.h header file. So if we want to use printf it is necessary include stdio.h header file
cout is object of ostream class and ostream class is declared in iostream.h that’s why if u want to use cout object is necessary to include iostream header file

operator which is used with cout is called as insertion operator <<

```
int res=30;
printf(“res=%d”, res);
```

```
int res=30;
cout<<“res=”<<res;
```

```
int  a=10,b=5;
printf("a=%d b=%d",a,b);
```

```
int a=10, b=5;
cout<<"a="<<a<<" b="<<b;
```

```
#include<stdio.h>
int main()
{        int num;
          scanf("%d",&num);
}
```

```
#include<iostream.h>
int main()
{        int num;
          cin>>num;
}
```

scanf is a function & declared in stdio.h header file. So if we want to use scanf it is necessary include stdio.h header file

cin is object of istream class and istream class is declared in iostream.hthat's why if u want to use cin object is necessary to include iostream header file

operator which is used with cin is called as operator extraction (>>)

```
int  no1, no2;
scanf("%d%d",&no1,
&no2);
```

```
int no1, no2;
 cin>>no1>> no2;
```

```
//#include<string.h>//#include<cstring>
#include<iostream>
int main()
{
    int no1, no2;
    std::cout<<"Enter No1::";
    //std::cin>>&no1; error  // scanf("%d", &no1);
    std::cin>>no1; // correct
    std::cout<<"Enter No2::";
    std::cin>>no2; // correct

    std::cout<<"\tno1="<<no1<<"\t &no1="<<&no1<<std::endl;
    std::cout<<"\tno2="<<no2<<"\t &no2="<<&no2<<"\n";

}
```

```
#include<iostream>
using namespace std;
int main()
{
    int no1, no2;
    cout<<"Enter No1::";
    //cin>>&no1; error  // scanf("%d", &no1);
    cin>>no1; // correct
    cout<<"Enter No2::";
    cin>>no2; // correct

    cout<<"\tno1="<<no1<<"\t &no1="<<&no1<<endl;
    cout<<"\tno2="<<no2<<"\t &no2="<<&no2<<"\n";
    return 0;
}
```

```
#include<iostream>
using namespace std;
int main()
{
    cout<<"int data type"<<endl;
    {
        int no1, no2;
        cout<<"Enter No1::";
        //cin>>&no1; error  // scanf("%d", &no1);
        cin>>no1; // correct
        cout<<"Enter No2::";
        cin>>no2; // correct

        cout<<"\tno1="<<no1<<"\t &no1="<<&no1<<endl;
        cout<<"\tno2="<<no2<<"\t &no2="<<&no2<<"\n";
    }
    cout<<"==========================="<<endl;
```

```
cout<<"float data type"<<endl;
{
    float no1, no2;
    cout<<"Enter No1::";
    //cin>>&no1; error  // scanf("%d", &no1);
    cin>>no1; // correct
    cout<<"Enter No2::";
    cin>>no2; // correct

    cout<<"\tno1="<<no1<<"\t &no1="<<&no1<<endl;
    cout<<"\tno2="<<no2<<"\t &no2="<<&no2<<"\n";
}
cout<<"=========================="<<endl;
```

```
cout<<"==========================="<<endl;

    cout<<"char data type"<<endl;
    {
        char no1, no2;
        cout<<"Enter No1::";
        //cin>>&no1; error  // scanf("%d", &no1);
        cin>>no1; // correct
        cout<<"Enter No2::";
        cin>>no2; // correct

        cout<<"\tno1="<<no1<<"\t &no1="<<(void*)&no1<<endl;
        cout<<"\tno2="<<no2<<"\t &no2="<<(void*)&no2<<"\n";
    }
    cout<<"===========================\n\n\n\n\n"<<endl;
    return 0;
}
```

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

```
#pragma pack(1) // slack bytes
class student
{
    private:  // variable // data member  // field
        int rollno;
        char name[10];
        float per;
    public:
    //void accept_stud_info(student *const this)
    void accept_stud_info()
    {
        cout<<"\n enter rollno::";
        cin>>this->rollno;
        cout<<"\n enter name::";
        cin>>this->name;
        cout<<"\n enter per::";
        cin>>this->per;
        return;
    }
```

```
  //void display_stud_info(student *const this)
   void display_stud_info()
   {
      cout<<"\n rollno  name  per "<<endl;
      cout<<" "<<this->rollno<<" "<<this->name<<"  "<<this->per;
      return;
    }
};
int main(void)
{
    student s1;//class student s1;
    cout<<"\n enter student info::";
    s1.accept_stud_info();

    cout<<"student info :: \n";
    s1.display_stud_info();
    return 0;
}
```

Sunbeam

Default arguments

Assigning default values to the arguments of function is called default arguments.
Default arguments are always assigned from right to left direction.

