Templates

2016-11-01

Problems with typed parameters

- How to create a function that sums two numbers?
- What is a "number"?
- What operator do we need for each of these "numbers"?

Ideally: Number SumtwoNumbers(Number X, Number Y);

Possible answers

- Create a unique function for each type
  - Int SumTwoInts(Int x, Int y)
  - Float SumTwoFloats(int x, int y)
  - …
- Use function overloading (slightly better, but not by much)
  - Int SumTwoNumbers(Int x, int Y)
  - Float SumTwoNumbers(float x, float y)

Create a "Number" base class

```
Class Number{
….
Virtual Number & operator+(const Number &rhs) = 0;
…
}
```

But this seems really tedious, but nice option if it's built in. (Think Java)

The solution in C++ is Templates

```
Template< class TypeT>
TypeT SumNumbers( TypeT X, TypeT, Y ){
    return X + Y;
}
```

Provide type when calling the function: SumNumbers<TypeT>(x,y);

```
int a = 3;
int b = 6;
int c = SumNumbers<int>(a,b);
```

Templates are done at compile time

- Compiler does all the tedious work of creating each function as needed in the program.
- There are no dynamic or runtime templates in C++
- There are some interesting possibilities combining polymorphism and templates, but I won't be giving any realistic examples.
- In The scope of this class Templates are covered for using the Standard Template Library(STL)

Class Templates

- Work basically the same way as templated functions
- Again, This is done at compile time. Template arguments must be explicit and known at compile time.
- Compiler will create new classes every time a new template type is used for the templated class.

MyList<int> grades;

MyList<String> names;

MyList<Books> readingList;

```
Template<class ItemType>
Class List
{
   public:
      bool isEmpty() const;
      ….
   private:
   int    maxLength;
   int   currentSize;
   ItemType *data;
   ….
```