ECS 34: Programming Assignment #7

Instructor: Aaron Kaloti

Fall 2020

Contents

1 Changelog

You should always refer to the latest version of this document.

- v.1: Initial version.
- v.2:
  - Autograder details.
  - Clarified that you need to actually implement a linked list (don't use `std::list` or `std::forward_list`).

2 General Submission Details

Partnering on this assignment is prohibited. If you have not already, you should read the section on academic misconduct in the syllabus.

This assignment is due the night of Tuesday, December 15. Gradescope will say 12:30 AM on Wednesday, December 16, due to the "grace period" (as described in the syllabus). *Be careful about relying on the grace period for extra time; this could be risky.*

You should use the `-Wall`, `-Werror`, and `-std=c++11` flags when compiling. The autograder will use these flags when it compiles your code.

3 Grading Breakdown

As stated in the syllabus, this assignment is worth 4% of the final grade. Below is the breakdown of each part.

- Part #1: 2% (20 points)
- Part #2: 2% (20 points)
- Part #3: 5% (50 points)

Note on extra credit: I will set the points cutoff at 40 points, which means that it is possible to earn extra credit on this assignment. [1]

4 Submitting on Gradescope

You should only submit `list.hpp`, `list.inl`, and the two shell scripts. You have infinite submissions until the deadline.

During the 10/02 lecture, I talked about how to change the active submission, just in case that is something that you find yourself wanting to do.

4.1 Regarding Autograder

Your output must match mine *exactly*.

There is a description about how to interpret some of the autograder error messages in the directions for the first two programming assignments. I will not repeat that description here.

4.1.1 Visible Test Cases' Inputs

Part #1:

Case #1:

```
./cp_mult.sh hi
echo $?
```

Case #2:

```
mkdir d1 d2 d3 d4
echo -e "123\n45\n78" > vals2.txt
./cp_mult.sh vals2.txt d4 d1 d3 d2
echo -e "=== retval ==="
echo $?
echo -e "=== d4/vals2.txt ==="
cat d4/vals2.txt
echo -e "=== d1/vals2.txt ==="
cat d1/vals2.txt
echo -e "=== d3/vals2.txt ==="
cat d3/vals2.txt
echo -e "=== d2/vals2.txt ==="
cat d2/vals2.txt
```

Case #3:

```
mkdir foo goo bar
echo -e "blah\nblah\nblah\nblah" > foo/vals3.txt
./cp_mult.sh foo/vals3.txt goo/vals3.txt bar/vals3.txt
echo -e "=== foo/vals3.txt ==="
cat foo/vals3.txt
echo -e "=== goo/vals3.txt ==="
cat goo/vals3.txt
echo -e "=== bar/vals3.txt ==="
cat bar/vals3.txt
```

Part #2:

Case #1:

```
echo "old contents to overwrite" > blah.txt
./run_steps.sh echo 18 27 3 blah.txt
cat blah.txt
```

Case #2:

```
chmod +x do_math.sh
./run_steps.sh ./do_math.sh -30 50 10 math_output.txt
cat math_output.txt
```

`do_math.sh`: (available on Canvas)

```
#!/bin/bash

echo "Add 2: "$(($1 + 2))
echo "Subtract 1: "$(($1 - 1))
```

[1] If, due to unique circumstances that we discussed over email, the worth of programming assignment #7 is scaled up to be worth more than 4% of your final grade, the extra credit will not scale, i.e. you cannot earn more extra credit than other students.

