FACE

‘this’ pointer in C++

It is important to know how objects look at functions and data members of a class.

- Each objects gets its own copy of the data members
- All objects share a single copy of member function

```cpp
#include<iostream>
class Book
{
	public:
		int Id;
	void display(int x)
	{
		Id=x;
		std::cout<<Id<<std::endl;
	}
};
int main()
{
	Book b1,b2;
	b1.display(10);
	b2.display(20);
}
```

Output

```
10
20
```

Where is it used?

- When local variable name is same as member’s name
- To return reference to the calling object

FACE

```cpp
#include<iostream>
class Book
{
        public:
                int Id;
        void display(int Id)
        {
                this->Id=Id;
                std::cout<<this->Id<<std::endl;
        }
};
int main()
{
        Book b1;
        b1.display(10);
        std::cout<<b1.Id;
}
```

Output

```
10
10
```

```cpp
#include<iostream>
class Super
{
      public:
            Super &print()
            {
                  std::cout<<"Hello\n";
                  return *this;
            }
};
int main()
{
      Super s1;
      s1.print().print();
}
```

Chained Function Calls

Output

```
Hello
Hello
```

```cpp
#include<iostream>
class Test
{
        private:
                int x;
                int y;
        public:
                Test(int x, int y)
                {
                        this->x = x;
                        this->y = y;
                }
                Test &setX(int a)
                {
                        x = a;
                        return *this;
                }
                Test &setY(int b)
                {
                        y = b;
                        return *this;
                }
```

```
                    void print()
          {
               std::cout<<"x = "<<x<<"y="<<y<<std::endl;
          }
};
int main()
{
Test obj1(5, 5);
obj1.print();
obj1.setX(10).setY(20);
obj1.print();
return 0;
}
```

Chained Function Calls

OUTPUT

```
x = 5 y = 5
x = 10 y = 20
```

Static in C++

1. **Static variable:**
    - Variables in a class
    - Variables in a function
2. **Static members of class:**
    - Class object as static
    - Static function in class

Static Variables in functions

- Space is allocated for the lifetime of program
- Value of variable in the previous call gets carried through the next function call
- Useful where previous state of function is required

```cpp
#include<iostream>
int addition(int i)
{
        int sum=0;
        sum=sum+i;
        return sum;
}
int main()
{
        addition(5);
        std::cout<<addition(10);
}
```

Output

```
10
```

```cpp
#include<iostream>
int addition(int i)
{
        static int sum=0;
        sum=sum+i;
        return sum;
}
int main()
{
        addition(5);
        std::cout<<addition(10);
}
```

Output

```
15
```

Static Variables in class

- Shared by the objects
- Multiple copy of same variable is not possible
- Can not be initialized using constructors

```cpp
#include<iostream>
class Test
{
   public:
      static int i;
};
int main()
{
  Test obj1;
  Test obj2;
  obj1.i =2;
  obj2.i = 3;
  std::cout<<obj1.i<<" "<<obj2.i;
}
```

Output

```
Error: Undefined refrence to ‘Test::i’
```

```cpp
#include<iostream>
class Test
{
   public:
     static int i;
};
int Test::i=3;
int main()
{
  Test obj1;
  Test obj2;
  std::cout<<obj1.i<<" "<<obj2.i;
}
```

Output

```
3 3
```

Class objects as static

- Like variables class objects can be static
- Scope of static object is through out the life time of program

```cpp
#include<iostream>
class Object
{
    public:
        Object()
        {
            std::cout << "Inside Constructor\n";
        }
        ~Object()
        {
            std::cout << "Inside Destructor\n";
        }
};
int main()
{
    if (1)
    {
        Object obj;
    }
    std::cout<<"End of main\n";
}

```

OUTPUT

```
Inside Constructor
Inside Destructor
End of main
```

```cpp
#include<iostream>
class Object
{
    public:
        Object()
        {
            std::cout << "Inside Constructor\n";
        }
        ~Object()
        {
            std::cout << "Inside Destructor\n";
        }
};
int main()
{
    if (1)
    {
        static Object obj;
    }
    std::cout<<"End of main\n";
}

```

OUTPUT

```
Inside Constructor
End of main
Inside Destructor
```

Static function in a class

- Does not depend on object of class
- Invoke the static members using the scope name and scope resolution operator
- Static member function are allowed to access only the static data members and other static member function

FACE

```cpp
#include<iostream>
class Object
{
    public:
       static void print()
       {
                  std::cout<<"Static function is running";
         }
};
int main()
{

      Object::print();
}
```

Output

```
Static function is running
```

Mutable Keyword

- mutable keyword is used with member variables of class, which we want to change even if the object is of const type.
- Hence, mutable data members of a const objects can be modified.

```cpp
class Zee
{
    int i;
    mutable int j;
    public:
    Zee()
    {
        i = 0;   j = 0;
    }
    void fool() const
    {
        i++;    // will give error
        j++;    // works, because j is mutable
    }
};
int main()
{
    const Zee obj;
    obj.fool();
}
```

