Lecture 17
OOP

Khola Naseem
khola.naseem@uet.edu.pk

Credit: Khola Naseem

Class Templates:

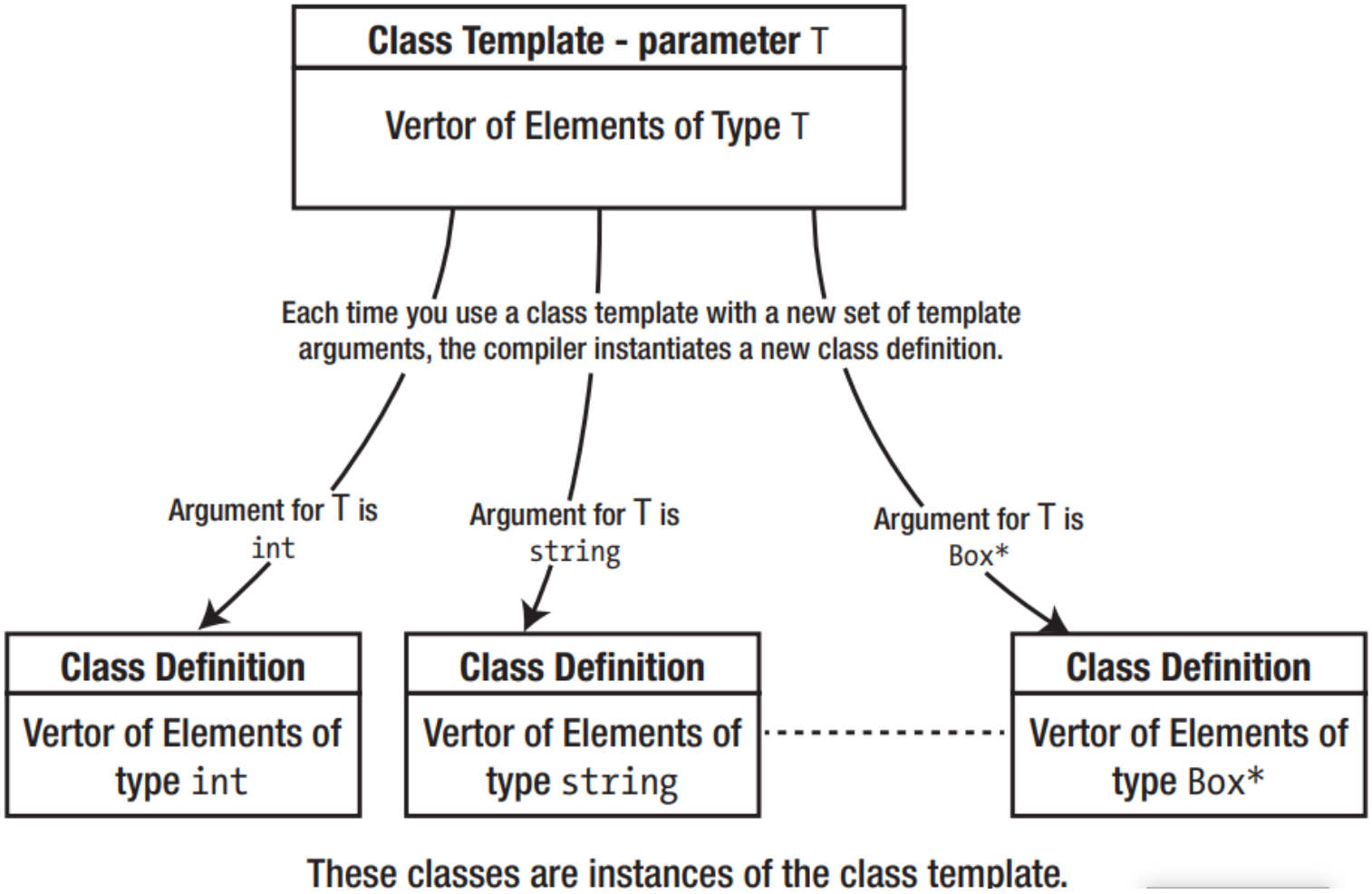
- A class template is based on the same idea as a function template
- a class template definition in itself has no executable code associated with it. It's only when the compiler instantiates concrete class definitions from a class template that it gives rise to actual code.
- E.g. vector<>, array<>
- **Defining Class Templates:**

```
template <template parameter list>
class ClassName
{
  /
};
```

- As with function templates, the template parameter list of a class template can contain any number of parameters of two kinds—type parameters and non-type parameters.