Case #3:

```
gcc -Wall -Werror foo.c -o foo
./run_steps.sh ./foo 35 115 15 the_output.txt
cat the_output.txt
```

`foo.c`: (available on Canvas)

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int num = atoi(argv[1]);
    if (num < 50)
        printf("Less than 50.\n");
    else if (50 <= num && num <= 100)
        printf("Between 50 and 100 (inclusive).\n");
    else
        printf("Greater than 100.\n");
}
```

Part #3: See `test_list.cpp` on Canvas. It is compiled with the following line:

```
g++ -Wall -Werror -std=c++11 test_list.cpp -o test_list
```

5 Your Programming Tasks

5.1 Part #1: Copy to Multiple

Filename: `cp_mult.sh`

Motivation: Recall that the `cp` command can be used to copy multiple files into a destination directory, but it cannot be used to copy one file into multiple destinations. You will write a script that does that.

Write a shell script that takes as argument a regular file (i.e. never a directory) and a list of destinations. The script should copy the file into each of the given destinations. If not enough arguments are provided, then a usage message should be printed and the script should return 1 (with `exit 1`); otherwise, the script should return 0. Note that just like `cp`, `cp_mult.sh` can be used to make multiple copies of a file in the same directory.

Below are some examples of how your script should behave.

```
$ ls d1
$ ls d2
$ cat vals.txt
5
18
3
7
$ ./cp_mult.sh vals.txt d1 d2
$ echo $?
0
$ cat vals.txt
5
18
3
7
$ ls d1
vals.txt
$ ls d2
vals.txt
$ ./cp_mult.sh
Usage: ./cp_mult.sh [src] [dest1] ...
$ echo $?
1
$ ./cp_mult.sh vals.txt
Usage: ./cp_mult.sh [src] [dest1] ...
$ ls *.txt
vals.txt
$ ./cp_mult.sh vals.txt vals2.txt vals3.txt vals4.txt
$ cat vals2.txt
5
18
3
```

```
7
$ cat vals3.txt
5
18
3
7
$ cat vals4.txt
5
18
3
7
$
```

5.2 Part #2: Run Steps

Filename: `run_steps.sh`

Motivation: As a happy undergraduate student, I encountered several situations in which I had to run an executable multiple times with different command-line arguments and collect the output of each of those runs. As you can hopefully imagine, doing this manually (i.e. typing each line into the terminal) would have been annoying; you can save time on such a task by writing a script like the one you will write for this part.

Write a script that takes five arguments:

- A command to run (which could be an executable compiled with `gcc`). Note that it must be the *exact* command needed to run it, not just the executable's name (which is why you see `./test` in the example instead of `test`).
- A start integer A.
- An end integer B.
- A step integer S.
- An output file.

The script should perform a series of runs of the given command. The first run should have A be passed as argument to the command to run. In the next run, $A + S$ should be passed. After that, $A + 2S$, then $A + 3S$, then $A + 4S$, and so on, until the value passed would exceed B. Each run's results should be logged into the output file. You may not assume this output file exists, and its old contents should be overwritten.

The script assumes that the executable only takes an integer as its only command-line argument. If not enough arguments are provided, then a usage message should be printed and the script should return 1 (with `exit 1`); otherwise, the script should return 0.

Below are some examples of how your script should behave.

```
$ cat test.c
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Provided: %s\n", argv[1]);
}
$ gcc -Wall -Werror test.c -o test
$ ./run_steps.sh ./test 10 100 5 output1.txt
$ cat output1.txt
=== 10 ===
Provided: 10
=== 15 ===
Provided: 15
=== 20 ===
Provided: 20
=== 25 ===
Provided: 25
=== 30 ===
Provided: 30
=== 35 ===
Provided: 35
=== 40 ===
Provided: 40
=== 45 ===
Provided: 45
=== 50 ===
Provided: 50
=== 55 ===
Provided: 55
```

