Chapter 16: Exceptions and Templates

Kafi Rahman
Assistant Professor
Truman State University

Function Template Notes

- Can define a template to use multiple data types:
  - `template<class T1, class T2>`
- Example:

```
template<class T1, class T2>
// T1 and T2 will be replaced with
// the data types of the arguments
double mpg(T1 miles, T2 gallons)
{
    return miles / gallons;
}
```

```
int main()
{
    cout<< mpg(100, 10)<<endl;
    cout<< mpg(100, 10.5)<<endl;
    cout<< mpg(100.5, 10.2)<<endl;
    cout<< mpg(100, ' ')<<endl;
    cout<< mpg(100, 10L)<<endl;
    //cout<< mpg(100, "14");
}
```

Function Template Notes

- Function templates can be overloaded Each template must have a unique parameter list

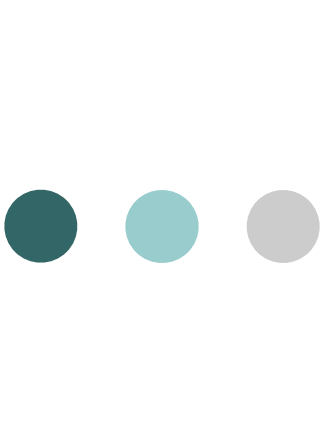
```
template <class T>
T sumAll(T num)
{...
}

template <class T1, class T2>
T1 sumall(T1 num1, T2 num2)
{ ...
}
```

Function Template Notes

- All data types specified in template prefix must be used in template definition
- Function calls must pass parameters for all data types specified in the template prefix
- Like regular functions, function templates must be defined before being called

Function Template Notes

- A function template is a pattern
- No actual code is generated until the function named in the template is used/called
- A function template uses no memory
- When passing a class object to a function template, ensure that all operators in the template are defined or overloaded in the class definition



16.3

Where to Start When Defining Templates

Where to Start When Defining Templates

- Templates are often appropriate for multiple functions that perform the same task with different parameter data types

- Develop function using usual data types first, then convert to a template:
  - add template prefix
  - convert data type names in the function to a type parameter (i.e., a T type) in the template and so forth.

16.4

Class Templates

Class Templates

- Classes can also be represented by templates.
- When a class object is created, type information is supplied to define the type of data members of the class.

- Unlike functions, classes are instantiated by supplying the type name (int, double, string, etc.) at object definition

Class Template Example

```
template <class T>
class Grade
{
      private:
        T score;
      public:
        Grade(T);
        void setGrade(T);
        T getGrade()
};
```

Class Template Example

- Pass type information to class template when defining objects:

```
Grade<int> math_test;

// array of objects
Grade<double> all_tests[10];
```

- Use as ordinary objects once defined

Class Template: more instance variable

```
// a template class of flexible type:
Student
template <class TINT, class TSTRING>
class Student
{
      private:
      TINT id;
      TSTRING name;

      public:
      // allocating n spaces
      Student(TINT id, TSTRING name)
      {     this->id = id;
            this->name = name;
      }

      // getting value at index pos
      void display()
      {     cout << id <<  + " " << name;
      }
};
```

```
// testing the defined class
int main()
{     // creating a student
      Student<int, string> student(100,
"Kafi");

      student.display();

      return 0;
}
```

Class Template: more instance variable

```
template <class TINT>
class Rectangle
{
     private:
     TINT width;
     TINT height;
     public:
     Rectangle(TINT w, TINT h)
     {    width = w;
          height = h;
     }

     void display()
     {
          cout << width << " "
               << height << endl;
     }
};
```

```
// testing the defined class
int main()
{    // another case
     Rectangle<int> rect (100, 1050);
     rect.display();
     return 0;
}
```

Class Template with Array

```
template <class T>
class Grade
{     private:
      int size;
      T * score; // array of T

      public:
      // allocating n spaces
      Grade(int n)
      {     size = n;
            score = new T[size];
      }
      // de-allocating the array
      ~Grade()
      {     delete [] score;
      }
      // setting value at index pos
      void setGrade(int index, T value);
      // getting value at index pos
      T getGrade(int index);
};
```

```
template <class T>
void Grade<T>::
setGrade(int index, T value)
{     score[index] = value;
}

template <class T>
T Grade<T>::getGrade(int index)
{     return score[index];
}


int main()
{     // creating 100 elements of strings
      Grade<string> myGrades(100);
      myGrades.setGrade(0, "Math");
      myGrades.setGrade(1, "Physics");

      // displaying the values
      cout<<myGrades.getGrade(0)<<endl;
      return 0;
}
```

Class Templates and Inheritance

- Class templates can inherit from other class templates:

```
template <class T>
class Rectangle
{ // definitions of the Rectangle class
};

template <class T>
class Square : public Rectangle<T>
{ // definitions of the Square class
};
```

- Must use type parameter T everywhere base class name is used in the derived class

Planet Plastic

- Since 1950, the world has created 6.3 trillion kilograms of plastic waste
  - and 91 percent has never been recycled even once.

Planet Plastic

- Of the 78 billion kilograms of plastic packaging materials produced in 2013, only 2% were recycled. The condition remains the same now.
- Roughly 8 billion kilograms of plastics enter the world's waters every year.
- Most plastics are shipped to Asian countries.
- What can we do?
  - The minimum that we can do would be to avoid using plastic water bottles.
  - Believe me, it will help.