CS205 C/ C++ Programming - Lab Assignment 7

Name: 唐千栋(Qiandong Tang)

SID: 11612730

Part 1 - Analysis

The requirement is to design a template to determine whether a given type is incrementable at compile time using C++'s template metaprogramming techniques.

We can use the idea of SFINAE in C++. We inherit two `is_incrementable` from `true_type` and `false_type`. We set the default template to inherit from `false_type`. Its template has two template parameters. The first one is the class type, and the second one is default to be void. The second template has one template parameter. Since it uses first template, its second parameter sets to be `void_t`. It uses `std::declval<T>`, if the class `T` supports incrementable, it will use the second template, otherwise it will use first template.

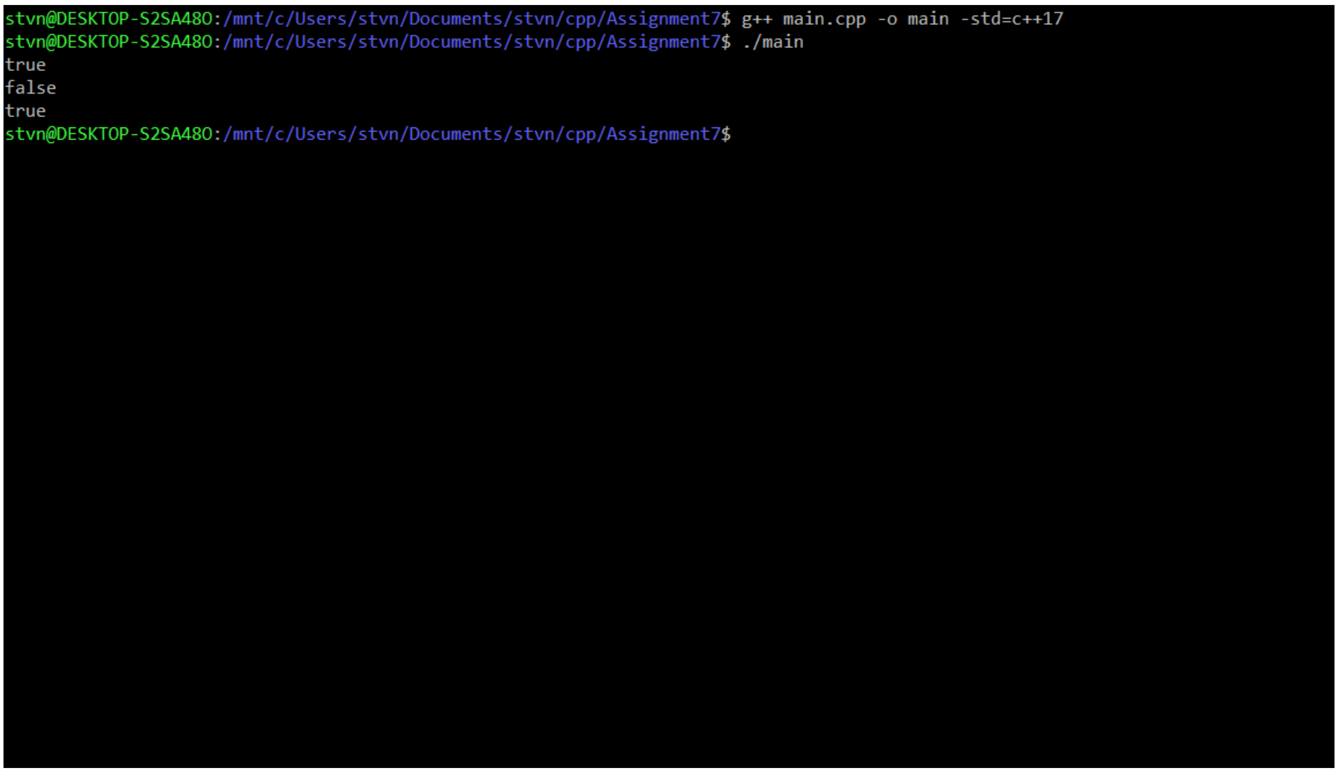
Part 2 - Code

```
#ifndef ASSIGNMENT7_TEMPLATE_HPP
#define ASSIGNMENT7_TEMPLATE_HPP

#include <cstdio>
#include <iostream>
#include <string>
#include <type_traits>

template<class, class = void>
struct is_incrementable : std::false_type {};

template<class T>
struct is_incrementable<T, std::void_t<decltype(((++std::declval<T&>())++))>> :
std::true_type {};

#endif //ASSIGNMENT7_TEMPLATE_HPP
```

Part 3 - Result & Verification

Test case:

```
#include "template.hpp"
using namespace std;

struct A {
    int b;

    const A operator++(int){
        A a{};
        a.b = b;
        b++;
        return a;
    };

    A& operator++() {
        b++;
        return *this;
    }

    bool operator==(const A &rhs) const {
        return b == rhs.b;
    }
};

int main() {
    cout << std::boolalpha << is_incrementable<int>() << endl;
    cout << is_incrementable<string>() << endl;
    cout << is_incrementable<A>() << endl;
    return 0;
}
```



```
stvn@DESKTOP-S2SA480:/mnt/c/Users/stvn/Documents/stvn/cpp/Assignment7$ g++ main.cpp -o main -std=c++17
stvn@DESKTOP-S2SA480:/mnt/c/Users/stvn/Documents/stvn/cpp/Assignment7$ ./main
true
false
true
stvn@DESKTOP-S2SA480:/mnt/c/Users/stvn/Documents/stvn/cpp/Assignment7$
```

Part 4 - Difficulties & Solutions

1. How to use `declval` ++

    ```
    ((++std::declval<T&>())++)
    ```