```
=== 60 ===
Provided: 60
=== 65 ===
Provided: 65
=== 70 ===
Provided: 70
=== 75 ===
Provided: 75
=== 80 ===
Provided: 80
=== 85 ===
Provided: 85
=== 90 ===
Provided: 90
=== 95 ===
Provided: 95
=== 100 ===
Provided: 100
$ ./run_steps.sh ./test 50 80 10 output2.txt
$ cat output2.txt
=== 50 ===
Provided: 50
=== 60 ===
Provided: 60
=== 70 ===
Provided: 70
=== 80 ===
Provided: 80
$ cat test2.c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int num = atoi(argv[1]);
    if (num < 100)
        printf("Less than 100.\n");
    else
        printf("Greater than or equal to 100.\n");
}
$ gcc -Wall -Werror test2.c -o test2
$ ./run_steps.sh ./test2 97 104 2 output3.txt
$ echo $?
0
$ cat output3.txt
=== 97 ===
Less than 100.
=== 99 ===
Less than 100.
=== 101 ===
Greater than or equal to 100.
=== 103 ===
Greater than or equal to 100.
$ ./run_steps.sh ./test2 97 104
Usage: ./run_steps.sh [executable] [start] [end] [step] [outfile]
$ echo $?
1
$
```

5.3 Part #3: Templated Linked List

Filenames: `list.hpp` and `list.inl`

Look at the files `list.hpp` and `list.inl` that are provided on Canvas. These files contain the incomplete definition of a templated linked list. You are to implement this class' methods. In the header file, I describe what each method is supposed to.

You are not allowed to use smart pointers in this assignment. (We talked about these in slide deck #13.) ***You are not allowed to use STL containers that make it so that you are not actually implementing your own linked list, e.g. you cannot use*** *std::list* ***or*** *std::forward_list*.

Manual review: I will do a quick manual review of your submission after the deadline in order to check that you obey the time complexity constraints (where applicable), that you obey the constraints mentioned above, and that you do not

have `public` member variables in `class List`.

Suggestions/tips:

- **Recommended order of completion**: I would probably start with the constructor, `pushFront()`, `pushBack()`, and `operator<<`. I would consider the copy constructor, copy assignment operator, and destructor to be the hardest parts.
- **Required concepts**: If you understand operator overloading, templates, and copy semantics well, then I would think that this assignment should be comparable to programming assignment #5 (`class UnweightedGraph`) in difficulty, so make sure to start by understanding the recent concepts.
- `list.hpp` **vs.** `list.inl`: Because `list.inl` is included in `list.hpp`, it isn't important which functions you choose to define in `list.hpp` vs. `list.inl`. I would recommend avoiding defining all of the required methods in one or the other; I think you should get used to the syntax for both defining a method within the class (in `list.hpp`) and defining a method outside the class (in `list.inl`). For `operator<<`, it did not seem that I could define it outside of `class List` (outside of `list.hpp`, to be clear), as the compiler seems to put additional restrictions on a `friend` of a templated class; I could not find a workaround, but maybe there is one.
- **Avoid duplicating code where possible**: Certain methods and/or operators are so similar (e.g. `pushBack()` and `operator+=`) that, probably, you should have one of them call the other.
- **Template compiler error messages**: If you ask any C++ programmer what the worst part of the language is, they will likely say it's the compiler error messages that you get when dealing with templates. The slightest mistake in dealing with templates can generate a *huge* number of error messages, and this can occur even when you're not explicitly using templates (e.g. when you're using `std::string`, which is a `typedef` for the templated class `std::basic_string<char>`). As is usually the case, you want to start looking at the first error message (in case that one causes all of the other ones), but this requires some scrolling. An alternative that I've sometimes found helpful is to do the command `g++ ... 2>&1 | head`, where `2>&1` redirects the standard error (the compiler error messages) of the `g++` command to standard output so that it can be pipelined into `head`.
- **Initializer list error messages**: If a compiler error is generated by one of your constructor initializer lists, the compiler (at least, `g++`) will flag the last line that the initializer list reaches. This only matters if you spread the initializer list across multiple lines like I tend to, but it's something to be aware of, because it can be deceiving if the cause of the error has to do with something towards the beginning of the initializer list instead of the end.

Below are examples of how your code should behave. You can find `main.cpp` on Canvas.

