Programming Assignment 1

Overview

In Programming Assignment 1 you will implement a templated version of a singly linked list. You will implement several of the features that are provided by the standard library `std::forward_list` in a class called `Forward_list`.

This programming assignment will build on what you have done in Tutorial 3 on linked lists and Tutorial 7 on sorting. Key new features that you will implement in this assignment include:

- maintain the size of the list
- pop_front() delete the first element of the list
- construct a new list from an existing one (copy constructor) or from an initializer list like {1,3,5,7}.
- sort the list. We have provided code to do merge sort, but left to you to implement the key needed subroutines of merging two sorted lists and splitting a list in half.

The Code

You are provided with two files:

- `my_forward_list.hpp`: This contains all the function declarations and instructions about which ones you need to implement. **All your implementations should be written in this file.** This is the only file we will use for testing. You can add helper functions as needed but do not change the signatures of the provided functions.
- `main.cpp`: this includes a `main` function. You may write some code to test your function implementations for `Forward_list`. This file will not be marked so you can do anything you like with it -- just ensure it does not cause any error. When the "*run*" button is pressed, it will compile and run `main.cpp`.

Specifically, the functions you need to implement are

- Linked list copy constructor (initialise a linked list from another linked list)
- Linked list constructor from an initializer_list (initialise a linked list with e.g. {1,2,3,4})
- push_front
- pop_front
- front
- display (print out the list)
- merge and split, two helper functions to implement merge sort.

The code in `my_forward_list.hpp`has been commented to explain the purpose of each function. Remember to read over all the code before starting.

When the "*mark*" button is pressed, your code in `my_forward_list.hpp` will be run against the tests in Ed. The testing code can only mark your code when your code does not cause a program crash. If you get any error during compiling or your program cannot stop by itself, make sure you fix that problem first!

The lectures may not readily give you the solution, and you may have to develop your own solutions to challenges that arise in the implementation. Don't hesitate to ask questions if you are unclear about the task requirements.

Testing

There are 10 tests that are run on your code.

1. Test 1 is test_size_initialization. It just checks that your code compiles with templated types and that the size of an empty list is initialised to 0.
2. Test 2 is test_push_pop. It checks that push_front, front, and pop_front are working and that the size_ variable is properly updated under these operations.
3. Test 3 is test_copy_constructor. It tests your implementation of the copy constructor.
4. Test 4 is test_initializer_list. It tests your constructor from an initializer list of ints.
5. Test 5 is test_initializer_list_str. It tests your constructor from an initializer list of strings.
6. Test 6 is test_merge. It tests your merge function.
7. Test 7 is test_merge_edge_cases. It tests some edge cases with respect to the merge function.
8. Test 8 is test_split. It tests your split function.
9. Test 9 is test_split_and_merge. It tests that your split and merge function work together to do one step of merge sort.
10. Test 10 is test_null_copy. It tests an edge case of your copy constructor, to copy an empty list.

Note that the split function uses your copy constructor to create its return value. Thus it is good to get your copy constructor working before doing the tests involving split.

In the tests.hpp file we also provide test_sort which tests that your merge and split when called by the merge_sort function actually sort the list. However, this test is for your enjoyment only, and is not part of the marked tests.

Marking

The assignment will be marked against three components:

Functionality (18%) will be marked automatically when you press the "*mark*" button and will be verified by the teaching team.

Design (5%) will be marked by your tutor, who will evaluate the efficiency of your solutions and may ask questions regarding your code. It will be marked qualitatively against the following rubric:

- Pass The code shows a basic understanding of how to employ data structures to achieve a goal. The design should avoid unnecessary data structures and should make reasonable use of iteration and recursion when appropriate.
- Credit The design shows a solid understanding of data structures and demonstrates effective use of control structures to achieve the program's goals.
- Distinction The design shows a high degree of understanding of how to use data structures to achieve a goal efficiently and demonstrates some evidence that the design does not use unnecessary resources. The design should be clean and efficient.
- High Distinction The design demonstrates a high degree of understanding of data structures and how to efficiently employ them to build algorithms that not only meet technical goals but also support maintenance and future development.

Style (2%) will also be marked by your tutor. It will be marked qualitatively against the following rubric:

- Pass The code mostly uses some formatting standards and is somewhat readable.
- Credit The code adheres well to a formatting standard and variables are well named.
- Distinction At least as well formatted as for Credit standards, along with sufficient inline commenting to explain the code.
- High Distinction Excellent formatting and variable naming. Excellent, judiciously employed comments that explain the code without just repeating the code.

Those who have in-person tutorials can demonstrate your code and get immediate feedback when you are being marked for *design* and *style* in weeks 9 and 10. For those who have an online tutorial, or if you cannot attend tutorials in person, you should **provide a video (2-3 mins length)** explaining the merits (for *design* and *style*) of your code for Programming Assignment 1. Your tutor will give your marks for *design* and *style* based on your code and video explanation. You may share your video through Google Drive, OneDrive, YouTube, or in any other form with your tutor -- just make sure your tutor can access it.

Submission

You must submit your code by pressing the "*mark*" button on Ed. No other forms of submission will be accepted. You are welcome to develop your code elsewhere, but remember it must compile and run on Ed. The underlying g++ compiler in Ed is set to C++17 for this assessment. Usually, code that compiles with clang++ should also work with g++.

You may submit as many times as you like before the due date.

Due Date

Code submission is due by 24 April 2022 at 11:59 PM. A 10% penalty applies to all late submissions made within a week after the due date. No code submissions will be accepted after 1 May 2022.

In-class code demonstration is due in Week 9-10 tutorials. Alternatively, video submission should be made before your Week 10 Tutorial. No late demonstration or video submission will be accepted.

Plagiarism Checking

All code will be checked for student misconduct and plagiarism using specialised code similarity detection software.