Const Keyword

- Using const keyword, we cannot change its value.
- Also, the constant variables must be initialized while they are declared.

```
int main
{
    const int i = 10;
    const int j = i + 10;     // works fine
    i++;    // this leads to error
}
```

FACE

Const Keyword

Defining Class's Member function as const

- A const member function never modifies data members in an object.

```
Syntax:

 return_type  function_name()  const;
```

Const

const function – can be accessed by const object and also by non-const object.

```cpp
#include<iostream>
using namespace std;
class Sample
{
	public:
	int i;
	Sample(int x)
	{
		i = x;
	}
	int alpha() const
	{
		cout << i << endl;
	}
	int gamma()
	{
		i++;
		cout << i << endl;
	}
};


int main()
{
		Sample s1(10);
		const Sample s2(20);
		s1.alpha();
		s2.alpha();
		cout<<s1.i<<" "<<s2.i<<endl;
		s1.gamma();
}
```

OUTPUT

```
10
20
10 20
11
```

Const

Non-const function – can be accessed only by const non-const object.

```cpp
#include<iostream>
using namespace std;
class Sample
{
    public:
    int i;
    Sample(int x)
    {
        i = x;
    }
    int alpha() const
    {
        i++;
        cout << i << endl;
    }
    int gamma()
    {
        i++;
        cout << i << endl;
    }
};

int main()
{
       Sample s1(10);
       const Sample s2(20);
       s1.alpha();
       s2.alpha();
       cout<<s1.i<<" "<<s2.i<<endl;
       s1.gamma();
       s2.gamma(); // compile error
}
```

OUTPUT

```
error: passing 'const Sample' as 'this' argument discards qualifiers [-
fpermissive]
         s2.gamma();
in call to 'int Sample::gamma()'
         int gamma()
```

Const

A const member function never modifies data members in an object.

```cpp
#include<iostream>
using namespace std;
class Sample
{
    public:
    int i;
    Sample(int x)
    {
        i = x;
    }
    int alpha() const
    {
        i++;
        cout << i << endl;
    }
    int gamma()
    {
        i++;
        cout << i << endl;
    }
};

int main()
{
        Sample s1(10);
        const Sample s2(20);
        s1.alpha();
        s2.alpha();
        cout<<s1.i<<" "<<s2.i<<endl;
        s1.gamma();
}
```

OUTPUT

```
prog.cpp: In member function 'int Sample::alpha() const':
prog.cpp:13:10: error: increment of member 'Sample::i' in read-only
object
         i++;
```

Const Keyword

- A const member function never modifies data members in an object.
- Const function - can be accessed by const object and also by non-const object.
- Non-const function - can be accessed only by non-const object.
- A const object cannot be used with a member function which tries to change its data members.

Inline Functions

- Function that is expanded in line when it is called
- This expansion is performed by the C++ compiler at compile time
- Used to save time in switching between the functions
- Syntax

```
inline void fun(int a)
{
         .
         .
         .
}
```

```cpp
#include <iostream>
using namespace std;
inline int Max (int x,int y)
{
    return (x>y)?x:y;
}
int main()
{
      cout<< "Max(20,10): "<<Max(20,10)<<endl;
      cout<< "Max(0,200): "<<Max(0,200)<<endl;
      cout<< "Max(100,1010): "<<Max(100,1010)<<endl;
      return 0;
}
```

OUTPUT

```
Max(100,1010): 1010
```

Question 1

Which of the following is a limit on inline functions?

A) Inline functions cannot return a value

B) Inline functions must return a value

C) Inline functions must be less than ten lines.

D) The compiler may choose to ignore an inline directive

Question 2

Why would you want to use inline functions?

A) To decrease the size of the resulting

B) To increase the speed of the resulting program

C) To simplify the source code file

D) To remove unnecessary functions

Question 3

Comment on the following code?

```
class A
{
    public:
      void func1()
      {
      }
      void func2();
};
inline void A::func2()
{
}
```

Question 3

A) Func1 is inline function

B) Func2 only is inline function

C) Func2 only is inline function

D) None of the above is inline

Question 4

What is the output for the following?

```
#include <iostream>
using namespace std;
class X
{
    private:
    static const int a = 76;
    public:
```

```
    static int getA()
    {
        return a;
    }
};
int main()
{
    cout <<X::getA()<<endl;
    return 0;
}
```

A) 76 **B)** 67 **C)** Runtime error **D)** Compile time error

FACE

Question 5

In C++ programming, cout is a/an

A) Function

B) Operator

C) Object

D) Macro

FACE

Question 6

Building block of C++ that leads to object oriented programming is termed as

- **A)** class
- **B)** object
- **C)** function
- **D)** construct

Question 7

Which of the following is a valid class declaration?

A) class A { int x; };

B) class B { }

C) public class A { }

D) object A { int x; };

Question 8

If a member function does not alter any data in the class, that may be declared as ....

A) constant member function

B) private member function

C) static member function

D) friend function

FACE

Question 9

What is the correct syntax of accessing a static member of a Class?

Example class:

```
class A
{
        public:
                static int value;
}
```

Question 9

A) A.value

B) A::value

C) A->value

D) A^value

THANK YOU

FACE