DTS Lab 3  
singly linked list and iterator

Contents

[Objective 2](#_Toc358034552)

[Prototypes 2](#_Toc358034553)

[Other Code 4](#_Toc358034554)

[Desired Output 6](#_Toc358034555)

[Submission 7](#_Toc358034556)

# Objective

You will implement a templated, single-ended, singly-linked list class named SLList and an iterator class named SLLIter to access the SLList.

Place all your code for both classes in a file named *SLList.h*.

# Prototypes

The SLList class will have the following public interface:

/////////////////////////////////////////////////////////////////////////////  
// Function : Constructor  
// Notes : constructs an empty list  
/////////////////////////////////////////////////////////////////////////////  
SLList()  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : Destructor  
// Notes : Destroys the list  
/////////////////////////////////////////////////////////////////////////////  
~SLList()  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : Assignment Operator  
/////////////////////////////////////////////////////////////////////////////  
SLList<Type>& operator=(const SLList<Type>& that)  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : Copy Constructor  
/////////////////////////////////////////////////////////////////////////////  
SLList(const SLList<Type>& that)  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : addHead  
// Parameters : v - the item to add to the list  
/////////////////////////////////////////////////////////////////////////////  
void addHead(const Type& v)  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : clear  
// Notes : clears the list, freeing any dynamic memory  
/////////////////////////////////////////////////////////////////////////////  
void clear()

/////////////////////////////////////////////////////////////////////////////  
// Function : insert  
// Parameters : index - an iterator to the location to insert at  
// v - the item to insert  
// Notes : do nothing on a bad iterator  
/////////////////////////////////////////////////////////////////////////////  
void insert(SLLIter<Type>& index, const Type& v)  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : remove  
// Parameters : index - an iterator to the location to remove from  
// Notes : do nothing on a bad iterator  
/////////////////////////////////////////////////////////////////////////////  
void remove(SLLIter<Type>& index)  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : size  
// Return : the number of items stored in the linked list.  
/////////////////////////////////////////////////////////////////////////////  
inline unsigned int size() const;

The SLLIter class will have the following public interface:

/////////////////////////////////////////////////////////////////////////////  
// Function : Constructor  
// Parameters : listToIterate - the list to iterate  
/////////////////////////////////////////////////////////////////////////////  
SLLIter(SLList<Type>& listToIterate)  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : begin  
// Notes : moves the iterator to the head of the list  
/////////////////////////////////////////////////////////////////////////////  
void begin()  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : end  
// Notes : returns true if we are at the end of the list, false otherwise  
/////////////////////////////////////////////////////////////////////////////  
bool end() const  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : operator++  
// Notes : move the iterator forward one node  
/////////////////////////////////////////////////////////////////////////////  
SLLIter<Type>& operator++()  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : current  
// Notes : returns the item at the current iterator location  
/////////////////////////////////////////////////////////////////////////////  
Type& current() const

# Other Code

The SLList needs access to the SLLIter’s private data members for the insert and remove functions and the SLLIter needs access to the SLList’s private data members for the begin function and in order to use the privately nested Node structure. To give these two classes access to each other’s private members you will need to declare them as mutual friends:

// forward declaration  
template<typename Type> class SLLIter;  
  
// class SLList  
template<typename Type> class SLList  
{  
 // the iterator is the list’s friend  
 friend class SLLIter<Type>;  
  
 // add members/methods here…  
   
};  
  
// class SLLIter  
template<typename Type> class SLLIter  
{  
 // the list is the iterator’s friend  
 friend class SLList<Type>;  
  
 // add members/methods here…  
};

Now that you have the relationship between the SLList and SLLIter classes established, the SLLIter can access the Node structure that is declared in the private section of the SLList class. You will need to declare at least one pointer to a Node for the SLLIter to function properly. Your first instinct might be to simply declare it like this:

Node \* c;

The problem with this is that Visual Studio will give you an error saying it doesn’t know what a Node is. This is because the node is nested inside the SLList class. Your next though might be to scope it in as follows:

SLList<Type>::Node \* c;

Now, it is scoped in and the compiler knows what a Node is, but it still gives you an error, something about a dependent name not being a type. You can tell the compiler that it is, indeed, a type by using the typename keyword as follows:

typename SLList<Type>::Node \* c;

With this syntax we can declare as many pointers to Nodes as we need.

# Desired Output

Compile and run your code with the DTSLab3.cpp file provided via FSO. Your console output should match the following block identically:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*\* LAB 3: \*\*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
\*\*\* TEST 1 \*\*\*  
Adding Nodes to the front...  
Size: 0 List Contents { }  
Size: 1 List Contents { Rules }  
Size: 2 List Contents { Sail Rules }  
Size: 3 List Contents { Full Sail Rules }  
Now Clearing...  
Size: 0 List Contents { }  
  
\*\*\* TEST 2 \*\*\*  
Inserting and removing from the list...  
Size: 6 List Contents { Pod Six is full of jerks }  
Removing 'Pod'  
Size: 5 List Contents { Six is full of jerks }  
Removing 'full'  
Size: 4 List Contents { Six is of jerks }  
Removing 'jerks'  
Size: 3 List Contents { Six is of }  
Inserting 'Pod'  
Size: 4 List Contents { Pod Six is of }  
Inserting 'full'  
Size: 5 List Contents { Pod Six is full of }  
Removing 'of'  
Size: 4 List Contents { Pod Six is full }  
  
\*\*\* TEST 3 \*\*\*  
Using operator=...  
Size: 3 List Contents { Full Sail Rules }  
Size: 3 List Contents { Full Sail Rules }  
Size: 0 List Contents { }  
  
\*\*\* Test 4 \*\*\*  
Stress testing the List (and possibly your trouble shooting)  
Size: 1 List Contents { is it safe?? }  
Size: 0 List Contents { }  
Size: 0 List Contents { }  
Size: 1 List Contents { still here }  
Size: 1 List Contents { should be here }  
Size: 7 List Contents { 24 20 16 12 8 4 0 }  
Size: 0 List Contents { }  
Size: 0 List Contents { }

# Submission

To submit the lab assignment:

1. Clean, build, and run DTSLab3.cpp with your SLList.h file in Visual Studio (debug mode).
   1. clear up any warnings you encounter.
   2. verify that your output is correct by comparing it to the lab document's Desired Output section, line-by-line.
   3. ensure there are no memory leaks.
2. On your desktop, create a new folder with your name in the following format:
   1. your last name
   2. a comma
   3. a single space
   4. your first name  
      **\* Appropriate capitalization for proper names should be used.**  
      Suitable examples include : "Pollack, Joey"; "De La Paz, Christhian"; "Tjarks, Matthew".
3. Copy your 'SLList.h' file into the folder that you created in step 2. This is the only file I need and should therefore be the only file you submit.
4. Right-click on the folder and select 'send to->compressed (zipped) folder'.
5. Submit the compressed folder via FSO.