

Workshop 11

Function Templates

In this workshop, you will code a function template for validating a value between a minimum and maximum range.

LEARNING OUTCOMES

Upon successful completion of this workshop, you will have demonstrated to abilities to:

- code a function template
- code explicit specialization for a function template
- reflect on what you have learned from this workshop

SUBMISSION POLICY

The “*in-lab*” section is to be completed **during your assigned lab section**. It is to be completed and submitted by the end of the workshop. If you do not attend the workshop, you can submit the “*in-lab*” section along with your “*at-home*” section (a 20% late deduction will be assessed). The “*at-home*” portion of the lab is **due the day before your next scheduled workshop**.

All your work (all the files you create or modify) must contain your name, Seneca email and student number.

You are responsible to regularly backup your work.

IN-LAB (80%)

Create a template function named `validate(...)` that accepts five arguments (a minimum, a maximum, an array of test-values, number of elements in the array, an array of Boolean for validation results) and returns a `bool` that is true only if all the test-value elements are between the minimum and maximum arguments, and false otherwise.

Also as the validation test is being done on each element, the result of validation is saved in the Boolean array so the caller program can determine exactly which element if the test-values array is invalid.

Make sure the function is designed in a way that all test objects (min, max and test elements) are not copied or modified in the function.

Store a template for your function in a header file named `validate.h`.

CLIENT MODULE

The main program that uses your implementation is listed below:

```
#include <iostream>
#include "Car.h"
#include "Employee.h"
#include "Student.h"
#include "validate.h"

using namespace std;
using namespace sict;

int main()
{
    int i;
    bool v[10];

    Student S[6] = { { 2345, "Lisa Simpson" }, { 12345, "Bart Simpson" },
                     { 4567, "Ralph Wiggum" }, { 56789, "Milhouse Van Houten" },
                     { 67890, "Todd Flanders" }, { 34567, "Nelson Muntz" } };
    Employee E[6] = { Employee(213456, "Carl Carlson", 62344.56), Employee(122345,
"Mindy Simmons", 65432.44),
                     Employee(435678, "Lenny Leonard", 43213.22), Employee(546789,
"Waylon Smithers", 654321.55),
                     Employee(657890, "Frank Grimes", 34567.88), Employee(364567,
"Homer Simpson", 55432.11) };
    int vals[10] = { 2, 6, 4, 67, 4, 6, 7, 5, 4, 6 };
    char cvals[10] = { 'A', 'e', 'B', 'd', 'e', 'G', 'H', 'l', 'p', 'x' };
    Car C[5] = { Car("DEFGHI", "Tesla Model S"), Car("BBCDEF", "BMW 320"),
                Car("CDEFG", "Ford Festiva"), Car("BCDEFG", "Ford Festiva"),
                Car("AFGHIJ", "Nissan Maxima") };

    if (!validate(Student(11111), Student(99999), S, 6, v))
    {
        cout << "These students have invalid student ids:" << endl;
        for (i = 0; i < 5; i++)
        {
            if (!v[i]) cout << S[i] << endl;
        }
    }
    else
    {
        cout << "All students have valid students ids" << endl;
    }

    if (!validate(Employee(111111), Employee(999999), E, 6, v))
    {
        cout << "These employee have invalid employee ids:" << endl;
        for (i = 0; i < 5; i++)
        {
```

```

        if (!v[i])
            cout << S[i] << endl;
    }
}
else
{
    cout << "All employee have valid employee ids" << endl;
}

if (!validate(3, 7, vals, 10, v))
{
    cout << "These integer values are invalid: " << endl;
    for (i = 0; i < 10; i++)
    {
        if (!v[i])
            cout << vals[i] << endl;
    }
}
else
{
    cout << "All integer values are valid" << endl;
}

if (!validate(Car("BBBBBB"), Car("ZZZZZ"), C, 5, v))
{
    cout << "These cars have invalid license plates:" << endl;
    for (i = 0; i < 5; i++)
    {
        if (!v[i])
            cout << C[i] << endl;
    }
}
else
{
    cout << "All cars have valid license plates" << endl;
}

if (!validate('B', 'P', cvals, 10, v))
{
    cout << "These character values are invalid: " << endl;
    for (i = 0; i < 10; i++)
    {
        if (!v[i])
            cout << cvals[i] << endl;
    }
}
else
{
    cout << "All character values are valid" << endl;
}

return 0;
}

```

```
These students have invalid student ids:
2345 Lisa Simpson
4567 Ralph Wiggum
All employee have valid employee ids
These integer values are invalid:
2
67
These cars have invalid license plates:
AFGHIJ Nissan Maxima
These character values are invalid:
A
e
d
e
l
p
x
```

IN-LAB SUBMISSION

To submit the in-lab section, demonstrate execution of your program with the exact output as the example above.

If not on matrix already, upload all the source files and your `validate.h` to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account (and follow the instructions):

```
~profname.proflastname/submit w11_in_lab <ENTER>
```

AT-HOME - TEMPLATE SPECIALIZATION (20%)

Looking at the output of *in-lab* section closely, you will notice that for character validation, although the test was done as `validate('B', 'P', cvals, 10, v)`, characters like `'e'`, `'l'` and `'p'` are tagged as invalid because they are lower case. Add a template specialization to your validation template so the character validation is **NOT** case sensitive.

Also looking at the array of `Cars`, you will notice that `Ford Festiva` has a 5 letter license plate that is tagged as valid. Add a template specialization to your validation template so when `Cars` are validated, any license plate that is **NOT** 6 letters is tagged as invalid.

After doing this, the program should generate the following output:

```
These students have invalid student ids:
2345 Lisa Simpson
4567 Ralph Wiggum
All employee have valid employee ids
These integer values are invalid:
2
67
These cars have invalid license plates:
CDEFG Ford Festiva
AFGHIJ Nissan Maxima
These character values are invalid:
A
X
```

Please provide brief answers to the following questions in a text file named `reflect.txt`.

1. Explain why templates should exist entirely in header files and **NOT** *.cpp files.
2. Name different types of polymorphism and show an example for each of the types from this workshop.

SUBMISSION

If not on matrix already, upload all the files (including `validate.h` and `reflect.txt`) to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account (and follow the instructions):

```
~profname.proflastname/submit w11_at_home <ENTER>
```