Templates and Exceptions

"function templates, class templates, exceptions, exception handlers"

Fundamentals of OOPs

Shakirullah Waseeb
shakir.waseeb@gmail.com

Nangarhar University

November 27, 2017

Agenda

1. Overview
2. Function templates
   - Intuition
   - Function Template Example
   - How it works
3. Class templates
   - Introduction
   - A stack example
4. Questions and Discussion

Overview

- **Templates:** make it possible to use a function or class to work with variant data types
- **Exceptions:** provide a convenient, uniform way to handle errors that occur with in class

Agenda

1 Overview

2 Function templates
- Intuition
- Function Template Example
- How it works

3 Class templates
- Introduction
- A stack example

4 Questions and Discussion

Intuition for templates

- consider the problem of calculating the absolute value of a number (short, int, long, float, double)

Example

```
int abs(int a) {
  return a > 0 ? a : - a;
}
float abs(float a) {
  return a > 0 ? a : - a;
}
long abs(long a) {
  return a > 0 ? a : - a;
}
double abs(double a) {
  return a > 0 ? a : - a;
}
```

Intuition -continue

- Function body in previous slide example is the same what difference is the type of data which these functions operates on
- This can be achieved by using function overloading but again this would require time and wastes space in the listings
- An error in any of such function would require to remember the corrections in every function body, which leads to program inconsistencies
- It would be quite useful to have a way to write such functions just once and have worked for many data types

Agenda

1. Overview
2. Function templates
   - Intuition
   - Function Template Example
   - How it works
3. Class templates
   - Introduction
   - A stack example
4. Questions and Discussion

Function Template for absolute value

Example

```
#include <iostream >
using namespace std;
template <class T>
T abs(T a) {
    return a > 0 ? a : - a;
}
int main() {
    short s = 13;
    short sn = -15;
    int i = 13;
    int in = -15;
    float f = 13.5;
    float fn = -15.5;
    cout « abs(s) cout «endl;
    cout « abs(sn) cout «endl;
    cout « abs(i) cout «endl;
    cout « abs(in) cout «endl;
    cout « abs(f) cout «endl;
    cout « abs(fn) cout «endl;
}
```

Agenda

1. Overview
2. Function templates
   - Intuition
   - Function Template Example
   - How it works
3. Class templates
   - Introduction
   - A stack example
4. Questions and Discussion

How it works

- **Syntax**
  - the idea is to represent the data types used by the function are not specific (e.g. int, float, double) but rather by a name that can stand for any type
  - template keyword notify the compiler with a plan for template function
  - class keyword with in angle brackets could be called as **type**
  - variables following the keyword class is called **template arguments**
- **How the compiler translates**
  - function template itself doesn't cause the compiler to create any code
  - code is generated when a calling statement to function is seen
  - because when calling with a type (e.g. int) the compiler knows the type thus generates a version of function for given type (e.g. int)
  - this is called **instantiating** the function template

Agenda

1. Overview
2. Function templates
   - Intuition
   - Function Template Example
   - How it works
3. Class templates
   - Introduction
   - A stack example
4. Questions and Discussion

Intuition for templates

- the template concept can also be extended to classes
- class templates are mostly used for data storage classes (e.g link-list, stack, trees, graphs, etc)
- to store different data types we must define separate classes for each data type
- with the help of class templates we can define a template which can be used for any type
- let us observe the stack class template in the next slide

Agenda

1. Overview
2. Function templates
   - Intuition
   - Function Template Example
   - How it works
3. Class templates
   - Introduction
   - A stack example
4. Questions and Discussion

A stack example

Example Code

```
template <class T>
class Stack{
    private:
      T items[20];
      int top;
    public:
      Stack(): top(-1){};
      void push(T itm){
        items[top++] = itm;
      };
      T pop(){
        return items[top–];
      };
};
int main(){
    Stack<float> stackFloat;
    stackFloat.push(76.50);
    stackFloat.push(87.99);
    Stack<int> stackInteger;
    stackInteger.push(65);
    stackInteger.push(78);
}
```

Next Lecture

Exceptions

Your Turn: Time to hear from you!

[1]

[1] https://fensafitters.files.wordpress.com/2013/07/3d095.jpg

References

- Robert Lafore
  Object-Oriented Programming in C++, 4th Edition .
  2002.

- Piyush Kumar
  Object oriented Programming (Using C++)
  http://www.compgeom.com/ piyush/teach/3330