CPE 212 - Fundamentals of Software Engineering

•••

Templates

Outline

- Defining Templates
- Why use them?
- Coding Examples

Templates

Templates is a C++ topic that allows us to operate on generic types and inject the correct type as compile time.

Templates can be attached to a function or an object and are scoped within either the template or object.

This tool allows us to create highly generic data structures that operate abstractly, and allow the contained type to specialize operations as needed.

Why use them?

```cpp
void Function(int data);

void Function(float data);

void Function(std::string data);
```

Why use them?

```c
void swap_int(int *one, int *two)
{
    int tmp = *one;
    *one = *two;
    *two = tmp;
}
```

```c
void swap_float(float *one, float *two)
{
    float tmp = *one;
    *one = *two;
    *two = tmp;
}
```

```c
int main()
{
    int data_i[123];
    float data_f[123];

    swap_int(&data_i[0], &data_i[1]);
    swap_float(&data_f[0], &data_f[1]);
}
```

How do we solve this problem?

What we want

```
void swap(SomeType *one, SomeType *two)
{
    SomeType tmp = *one;
    *one = *two;
    *two = tmp;
}
```

Template Declaration

```
// Forward Declaration
template<typename T>
void Function(T data);

template<typename T>
void Function(T data)
{
   // do stuff with data of type T here
}
```

Swap Function using Templates

```
template<typename SomeType>
void swap(SomeType *one, SomeType *two)
{
    SomeType tmp = *one;
    *one = *two;
    *two = tmp;
}
```

```
int main()
{
   int data_i[123];
   float data_f[123];

   swap_int(&data_i[0], &data_i[1]);
   swap_float(&data_f[0], &data_f[1]);
}
```

Templates: Multiple Types

```cpp
template<typename TypeOne, typename TypeTwo>
void AddAndPrint(TypeOne varOne, TypeTwo varTwo)
{
    std::cout << varOne + varTwo << std::endl;
}
```

Templates: Multiple Types

```
AddAndPrint(1.0, 2);
```

Outputs

```
3
```

```
AddAndPrint(1.1, 2);
```

Outputs

```
3.1
```

```
AddAndPrint(1.0, "Hello!");
```

Outputs

```
?
```

Templates: Errors

```
AddAndPrint(1.0, "Hello!");
```

```
➜  Templates g++ Example.cpp -o Example
Example.cpp:7:25: error: invalid operands to binary expression ('double' and
'const char *')
    std::cout << varOne + varTwo << std::endl;
                 ~~~~~~ ^ ~~~~~~
Example.cpp:18:5: note: in instantiation of function template specialization
'AddAndPrint<double, const char *>' requested here
    AddAndPrint(1.2, "Hello!");
    ^
1 error generated.
```

Template Function Rules

Template function types are scoped to the function alone!

You do not have to explicitly declare the types you are using within a function. The compiler will interoperate this for you.

Types must have compatible operations, or a declaration for how to handle an operator.

You can call functions of template types, but an error will occur if that function call does not exist.

FAILING TO DO THE ABOVE WILL RESULT IN A COMPILE TIME ERROR RATHER THAN A RUN TIME ERROR.

Template Objects

```
struct Node
{
   typedef int Type;
   Type data;
   Node *next;

   Node(int data)
   {
       this->data = data;
       next = NULL;
   }
}
```

Template Objects

```
struct Node
{
    typedef int Type;
    Type data;
    Node *next;

    Node(int data)
    {
        this->data = data;
        next = NULL;
    }
}
```

```
template<typename Type>
struct Node
{
    Type data;
    Node<Type> *next;

    Node(const Type & t)
    {
        data = t;
        next = NULL;
    }
};
```

Template Objects

```
int main()
{
    Node<int> intNode(1);
    Node<float> floatNode(1.1f);
    Node<std::string> strNode("Hello World");
}
```

```
Node injectedNode(10.0f);
```

Template Objects

```cpp
Node injectedNode(10.0f);
```

```
➜  Templates g++ templateobjects.cpp -o Objects
templateobjects.cpp: In function ‘int main()’:
Templateobjects.cpp:26:10: error: missing template arguments before ‘injectedNode’
         Node  injectedNode(10.0f);
               ^~~~~~~~~~~~~
1 error generated.
```

Template linking issues

If you try to compile these files, one by one, later you will get a linking error saying undefined reference to for the methods of the class.

The code in the template is not sufficient to instruct the compiler to produce the methods that are needed by main.cpp (e.g. List<int>::push(...) and List<string>::push(...)) as the compiler does not know, while compiling List.cpp by itself, the data types it should provide support for.

This is because templates are not “compiled” until they are instantiated by providing a type.

Template linking issues

There are 3 solutions to this issue:

1. Implicitly declare the types at the end of the C++ implementation file
   a. template class List<int>;
   b. We will go this route in many examples.
2. implement all code directly into the header file.
   a. This is a ban if you do this in this class.
3. Include a “object_impl.hpp” in your header and put all your magic here.
   a. We will do this eventually as well.

Template Rules

Templates do not have to be just typenames, you can also provide other things into the template

```
template <int count>
struct Items
{
   int MyItems[count];
}

Items<200> itemData;
```

Template Resources

https://youtu.be/NIDEjY5ywqU This is a great video on templates from a very popular convention on C++. This has some niddy griddy on templates that is nice to know.

http://www.cplusplus.com/doc/oldtutorial/templates/ A great recap on templates.

Next week: Operator Overload

Book: 6.5, page 383 -> 387.

https://www.geeksforgeeks.org/operator-overloading-c/ is also a good read.