Aim:

To understand the concept of **Inline Functions** with the help of on example.

Theory:

C++ inline function is powerful concept that is commonly used with classes. If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.

Any change to an inline function could require all clients of the function to be recompiled because compiler would need to replace all the code once again otherwise it will continue with old functionality.

To inline a function, place the keyword inline before the function name and define the function before any calls are made to the function. The compiler can ignore the inline qualifier in case defined function is more than a line.

A function definition in a class definition is an inline function definition, even without the use of the inline specifier.

Syntax :

```
inline return_type function_name ( **args ) {
    // function body
}
```

Code :

```
#include <iostream>

using namespace std;

inline int square(int x){
    return x*x ;
}

int main (){
    int num ;
    cout << "Enter a Number" << endl ;
    cin >> num ;
    cout << "Square of Number : " << square(num) << endl ;
}
```

Output :

```
PS D:\College\OOPS> .\inline-function
Enter a Number
12
Square of Number : 144
```

Discussion :

The program demonstrates the concept of inline function using the function `square`. The function is expanded in line when it is called. When the function is called whole code of the inline function gets inserted or substituted at the point of inline function call. This substitution is performed by the C++ compiler at compile time. Inline function may increase efficiency if it is small.

Learning Outcomes :

- Function call overhead doesn't occur.
- It also saves the overhead of push/pop variables on the stack when function is called.
- It also saves overhead of a return call from a function.
- When you inline a function, you may enable compiler to perform context specific optimization on the body of function. Such optimizations are not possible for normal function calls. Other optimizations can be obtained by considering the flows of calling context and the called context.
- Inline function may be useful (if it is small) for embedded systems because inline can yield less code than the function call preamble and return.

Aim:

To understand the concept of **Friend Functions** with the help of on example.

Theory:

A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all of its members are friends.

To declare a function as a friend of a class, precede the function prototype in the class definition with keyword `friend` .

Syntax :

```
class A {
   //data members

   public:
      friend void name( A a );
      //body
};
```

Code :

```
#include <iostream>

using namespace std ;


class B ;

class A {
    int x ;
    public :
        void input(){
            cout << "Enter the 'x' of Class A : "  ;
            cin >> x ;
        }
        void display() {
            cout << "'x' of class A is : " << x << endl;
        }
        friend float mean( A , B) ;
} ;
class B {
    int y ;
```

```
    public :
        void input(){
            cout << "Enter the 'y' of Class B : " ;
            cin >> y ;
        }
        void display() {
            cout << "'y' of class B is : " << y << endl;
        }
        friend float mean( A , B) ;
} ;

float mean(A a , B b){
    return (a.x + b.y)/ 2.0 ;
}

int main(){
    A a;
    B b;

    a.input() ;
    b.input() ;

    a.display() ;
    b.display() ;

    cout << "Mean is : " <<  mean(a,b) << endl;

    return 0;
}
```

Output :

```
PS D:\College\OOPS> .\mean-freind-fucntion
Enter the 'x' of Class A : 12
Enter the 'y' of Class B : 15
'x' of class A is : 12
'y' of class B is : 15
Mean is : 13.5
```

Discussion :

The program demonstrates the concept of friend function using the class `A` and `B` . The function `mean` is friend of both classes and can access the private members of both the classes. The function calculates the mean of data members and returns it.

Learning Outcomes :

- Provides additional functionality which is kept outside the class.
- Provides functions that need data which is not normally used by the class.
- Allows sharing private class information by a non member function.

Aim:

To understand the concept of **Pointers in Classes** with the help of an example.

Theory:

Just like pointers to normal variables and functions, we can have pointers to class member functions and member variables. The members must be public for pointers to exist.

Pointer to Data Members

We can use pointer to point to class's data members (Member variables).
`datatype class_name::*pointer_name = class_name::datamember_name ;`

Pointer to Member Functions

Pointers can be used to point to class's Member functions.
`return_type (class_name::*ptr_name) (argument_type) = &class_name::function_name ;`

Syntax :

```
class A {
   //data members

   public:
      //body
};

// Pointer to Member Function
int* (A::* ptr)() = &A::func  ;

//Pointer to Data Member
int A::*ptr = &A::x;
```

Code :

```
#include <iostream>

using namespace std ;

class A {
    int x ;
    int y ;
    public :
        void input(){
            cout << "Enter the 'x' of Class A : "  ;
            cin >> x ;
            cout << "Enter the 'y' of Class A : "  ;
            cin >> y ;
        }
        void display() {
            cout << "'x' of class A is : " << x << endl;
            cout << "'y' of class A is : " << y << endl;
        }
```

```
        int* getXPtr(){
            return &x ;
        }
        int* getYPtr(){
            return &y ;
        }
        friend void exchange(A) ;
} ;

int* (A::* ptrx)() = &A::getXPtr ;
int* (A::* ptry)() = &A::getYPtr ;


void exchange(A* a){

    int* px = (a->*ptrx)() ;
    int* py = (a->*ptry)() ;

    int temp = *py ;
    *py =  *px ;
    *px =  temp ;

}

int main(){
    A a;
    a.input() ;
    a.display() ;

    cout << "Exchanging Numbers" << endl ;
    exchange(&a) ;

    a.display();

    return 0;
}
```

Output :

```
PS D:\College\OOPS> .\exchange-friend-function
Enter the 'x' of Class A : 12
Enter the 'y' of Class A : 13
'x' of class A is : 12
'y' of class A is : 13
Exchanging Numbers
'x' of class A is : 13
'y' of class A is : 12
```

Discussion :

The program demonstrates the use of friend function for exchanging the data members of class A. The program uses pointer to data members and pointer to member functions to exchange the data members.

The pointer to member function `getXPtr()` and `getYPtr()` are invoked to return pointer to members `x` and `y` respectively. These pointers are used to exchange values.

Learning Outcomes :

- The class will be smaller in memory by use of pointers
- We can change the value and behaviour of these pointers on runtime. That means, you can point it to other member function or member variable.
- To have pointer to data member and member functions we need to make them public.