Homework 4

EE 262 (Fall 2021)

Department of Electrical and Computer Engineering
Clarkson University

Instructions

Note 1: Please read all instructions carefully before submitting your work. There are **2 questions** in this homework for a total of **60** points.

Note 2: Solve all problems and upload your answers to Moodle using the instructions below. Make sure any code you write works on Polaris before uploading your files. You will likely lose many points if your code doesn't compile. Whenever you write solutions on paper, you need to scan all documents before submitting.

Note 3: For this homework, you can work individually or in groups of up to 3. There should be only one submission per group.

Note 4: `USR` stands for your login ID on Polaris (`polaris.clarkson.edu`). `MJR` represents your major, i.e. CE, SE, EE, CS, etc.

Note 5: In program listings and shell interactions, any **`bold, underlined, constant-width`** text indicates that it is user input.

Note 6: In program listings and shell interactions, any *`italicized, underlined, constant-width`* text has to be explicitly replaced by an appropriate user-supplied value.

Note 7: Do not upload any executable or intermediate files as answers to problems, unless specifically asked to do so.

Note 8: You should upload a single file named `USR_MJR_HW4.zip` (or named using the pattern `USR1_MJR_USR2_MJR2_HW4.zip` if submitting in a group of two, provided group submissions are allowed) to Moodle. Similarly, the file name should include all group members for larger groups.

Note 9: If you have multiple majors, please mention all of them in the submission file name, e.g. `USR_MJR1_MJR2_HW4.zip`; or if submitting in a group of two, the file name will have the pattern `USR1_MJR1_MJR2_USR2_MJR_HW4.zip`, if `USR1` has two majors and `USR2` has one major.

Note 10: In the ZIP file, you should have a directory named `hw4`, inside which you should have directories for each problem, named to clearly indicate the problem number, i.e. `p1`, `p2`, etc, each of which should contain the appropriate deliverable. To verify the directory structure before creating the zip file, the `tree` command on Polaris can be used to display the directory contents in a hierarchical fashion, e.g. `tree dirName`.

Note 11: Please read the "Note on academic honesty" in the syllabus. In the directory for *each* problem, please include a `readme.txt` file containing the following:

- It should state whether you worked alone on that probem or list the full names of all the people you discussed it with. This includes the names of the course staff, if any of them helped you with the problem.

- If your answer includes or is inpired by a non-trivial amount of content (including code) from specific sources, please cite all of them prominently.

Note 12: On Polaris, the following command can be used to zip the contents of directory `hw4` into a file: `ZIPFLNAME.zip`: `zip -9 -yrq ZIPFLNAME.zip hw4`

Note that you will need to rename the ZIP file as per the instructions described earlier. Please test the generated zip file by listing its contents (`unzip -l ZIPFLNAME.zip`) and also by extracting and viewing its contents in a separate (temporary) directory (`unzip ZIPFLNAME.zip`).

1. [30 points] In this problem, you will implement the design of a C++ *template class* `Triple` that can be used to represent *triples of a specified type*, e.g. a triple of integers like (10, -40, 200) where 10, -40, and 200 are the first, second and third elements in the triple, respectively.

   The interface (header) file defining `Triple` has been provided. Write the implementation in a file called `triple.cpp` and make sure it works with the driver `client2.cpp`.

   At the end of the implementation file, you will may need to do an 'explicit instantiation' of the template for types that are used in the driver as shown in Listing 1.

   Listing 1: Explicit Instantiation

```
//explicit instantiation
template class Triple<long>;
```

   Write your code so as to the same output on running the executable generated by `client2.cpp` as shown in Listing 2.

   Listing 2: Sample interaction the driver in `client2.cpp`.

```
$ g++ -c triple.cpp
$ g++ client2.cpp triple.o -o client2.out
$ ./client2.out
var1: (11, 1500, 780000)
var1's third element: 780000

var2: (t, n, o)
var2's first element: t

var3: (0.44, 1.567, -89.4325)
var3's second element: 1.567

var4: (go, away, corona)

unnamed: (-10, 20, 30)
```

   In `readme.txt`, clearly describe how to compile and run your code on Polaris.

   Deliverable: `triple.cpp`, `readme.txt`.

2. [30 points] In `safesum.cpp`, write a C++ program containing a function `safeadd` with the following signature:

```
//precondition: n1 >= 0, n2 >= 0
int safeadd(int n1, int n2);
```

`safeadd` protects against overflow by raising an `std::overflow_error` exception if the sum is going to be more than `INT_MAX`, otherwise it returns the sum of its arguments.

Note: `INT_MAX` is defined in the C++ header file `<climits>`.

The driver code in `safesum.cpp` is shown in Listing 3. The driver handles the exception generated by `safeadd`. Note that `std::runtime_error` is a superclass of `overflow_error`.

Listing 3: Driver code for `safesum.cpp`

```
int main(void) {
  cout<<"INT_MAX (on this machine): "<<INT_MAX<<endl;
  int bigNum1 = INT_MAX - 100;

  try {
    cout<<"safeadd(7, 500): "<<safeadd(7, 500)<<endl;

    cout<<"safeadd("<<bigNum1<<", 200): "<<std::flush;
    cout<<safeadd(bigNum1, 200)<<endl;
  } catch (std::runtime_error exp) {
    cerr<<endl<<"main: Caught exception at runtime:
       "<<exp.what()<<endl;
  }

  cout<<endl<<endl;
  cout<<"safeadd(300, 11): "<<safeadd(300, 11)<<endl;

  ostringstream oss;
  oss<<"calling safeadd("<<bigNum1<<", 200): "<<std::flush;
  cout<<oss.str()<<std::flush;
  cout<<safeadd(bigNum1, 200)<<endl;

  return 0;
}
```

Note: The driver code shown in Listing 3 should be in your `safesum.cpp`.

The result of running the driver in `safesum.cpp` that contains a correct implementation of `safeadd` is shown in Figure 1.

In `readme.txt`, briefly describe how to compile and run your code on Polaris. In the same file, briefly explain the difference in behavior for the two exceptional situations shown in Figure 1 with respect to the control flow and the output obtained.

Figure 1: Expected output when using the driver in Listing 3.

```
$ g++ -o safesum.out safesum.cpp
$ ./safesum.out
INT_MAX (on this machine): 2147483647
safeadd(7, 500): 507
safeadd(2147483547, 200):
main: Caught exception at runtime: safeadd: overflow during integer addition.



safeadd(300, 11): 311
calling safeadd(2147483547, 200): terminate called after
throwing an instance of 'std::overflow_error'
  what():  safeadd: overflow during integer addition.

Aborted
```

Deliverable: `safesum.cpp`, `readme.txt`.

End