CMPUT 350 Lab 5 Exercise Problems

Rules:

- You can use all course material and man pages, but no other information such as web pages, books, or written notes. Using other information sources during the exercise constitues cheating.
- Your programs must compile without warning using

  ```
  g++ -Wall -Wextra -Wconversion -Wsign-conversion -O -g -std=c++17 ...
  ```

  In case there are compiler warnings or errors, marks will be deducted.
- Test your programs with different values. For now, the speed of your program is irrelevant. So don't spend time on optimization
- You must check for the appropriate preconditions/postconditions. Your program shouldn't crash or have undefined behaviour (**hint**: use asserts)!
- Your programs must be well structured and documented. Use ctrl-x t in Emacs to pretty-print it. Marks are assigned to functionality, program appearance, and comments.
- In case your program hangs, use ctrl-c to terminate it.
- Remember that you need to include the appropriate header files. To find out which ones you need for specific functions such as printf, use then man command.

Submit your solution files `max.cpp point.cpp` on eClass under "Lab 5 / Submission".

Important: Submit often (the system will only accept solutions submitted before 16:50)

---

1. [20 marks] Function templates

In file `max.cpp`:

a) Write function

```
size_t index_of_max(const int *A, size_t n)
```

which is supposed to return the smallest index of a maximum element in array `A` that holds `n` elements. Only use `<` to compare elements

b) Write function template `index_of_max` that generalizes above function to work with any element type `T`. You can only assume that `T` supports operator `<` , and nothing else (**in particular, the CC or AO doesn't have to be supported**)

c) Specialize (or overload) the template function for arrays of element types `char *` and `const char *` so that the smallest index of a maximum string in lexicographical ordering is returned (hint: use strcmp instead of `<`)

d) Write function `void print(const int *a, size_t n);` that prints all array elements to cout using a single space as delimiter

e) Write function template print that works like the function in (d) for arbitrary element types `T`

f) In `main()` write test code that populates a few C-arrays of different type, prints them using above print function template, prints the result of the `index_of_max` function call, and complains if a test failed. **Hint**: When testing `const char*` versus `char*`, recall that string literals are have the type `const char *`. How can we explicitly declare string literals to be of type `char*`? We shouldn't do this in practice ... WHY?

---

2. [20 marks] Function templates

In file `point.cpp`:

a) Write struct template `Point` that stores two coordinates `x,y` of arbitrary type `T` and has a constructor that takes two parameters `(x0,y0)` with which `x,y` are initialized, respectively. Their default values must be `T{}` (representing origin coordinate (0)).

A print member function also needs to be implemented that writes `(x,y)` to cout.

```
void print() const ...
```

Finally, struct member `operator==` needs to be implemented that returns true iff the rhs point's coordinates are equal to `x,y`, respectively.

b) Write function template sum that accepts a const C-array of `Point` elements, array size `int n` and returns a `Point` of the same type representing the sum of all input `Points` (component-wise)

c) Write function template find that, like in (b), accepts a const C-array of `Points` and `size n`, but also a `Point p` of the same type and returns the smallest index for which `p` is equal to an array element, or -1 otherwise

d) Write test code for (b) and (c) in `main()`