PIC 10C Section 1 - Homework # 7 (due Friday, Sunday, May 15, by 11:59 pm)

You should upload each .cpp and/or .h file separately and submit them to CCLE before the due date/time! Your work will otherwise not be considered for grading. Do not submit a zipped up folder or any other type of file besides .h and .cpp.

Be sure you upload files with the precise name that you used in your editor environment otherwise there may be linker and other errors when your homeworks are compiled on a different machine.

Also be sure your code compiles and renders the correct output on g++ with -O3 optimization on the C++20 Standard!!!

SPECIALIZAING A SET

In this homework, you will get to write a **simple_set** class to store values uniquely. The class will be specialized partially and completely. Before you have a **PTTD** (Post Traumatic Tree Disorder) flashback from the red-black tree assignment, rest assured, you will *not* be writing such a structure. This homework is supposed to very gentle practice in template specialization. For this homework, the **only header file you are allowed to use** is:

- **vector**

You will submit **SimpleSet.h**. Please refer to the syllabus for how the work will be graded: note that more than half of the marks come from good coding practices and code documentation.

You can assume the code will be compiled with a **makefile**:

```
hw7: main.cpp SimpleSet.h
    g++-8 -std=c++2a -O3 main.cpp -o hw7
```

for a **main.cpp** that could differ from the example given!

Our focus is upon our sets storing values uniquely. We don't care about their order! Within the **pic10c** namespace, you will write:

- a templated class, **simple_set**, templated by a type **T**. The class should
  - store an **std::vector<T>** for its data;
  - have a default constructor to begin empty;
  - have an **insert** function to insert **T**-values into the vector, unless the value is already there in which case the values are not added;
  - have an **erase** function, accepting a **T**-value and removing that value from the data if found;
  - have a **find** function, accepting a **T**-value and returning **true** if that value is found in the set; and
  - a **size** function to return the number of elements stored in the set.
- a *partial specialization* of **simple_set** for pointers to **T**, **T***. This partial specialization should
  - store an **std::vector<T*>** for its data;
  - have a default constructor to begin empty;
  - have an **insert** function to insert **T***-values into the vector, unless the value *pointed to by the T** is already there in which case the data is not added;
  - have an **erase** function, accepting a **T**-value and removing any such **T*** that points to the value if found;
  - have a **find** function, accepting a **T**-value and returning **true** if a pointer pointing to that value is found in the set; and
  - a **size** function to return the number of elements stored in the set.
- an explicit specialization of **simple_set** for the **bool** type. This explicit specialization should:
  - have two member variables **has_true** and **has_false** of type **bool** to indicate whether the set stores **true** and **false**, respectively;
  - have a default constructor to begin empty;
  - have an **insert** function to insert **bool** values by updating the **has_true** and **has_false** values appropriately;
  - have an **erase** function, accepting a **bool** and removing that value if found by updating the **has_true** and **has_false** values appropriately;
  - have a **find** function, accepting a **bool** and returning **true** if that **bool** is found in the set; and

  - a **size** function to return the number of elements stored in the set (can be 0, 1, or 2 only!!!).

A test case, code and output, are provided below.

```
#include <iostream>
#include <string_view>
#include "SimpleSet.h"

int main()
{
    constexpr std::string_view line("---------------\n");

    pic10c::simple_set<float> floats;
    floats.insert(2.2f);
    floats.insert(4.4f);
    floats.insert(4.4f);
    floats.insert(-0.98333f);

    std::cout << std::boolalpha;

    std::cout << "floats:\n";
    std::cout << floats.find(4.4f) << '\n';
    std::cout << floats.find(19.f) << '\n';
    std::cout << floats.size() << '\n';

    std::cout << line;

    pic10c::simple_set<int*> ints;
    int i1 = 1, i2 = 2, i3 = 3;
    int j1 = 1, j2 = 2;
    ints.insert(&i1);
    ints.insert(&j1);
    ints.insert(&i2);
    ints.insert(&j2);
    ints.insert(&i3);

    ints.erase(1);

    std::cout << "ints:\n";
    std::cout << ints.find(1) << '\n';
    std::cout << ints.find(3) << '\n';
    std::cout << ints.size() << '\n';
    std::cout << line;
    pic10c::simple_set<bool> bools;
    bools.insert(true);
    bools.insert(true);
    bools.insert(false);

    bools.erase(true);

    std::cout << "bools:\n";
    std::cout << bools.find(true) << '\n';
    std::cout << bools.find(false) << '\n';
    std::cout << bools.size() << '\n';
    std::cout << line;

    return 0;
}
```

An example of the output is:

```
floats:
true
false
3
---------------
ints:
false
true
2
---------------
bools:
false
true
1
---------------
```