Passing these arguments while calling a function is optional. If such argument is not passed, then its default value is considered. Otherwise arguments are treated as normal arguments

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

```
int sum(int a=0, int b=0, int c=0, int d=0)  // right to left
{
    cout<<" a="<<a<<" b="<<b<<" c="<<c<<" d="<<d<<endl;
    return a+b+c+d;
}
int main()
{
    int res=sum(10, 20,30, 40);
    cout<<"res="<<res<<endl; // a=10 b=20 c=30 d=40
    res=sum(10, 20,30);
    cout<<"res="<<res<<endl; // a=10 b=20 c=30 d=0
    res=sum(10, 20);
    cout<<"res="<<res<<endl; // a=10 b=20 c=0 d=0
    res=sum(10);
    cout<<"res="<<res<<endl; // a=10 b=0 c=0 d=0
    res=sum();
    cout<<"res="<<res<<endl; // a=0 b=0 c=0 d=0
    return 0;
}
```

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

```
int sum(int a=0, int b=0, int c=0, int d=0);  // right to left
int main()
{
    int res=sum(10, 20,30, 40);
    cout<<"res="<<res<<endl; // a=10 b=20 c=30 d=40
    res=sum(10, 20,30);
    cout<<"res="<<res<<endl; // a=10 b=20 c=30 d=0
    res=sum(10, 20);
    cout<<"res="<<res<<endl; // a=10 b=20 c=0 d=0
    res=sum(10);
    cout<<"res="<<res<<endl; // a=10 b=0 c=0 d=0
    res=sum();
    cout<<"res="<<res<<endl; // a=0 b=0 c=0 d=0
    return 0;
}
int sum(int a, int b, int c, int d)
{
    cout<<" a="<<a<<" b="<<b<<" c="<<c<<" d="<<d<<endl;
    return a+b+c+d;
}
```

Member Functions

- **Member functions are generally declared as public members of class.**
- **Constructor : Initialize Object**
- **Destructor : De-initialize Object**
- **Mutators : Modifies state of the object**
- **Inspectors : Don’t Modify state of object**
- **Facilitator : Provide facility like IO**

Constructor

- We can have constructors with
  - No argument : initialize data member to default values
  - One or more arguments : initialize data member to values passed to it
  - Argument of type of object : initialize object by using the values of the data members of the passed object. It is called as copy constructor.

Copy Constructor

- Complex c1(c2);
- This statement gives call to copy constructor. State of c1 become same as that of c2.
- If we don’t write, compiler provides default copy constructor.
- Generally, we implement copy constructor when we have pointer as data member which is pointing to dynamically allocated memory.

Constructor

- **Constructor is a member function of class having same name as that of class and don't have any return type.**
- **Constructor get automatically called when object is created i.e. memory is allocated to object.**
- **If we don't write any constructor, compiler provides a default constructor.**

Destructor

- **Destructor is a member function of class having same name as that of class preceded with ~ sign and don’t have any return type and arguments.**
- **Destructor get automatically called when object is going to destroy i.e. memory is to be de-allocated.**

Destructor

- **If we don’t write, compiler provides default destructor.**
- **Generally, we implement destructor when constructor of the object is allocating any resource**
- **e.g. we have pointer as data member, which is pointing to dynamically allocated memory.**

- **Size of empty class object is always 1 byte**
- **When you create object of an class it gets 3 characteristics.**
    - **State**
    - **Behavior**
    - **Identity**
- **When you create object of empty class at that time state of object is nothing. Behavior of that object is also nothing. But that object have unique identity(address).**
- **memory in computer is always organized in form of bytes.**
- **Byte is unit of memory. Minimum memory at objects unique address is one byte that's why size of empty class object is one byte.**

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

```
class Empty
{
};
struct empty
{
};
int main()
{
    empty e1;
    Empty e2;
    cout<<"size of e1="<<sizeof(e1)<<endl;
    cout<<"size of e2="<<sizeof(e2)<<endl;

    cout<<"&e1="<<&e1<<endl;
    cout<<"&e2="<<&e2<<endl;
    //cout<<"&empty="<<&empty<<endl;
    //cout<<"&Empty="<<&Empty<<endl;
    return 0;
}
```