Class Templates:

- A class template



Credit: Khola Naseem

Class Templates:

```
template <class T>
class className {
  private:
    T var;
    ... .. ...
  public:
    T functionName(T arg);
    ... .. ...
};
```

Class Templates:

```
template <typename T>
class className {
  private:
    T var;
    ... .. ...
  public:
    T functionName(T arg);
    ... .. ...
};
```

Class Templates:

- Creating a Class Template Object
  - className<dataType> classObject;
- Example:

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

// Class template
template <typename T>
class Number {
   private:

    T num;

   public:
    Number(T n) : num(n) {}   // constructor

    T getNum() {
        return num;
    }
};

int main() {
    Number<int> numInt(9);
    Number<double> numDouble(9.7);
    cout << "int value " << numInt.getNum() << endl;
    cout << "double value " << numDouble.getNum() << endl;

    return 0;
}
```

Output:

```
int value 9
double value 9.7
```

Credit: Khola Naseem

Class Templates:

- Define outside the class

```
template <typename T>
class ClassName {
    … .. …
    // Function prototype
    returnType functionName();
};
// Function definition
template <typename T>
returnType ClassName<T>::functionName() {
    // code
}
```

Credit: Khola Naseem

Class Templates:

- **Defining a Class Member Outside the Class Template**

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

// Class template
template <typename T>
class Number {
   private:
    T num;
   public:
    Number(T n) : num(n) {}   // constructor
    T getNum() ;
};
//definition outside the class
template <typename T>T Number<T>:: getNum() {
    return num;
}
int main() {
    Number<int> numInt(9);
    Number<double> numDouble(9.7);
    cout << "int value " << numInt.getNum() << endl;
    cout << "double value " << numDouble.getNum() << endl;
    return 0;
}
```

Credit: Khola Naseem

Class Templates:

- Simple Calculator Using Class Templates

```cpp
#include <iostream>
using namespace std;
template <typename T>
class Calculator {
   private:
    T num1, num2;
   public:
    Calculator(T number1, T number2) {
        num1 = number1;
        num2 = number2;
    }
    void displayResult() {
        cout << "Numbers: " << num1 << " and " << num2 << "." << endl;
        cout << num1 << " + " << num2 << " = " << add() << endl;
        cout << num1 << " - " << num2 << " = " << sub() << endl;
        cout << num1 << " * " << num2 << " = " << mul() << endl;
        cout << num1 << " / " << num2 << " = " << divide() << endl;
    }
    T add() { return num1 + num2; }
    T sub() { return num1 - num2; }
    T mul() { return num1 * num2; }
    T divide() { return num1 / num2; }
};
int main() {
    Calculator<int> intCal(11, 6);
    Calculator<float> floatCal(7.4, 6.2);
    cout << "int results:" << endl;
    intCal.displayResult();
    cout << endl
         << "Float results:" << endl;
    floatCal.displayResult();
    return 0;
}
```

Output:

```
int results:
Numbers: 11 and 6.
11 + 6 = 17
11 - 6 = 5
11 * 6 = 66
11 / 6 = 1

Float results:
Numbers: 7.4 and 6.2.
7.4 + 6.2 = 13.6
7.4 - 6.2 = 1.2
7.4 * 6.2 = 45.88
7.4 / 6.2 = 1.19355
```

Class Templates:

- Example

```
#include <iostream>
using namespace std;
template <typename T>
class Add {
   private:
    T num1, num2;
   public:
    Add(T number1, T number2) {
        num1 = number1;
        num2 = number2;
    }
    void displayResult() {

        cout << num1 << " + " << num2 << " = " << add() << endl;

    }
    T add() { return num1 + num2; }

};
int main() {
    Add<int> intAdd(11, 6);
    Add<float> floatAdd(7.4, 6.2);
    Add<string> StringAdd("Khola "," Naseem");
    cout << "int results:" << endl;
    intAdd.displayResult();
    cout << endl
         << "Float results:" << endl;
    floatAdd.displayResult();
    cout << endl
         << "String results:" << endl;
    StringAdd.displayResult();
    return 0;
}
```

Output:

```
int result
11 + 6 = 17

Float result:
7.4 + 6.2 = 13.6

String result:
Khola  +  Naseem = Khola  Naseem
```

Credit: Khola Naseem

C++ Templates With Multiple Parameters:

- Example

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

// Class template with multiple and default parameters
template <typename S, typename T, typename U = char>
class ClassTemplate {
   private:
    S var1;
    T var2;
    U var3;

   public:
    ClassTemplate(S v1, T v2, U v3) : var1(v1), var2(v2), var3(v3) {}

    void printVar() {
        cout << "var1 = " << var1 << endl;
        cout << "var2 = " << var2 << endl;
        cout << "var3 = " << var3 << endl;
    }
};

int main() {

    ClassTemplate<int, double> object(20, 8.7, 'a');
    cout << "object values: " << endl;
    object.printVar();

    ClassTemplate<double, char, bool> object2(8.4, 's', false);
    cout << "\n object2 values: " << endl;
    object2.printVar();

    return 0;
}
```

Output

```
object values:
var1 = 20
var2 = 8.7
var3 = a

 object2 values:
var1 = 8.4
var2 = s
var3 = 0
```

Credit: Khola Naseem

C++ Templates With Multiple Parameters:

- Example