```
$ cat main.cpp
#include "list.hpp"

struct X
{
    X() { std::cout << "Constructor!\n"; }
    ~X() { std::cout << "Destructor!\n"; }
};

int main()
{
    std::cout << std::boolalpha;
    List<int> lst;
    lst.pushBack(15);
    lst.pushFront(33);
    lst.pushFront(20);
    lst.pushBack(50);
    lst.pushFront(40);
    lst += 80;
    std::cout << lst;
    std::cout << lst.length() << '\n';
    std::cout << "lst[3]=" << lst[3] << '\n';
    lst[2] = 90;
    std::cout << lst;
    std::cout << "Contains 50: " << lst.contains(50) << '\n'
        << "Contains -10: " << lst.contains(-10) << '\n';
    std::cout << "Remove 50: " << lst.remove(50) << ' ' << lst;
    std::cout << "Remove 80: " << lst.remove(80) << ' ' << lst;
    std::cout << "Remove -10: " << lst.remove(-10) << ' ' << lst;
    lst.pushFront(200);
    lst.pushBack(300);
    lst += 100;
    std::cout << lst;
    lst.remove(200);
    lst -= 40;
    std::cout << lst;
```

```
    std::cout << "Copy construction...\n";
    List<int> lst2{lst};
    lst2.remove(90);
    std::cout << lst << lst2;
    std::cout << "Copy assignment...\n";
    lst2 = lst;
    std::cout << lst << lst2;
    lst.pushBack(-400);
    lst2.pushFront(-100);
    lst2 += -200;
    std::cout << lst << lst2;
    std::cout << lst.length() << ' ' << lst2.length() << '\n';

    std::cout << "Trying strings...\n";
    List<std::string> strings;
    std::cout << strings;
    std::cout << strings.length() << '\n';
    strings.pushBack("def");
    strings.pushFront("abc");
    strings += "ghi";
    std::cout << strings;
    std::cout << strings.length() << '\n';
    strings -= "gh";
    strings -= "abc";
    std::cout << "After removing \"abc\": " << strings;
    std::cout << strings.length() << '\n';
    strings.pushFront("jklmn");
    std::cout << strings;
    std::cout << strings[2] << '\n';
    std::cout << "Copy construction...\n";
    auto strings2 = strings;
    std::cout << strings[1] << ' ' << strings2[1] << '\n';
    strings.remove("def");
    strings2.pushFront("op");
    std::cout << "strings: " << strings
              << "strings2: " << strings2;
    std::cout << "strings[1]: " << strings[1] << '\n'
              << "strings2[1]: " << strings2[1] << '\n';

    std::cout << "Trying class X...\n";
    List<X> objs;
    objs.pushBack(X{});
    std::cout << "Trying out-of-range index...\n";
    try
    {
        auto o = objs[3];
    }
    catch (const std::out_of_range& ex)
    {
        std::cerr << "I accept your exception!\n";
    }
    catch (...)
    {
        std::cerr << "Wrong type of exception!\n";
    }

    std::cout << std::noboolalpha;
}
$ g++ -Wall -Werror main.cpp
$ ./a.out
40 20 33 15 50 80
6
lst[3]=15
40 20 90 15 50 80
Contains 50: true
Contains -10: false
Remove 50: true 40 20 90 15 80
Remove 80: true 40 20 90 15
Remove -10: false 40 20 90 15
200 40 20 90 15 300 100
20 90 15 300 100
Copy construction...
20 90 15 300 100
20 15 300 100
```

```
Copy assignment...
20 90 15 300 100
20 90 15 300 100
20 90 15 300 100 -400
-100 20 90 15 300 100 -200
6 7
Trying strings...

0
abc def ghi
3
After removing "abc": def ghi
2
jklmn def ghi
ghi
Copy construction...
def def
strings: jklmn ghi
strings2: op jklmn def ghi
strings[1]: ghi
strings2[1]: jklmn
Trying class X...
Constructor!
Destructor!
Trying out-of-range index...
I accept your exception!
Destructor!
$ valgrind ./a.out > /dev/null
...
==3198== All heap blocks were freed -- no leaks are possible
==3198==
==3198== For counts of detected and suppressed errors, rerun with: -v
==3198== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
$
```

UCDAVIS
COMPUTER SCIENCE