Project 6: Generic Programming

Due: Monday June 18, 11:59 pm

1 Assignment Overview

This project focuses on generic functions. You will write functions that can operate on data structures that hold various types. You will also use exisiting generic functions to help you do this.

2 Assignment Deliverables

You must turn in completed versions of the following files:

- proj06/generic.cpp

Be sure to use the specified file name and to submit your files for grading via **Mimir** before the project deadline.

3 Background

The `<algorithm>` library contains a number of generic functions can operate on a wide variety of data structures. These functions include both non-modifying operations, such as `all_of`, `find`, and `equal`, and modifying operations, such as `copy`, `move`, and `reverse`. Unfortunatly, there are plenty of other common functions that are not included.

Additionally, the `accumulate` function from the `<numeric>` library is very powerful, allowing you to do many computation, but may be more complicated than necessary to write.

When working with lists of numbers it is often necessary to find their sum or product. You will compute these using the `Sum` and `Product` functions. You will also write a `SumBy` function that takes sequences of non-numeric items, such as `string`s, and converts them into numbers first. This will require a function parameter that handles the conversion between types.

The `<algorithm>` already has `min_element` and `max_element` functions which return pointers to the smallest and biggest items in the sequence. There are variants that receive a comparison function that directly compares two elements. In Python, the corresponding functions instead allow a function to define a key for each item that determines their relative order. You will create `MinBy` and `MaxBy` functions that find the min and max item using the Pythonic substitution method.

C++ already has a `remove_if` function that remove all of the items from a sequence that match a given predicate. Sometimes it is more natural to specify which items *should* be in the list than those that *should not* be. You will write a `KeepIf` function that does this.

Finally, in Python, the `join` function can be used to merge a list of strings into a sequence that are separated by a single deliminator (such as a comma or space). You will create a similar `Join` function.

4 Assignment Specifications

In this and in future projects we will provide *exactly* our function specifications: the function name, its return type, its arguments, and each arguments type. The functions will be tested individually in Mimir using these exact function specifications. If you do not follow the function specifications, these independent tests of your functions will fail. Do not change the function declarations!

What you test on Mimir is a file that contains only the functions. You do not turn in a main program. We can test the functions individually on Mimir. However, you should write your own main program to test your functions separate from Mimir. It is more flexible and you can debug more easily.

For this project you are not allowed to create any additional data structures, such as `vector`. **You are also not allowed to use any `for` or `while` loops**. You *are* allowed to use other fixed-size variables. Your functions must be generic and will be tested on multiple data types.

Each of the functions will expect two iterators denoting a semi-open range. Some of these will be `InputIterator`s which can only be read once and then must be advanced to the next element. Others will expect `ForwardIterator`s, which can be read any number of times before being advanced. Neither of these iterators allow reverse traversal.

Additionally, some functions will take a `Function` or `Predicate` (a predicate is a function that returns a Boolean). These functions will receive one of the iterator items as input.

function `Sum`
params: front : InputIterator, back : InputIterator
returns: The sum of the sequence

function `Product`
params: front : InputIterator, back : InputIterator
returns: The product of the sequence

function `SumBy`
params: front : InputIterator, back : InputIterator, f : Function
returns: The sum of the sequence

function `ProductBy`
params: front : InputIterator, back : InputIterator, f : Function
returns: The product of the sequence

function `MinBy`
params: front : ForwardIterator, back : ForwardIterator, f : Function
returns: An iterator to the smallest value in range or *last* if the range is empty.

function `MaxBy`
params: front : ForwardIterator, back : ForwardIterator, f : Function
returns: An iterator to the largest value in range or *last* if the range is empty.

function `KeepIf`
params: front : ForwardIterator, back : ForwardIterator, f : Predicate
returns: An iterator that follows the last element kept.

function `Join`
params: front : InputIterator, back : InputIterator, sep : const string&
returns: A formatted string

5 Assignment Notes

- You will receive no points if your solution does not compile on Mimir.
- Points will be deducted if your solution has any compiler warnings.
- Points will be deducted if you include `using namespace std`.
- You may not use a `for` or `while` loop.

- You *do not* need to check for bad input values for this project. In future projects we will explicitly indicate the errors we are looking for.
- You *may* use the `<algorithm>` library. Some functions may be easier to frame as a variation on an existing function.
- Remember that the additive identity is 0 and the multiplicative identity is 1.
- Most of the functions can be written as special cases of existing functions such as `accumulate`.
- `auto` may be helpful to avoid writing some complicated types.
- You can get the return type of function pointer `f` by using `decltype(f(args))` and the value type of an iterator with `typename std::iterator_traits<InputIterator>::value_type`. Yes, they are different.
- A stub of `SumBy` is available on D2L to help you get started.
- The return types for `Sum` and `Product` are determined by the types of the items passed in. Make sure that you return the correct type.