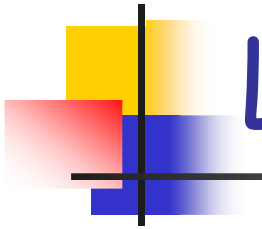
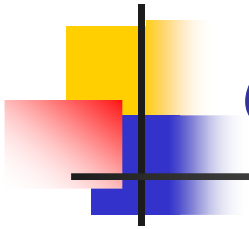


Lecture-4

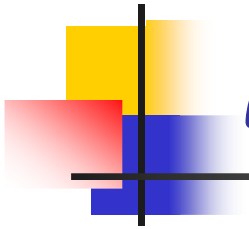
- Const functions
- Passing arguments
  - By values
  - By reference
- Templates



const member functions

```
class myClass
{
    int a;
    ...
    void f1( ) const
     { a = 3; } // Not allowed
};
```

- const functions can't change any attributes of myClass.
- function_1 can't change a in the above example



Making arguments const

```
class myClass
{
   int a;
   …
   void f1(const int i )
    {  i = 3; } // not allowed
```

- Cannot change the value of const arguments

Passing args. to a function … by value

- Compiler creates its own copy.
- Any changes made inside the function are not reflected after the function.

```
class myClass
{
        void f1(int i ) // i is passed by value
         { i = 3; }
};
int x = 5;
myClass obj;
obj.f1( x );
cout << "value of x: " << x << endl; // x is still 5
```

Passing args. to a function … by reference

- Compiler takes the original object.
- Any changes made inside the function are reflected after the function.

```
class myClass
{
        void f1(int& i ) // i is passed by reference.
         { i = 3; }
};
int x = 5;
myClass obj;
obj.f1( x );
cout << "value of x: " << x << endl; // x is 3
```

Templates – motivation

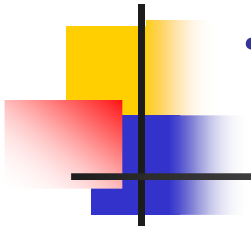
- Consider a function to sort integers

```
int findMaxInt (int num1, int num2)
{
        if (num1 > num2) return num1;
        return num2;
}
```

- Consider a function to sort floats

```
float findMaxFloat (float num1, float num2)
{
        if (num1 > num2) return num1;
        return num2;
}
```

- Same code, but two functions are needed
  - One for each data type



Templates motivation … cont.

- Problem here
  - One sorting function is needed for each data type (arguments).
  - But the sorting code itself is the same.
- Templates are used to solve this issue.
- Templates
  - Define functions with generic types
  - Define generic classes.
  - Same code can be used with any data type.
  - No need for code repetition.

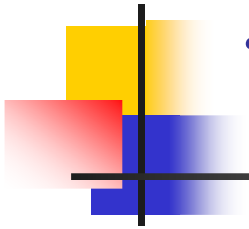
Template functions

- Define functions with generic types

```
template <class anyType>
        function definition
```

- Example

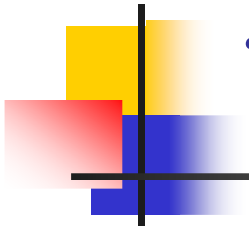
```
template <class anyType>
    anyType GetMax (anyType a, anyType b)
    {
            if (a > b) return a;
            return b;
    }
```



Template classes

- Define any class
- Use template definition to define any generic data type.
- Rest of the code remains the same.

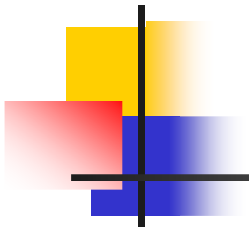
```
template <class anyType>
class someClass
{
    someClass (…) // constructor
    ~someClass( ) // destructor
    member data & methods // can use anyType
};
```



Template class example

```
template <class anyType>
class myClass
{
   private:
     anyType value1, value2;

    public:
       myClass (anyType i, anyType j) : value1(i), value2(j) {  }
       ~myClass() { }
       anyType findMaxValue ()
       {
            if (value1 > value2) return (value1);
            return (value2);
       }
};
```



Template class example … contd.

```
main()
{
   myClass <int> intObject (2, 3);
   int maxInt = intObject.findMaxValue();
   cout << "maxInt: " << maxInt << endl;

   myClass <float> floatObject (4.3, 3.2);
   float maxFloat = floatObject.findMaxValue();
   cout << "maxFloat: " << maxFloat << endl;
}
```