**CYB 220 Lab 3 – Integer Overflow**

**Turn in: (1) this report with the answers typed in and screenshots pasted at specific places, and**

**(2) program (.cpp file) .**

**Due: Friday, Sep 20th, 11:59pm on CANVAS.**

**Points: 30 pts**

**Name:**

Objective: Understand the integer overflow/underflow problem.

There are three tasks in this lab.

* (5 pts) Task 1: What happens if we assign an out-of-range value to a variable or get an out-of-range value from input?
* (10 pts) Task 2: What happens when integer calculations result in a number outside that set?
* (10 pts) Task 3: Know the ranges available for each data type.
* (5 pts) Turn in your program

**TASK 1:** What happens if we assign an out-of-range value to a variable or get an out-of-range value from input? (Note: A **numbers-lab3.cpp** can be found on CANVAS.)

* When assigning a value out of range?

int num;

cout << "num is declared as a signed int (-2147483648 ~ 2147483647)" << endl;

num = 2147483999;

cout << "After the assignment, num is: " << num << endl;

* When read in an out-of-range value from input (cin) to a variable?

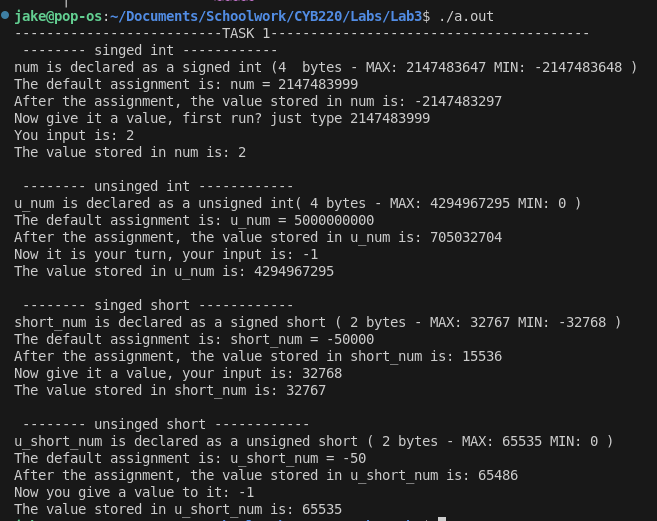
cin >> num;

cout << "The value stored in num is: " << num << endl;

RUN the number-lab3.cpp program and see what values you get.

* + first run, try the default numbers
  + later runs, try other out-of-range values you are interested in.

[Screenshot(s) (paste here) on the first run and at least another run with out-of-range values.]



Answer questions:

Question 1a: What values did you get? When assigning a value out of range? Explain in your own words.

When I initialized values out of range, it would wrap around positive to negative or negative to positive, as expected.

Question 1b: What values did you get? When read in an out-of-range value from input (cin) to a variable? Explain in your words.

When I used cin to change the value of a variable, It would truncate the input to one limit or the other, so my value would always be changed to INT\_MIN or INT\_MAX.

**Task 2:** What happens when integer calculations result in a number outside that set?

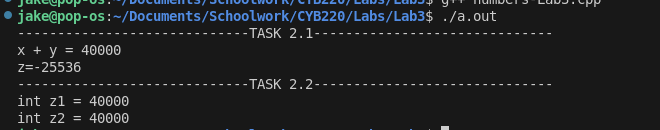
A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Task 2.1 - see slides for code details.



[Paste the screenshot here to show the output for the 4 examples.]

Question 2.1a: Will two examples (2.1a and 2.1b) output the same? What will they output?

They will not output the same. 2.1a has x+y automatically as an int, so it will print the correct value (40000). 2.1b specifies short, and as such, the value will be wrong, and should be overflow amount plus remainder = 40000 - 32767 - 32768, which is -25536.

Question 2.1b: Out of the four examples, which one(s) got correct value vs. which one(s) got overflowed value?

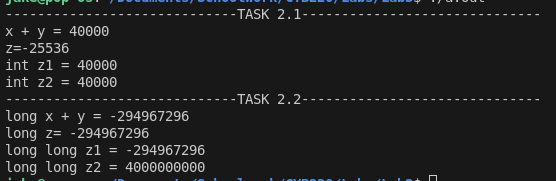
All of the ones that had some sort of integer type or type conversion got the correct answer, whereas ones that didn't specify a type go the overflowed answer.

Task 2.2 – see slides for more detailed requirements.

A screenshot of a computer

Description automatically generated

Note: Do not make modifications on your 1a, 1b, 1c, 1d code, have new code for 2a, 2b. 2c, 2d.



[Take a screenshot of your output from task 2.1 and task 2.2 (should have 8 output), and paste here.]

Question 2.2a: Out of the four examples, which one(s) got the correct value vs. which one(s) got the overflowed value?

a, b, and c all had wraparound, whereas d worked.

Question 2.2b: Compare the results from task 2.1 with task 2.2’s. Document any observation. Any interesting finding?

For the same reason why having the addition happen in the cout worked in 2.1a, it didn't work in 2.2a. It was automatically typecast to an int. Same thing applies to 2.2b, but I'm unsure why it didn't work for long long addition.

**Task 3:** know the ranges available for each data type.

Print sizeof(), MAX and MIN values/macro constants for each of the 10 data types

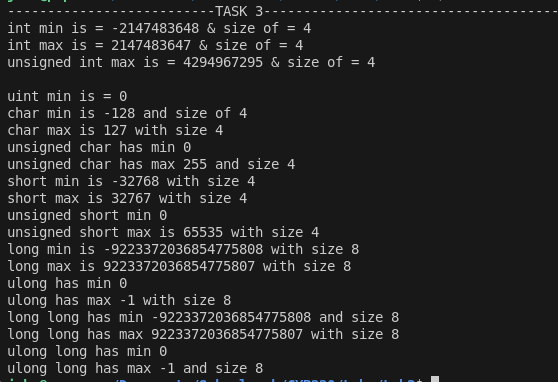
* (1) int, (2) unsigned int, (3) char, (4) unsigned char,
* (5) short, (6) unsigned short, (7) long, (8) unsigned long
* (9) long long, (10) unsigned long long

eg. for int, print sizeof(int), INT\_MAX, INT\_MIN

Note: unsigned datatypes do not have the MIN macro, but we all know the MIN of an unsigned datatype is 0, so no need to have those in your code.

More information about the <climits> and MACROs - <https://www.cplusplus.com/reference/climits/>

Paste a screenshot here to show this output.



Answer question 3: examine the values you got, are these values same with the mins and maxs (the range) we discussed in class?

No, my long and long long values are the same, and then the ulong long max value didn't print correctly because it is larger than the long long data type that it's being printed as can support.