$Id: cse111-2020q4-final.mm,v 1.73 2020-12-09 15:17:29-08 - - $

Assume all necessary #include directives and do not show #include directives in your answers.

You many not asssume #include <algorithm>. Do not make use of any functions defined in <algorithm>. For such questions, show the actual code, and do not write a call to a function in <algorithm>.

Use proper indentation. Points will be deducted for messy or unreadable answers.

Question 1. **[[4✔]]** <PRE>

Write code for an inner product template function. Its single template argument is a forward iterator type. It has four function arguments :

```
   double innerp (itor begin1, itor end1,
                  itor begin2, itor end2)
```

An inner product multiplies corresponding elements of each range, and returns the sum of these products. Assume the iterators point at doubles or things that can be implicitly converted to double. Throw domain_ error if the ranges are of different lengths. Since the iterators are forward, you may not subtract them.

Example: If v={1,2,3} and w={4,5,6}
then innerp(v,w) = {1*4 + 2*5 + 3*6}.

Question 2. **[[3✔]]** <PRE>

Write a template function to reverse a range of elements. In other words swap the first with the last, the second with the second last, etc. The range may have an even or an odd number of elements, or in may be empty.

Assume #include <utility>. To exchange items, use:

```
   template <class T>
   void swap (T& a, T& b)
```

The function has one template iterator type. Its function arguments are a begin and an end iterator, both of which are bidirectional.

Bidirectional means you can use the ++ and -- operators on these iterators (either prefix or postfix), as needed. You may not use subscripts with the iterators.

Question 3. **[[4✔]]** <PRE>

Operator++.

(a) Show the prototypes for operator++, both prefix and postfix versions, as they would appear as members declared inside of class foo.

(b) Show the prototype for the prefix operator++ which increments an object of class bar, but which is *NOT* a member of bar. It’s prototype appears outside of the definition of class bar.

(c) Write the complete function, including the code inside postfix operator++ which increments a bar. This calls the prefix operator++, as part of the implementation.

Question 4. **[[1✔]]** <PRE>

Translate the following statement into an equivalent for-loop, that uses two semi-colons (;) instead of a colon (:).

```
   for (auto& i: c) f(i);
```

Question 5. **[[6✔]]** <PRE>

Write a function equal which returns a bool, which compares two ranges of elements. It returns true if all elements of both ranges are equal and both ranges are of the same length.

It has three template parameters: an iterator type for the first range, an iterator type for the second range, and equal_to, which is the type of a function object that performs the comparisons.

It has five function arguments: Begin and end iterators of the first iterator type, begin and end iterators for the second iterator type, and an equal_to comparison function object which defaults to the default constructor for equal_to.

```
   bool equal (itor1 begin1, itor1 end1,
               itor2 begin2, itor2 end2,
               equal_to equal = equal_to()) {
```

Assume only forward iterators. You may not subtract iterators to determine the number of elements in each range.

Question 6. **[[4✔]]** <PRE>

Write a function fold. It has three template parameters: a forward iterator type, a unit type, a binary function object type.

It has four function parameters: begin and end itertors, a unit passed in by value which is used to accumulate a result, and a binary function object whose arguments are of the same type as the unit, and which returns a value which is the same type as the unit.

For example, if we assume:

```
   double add (double a, double b) { return a + b; }
   double mul (double a, double b) { return a * b; }
```

then the following statements will produce the sum and product of the contents of a collection v:

```
   double sum = fold (v.begin(), v.end(), 0.0, add);
   double prod = fold (v.begin(), v.end(), 1.0, mul);
```

Question 7. **[[2✔]]** <PRE>

Iteration, Part 1.

Given the following class declaration of a trivial implementation of a vector of ints:

```
   class ivec {
      private:
         int* vp;
         size_t siz;
      public:
         class iterator;
         iterator begin();
         iterator end();
   };
```

Write code to implement begin and end as they would appear outside of the class ivec, in the implementation file.

Continued in Iteration, part 2.

Question 8. **[[6✔]]** <PRE>

Iteration, Part 2.
Continued from Iteration, Part 1.

Show the implemntation of ivec::iterator with all necessary functions declared inline. Write the iterator as a complete implementation.

Show only those members of ivec::iterator that are necessary for the following main function to compile:

```
   int main() {
      ivec v;
      int s;
      for (auto i: v) s += i;
   }
```

Question 9. **[[5✔]]** <PRE>

Given the following struct declaration, and the prototype for a find function, write the complete function body which performs a binary search on the tree passed in as an argument. You may only compare values by using the less function object. Smaller objects are on the left side and larger objects are on the right side. Return a pointer to the node containing the item.

```
   template <typename T>
   struct tree {
      T value;
      tree* left;
      tree* right;
   };

   template <typename T, typename lesst>
   tree<T>* find (tree<T>* t, T item, lesst less) {
```

Question 10. **[[2✔]]** <PRE>

Inheritance, part 1.
This and the following questions labelled "Inheritance" are all part of the same question.

Define a base class called expr with the following functions. It has no fields.

- * Abstract function eval which returns a double and is const.
- * Abstract function print which returns void and is const. It has one argument which is an ostream&.
- * Destructor.

Operator<< is a non-member friend.

Question 11. **[[4✔]]** <PRE>

Inheritance, part 2.

Class number is derived from class expr.

There is a double field called value. By default it is initialized to 0.0.

For the function members:

- * The ctor is both a default ctor and a ctor with one argument, host default argument value is 0.0. This ctor may also be used to implicitly convert a double into an object of class number.
- * Eval just returns the value field.
- * Print just prints the number in the default format.

Question 12. **[[9✔]]** <PRE>

Inheritance, part 3.

Class adder is derived from class expr. It has fields called left and right both of which are of type expr* (raw pointers to exprs).

For the function members:

- * A ctor with two arguments that initializes the left and right fields.
- * Eval returns the sum of evaluation the two subtrees.
- * Print prints a left paren, the entire left subtree (recursively), a plus (+) sign, the entire right subtree (recursively), and a right paren. Operator<< can be used to perform the recursion in printing.
- * Destructor.

SCORE-TOTAL=50