```
template <typename T> class Array {
private:
    T* ptr;
    int size;
public:
    Array(T arr1[], int s);
    void print();
};
template <typename T> Array<T>::Array(T arr1[], int sz)
{
    ptr = new T[sz];
    size = sz;
    for (int i = 0; i < size; i++)
        ptr[i] = arr1[i];
}
template <typename T> void Array<T>::print()
{
    for (int i = 0; i < size; i++)
        cout << " " << *(ptr + i);
    cout << endl;
}
int main()
{
    int arr[] = { 1, 2, 3, 4, 5 };
    Array<int> a(arr, 5);
    a.print();
    cout<<endl;
    float arr2[] = { 3.4, 2.6, 3.0, 42.2, 50.5 };
    Array<float> a2(arr2, 5);
    a2.print();
    cout<<"\n";
    char arr3[] = { 'a' ,'b','b','d','e'};
    Array<char> a3(arr3, 5);
    a3.print();
    return 0;
}
```

Output

```
1 2 3 4 5

3.4 2.6 3 42.2 50.5

a b b d e
```

Credit: Khola Naseem

Class Templates:

- A class template

```
template <typename T>
class Array
{
public:
  explicit Array<T>(size_t size);           // Constructor
  ~Array<T>();                              // Destructor
  Array<T>(const Array<T>& array);          // Copy constructor
  Array<T>& operator=(const Array<T>& rhs); // Copy assignment operator
  T& operator[](size_t index);              // Subscript operator
  const T& operator[](size_t index) const;  // Subscript operator-const arr
  size_t getSize() const { return m_size; } // Accessor for m_size

private:
  T* m_elements;    // Array of type T
  size_t m_size;    // Number of array elements
};
```

Credit: Khola Naseem

Class Templates:

- A class template

```
template <class T> class DynArray {
protected:
    int size;
    T * DynamicArray;
public:
    DynArray(){};
    DynArray(size_t s): size(s) {
        DynamicArray = new T[size];
        for (int i = 0; i<size; i++) {
            cout << "Element " << i << ": ";
            cin >> DynamicArray[i];
        }
    }
    void show(){
        for (int i=0; i<size; i++) {
            cout << DynamicArray[i] << endl;
        }
    }
    ~DynArray() {
        delete []DynamicArray;
    }
};
int main() {
    int sizeOfArry;
    cout << "Enter size of Array: ";
    cin >> sizeOfArry;
    DynArray<int> intArray = DynArray<int>(sizeOfArry);
    intArray.show();
}
```

```
Enter size of Array: 4
Element 0: 2
Element 1: 5
Element 2: 4
Element 3: 1
2
5
4
1
```

Credit: Khola Naseem

Class Templates:

- A class template

```
template <class T> class DynArray {
protected:
    int size;
    T * DynamicArray;
public:
    DynArray(){};
    DynArray(size_t s): size(s) {
        DynamicArray = new T[size];
        for (int i = 0; i<size; i++) {
            cout << "Element " << i << ": ";
            cin >> DynamicArray[i];
        }
    }
    void show(){
        for (int i=0; i<size; i++) {
            cout << DynamicArray[i] << endl;
        }
    }
    ~DynArray() {
        delete []DynamicArray;
    }
};
int main() {
    int sizeOfArry;
    cout << "Enter size of Array: ";
    cin >> sizeOfArry;
    DynArray<int> intArray = DynArray<int>(sizeOfArry);
    intArray.show();
    cout<<"string array:\n";
    DynArray<string> sArray = DynArray<string>(sizeOfArry);
    sArray.show();
}
```

```
Enter size of Array: 4
Element 0: 2
Element 1: 5
Element 2: 4
Element 3: 1
2
5
4
1
string array:
Element 0: y
Element 1: t
Element 2: r
Element 3: e
y
t
r
e
```

Credit: Khola Naseem

Class Templates:

- A class template

```
template <typename T>
class Array
{
public:
  explicit Array(size_t size);          // Constructor
  ~Array();                             // Destructor
  Array(const Array& array);            // Copy constructor
  Array& operator=(const Array& rhs);   // Copy assignment operator

  // Other members remain the same...
};
```

Credit: Khola Naseem

Conversion:

- Implicit conversion

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

class ComplexNumber {
private:
    double real;
    double imag;
public:
    ComplexNumber(double re = 0.0,
            double im = 0.0) : real(re), imag(im)
    {
    }
    //compare values
    bool operator == (ComplexNumber rhs)
    {
        return (real == rhs.real &&  imag == rhs.imag);
    }
};

int main()
{
    ComplexNumber com(4.0, 0.0);

    if (com == 4.0)
        cout << "Same values";
    else
        cout << "different values";
    return 0;
}
```

Credit: Khola Naseem

Conversion:

- Implicit conversion

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

class Number {
private:
    double real;

public:
   Number(int a){
        real=a;
        }
    int getval()
    {
        return real;
    }
};

int main()
{
    Number n=1;
    cout<<n.getval();
}
```

Reference material

For Practice Questions, refer to these books

- C++ Programming From Problem Analysis To Program Design, 5th Edition, D.S.Malik. Chapter 12.
- C++ How to Program, Deitel & Deitel, 5th Edition, Prentice Hall.
- Object Oriented Programming in C++ by Robert Lafore.
- Object Oriented Software Construction, Bertrand Meyer's
- Object-Oriented Analysis and Design with applications, Grady Booch et al, 3Rd Edition, Pearson, 2007
- Web