

CSE278: Introduction to Systems Programming (Systems I)

Homework #9

Due: Friday May 8, 2020 before 11:59 PM

Email-based help Cutoff: 5:00 PM on Wed, May 6, 2020

Maximum Points: 50

Submission Instructions

This part of the homework assignment must be turned-in electronically via Canvas. Ensure you name this document `HW9_MUID.docx`, where `MUID` is your Miami University unique ID. (Example: `HW9_ahmede.docx`)

Copy pasting from online resources is **Plagiarism**. Instead you should read, understand, and use your own words to respond to questions.

Submission Instructions:

Once you have completed answering the questions save this document as a PDF file (**don't just rename the document; that is not the correct way to save as PDF**) and upload it to Canvas.

General Note: Upload each file associated with homework (or lab exercises) individually to Canvas. Do not upload archive file formats such as zip/tar/gz/7zip/rar etc.

Objective

The objective of this homework is to review basic concepts of:

- C++
- Computer architecture and organization
- Systems Security

Name: **Ben Hilger**

Required reading

- Lecture Slides: cpp (part 1..4)
- Lecture Slides & ClassNotes: Security
- Lecture Slides & ClassNotes: ComputerArchitecture

1. Netbeans 8.2 has a built-in editor; is that the right thing to use?

This is the right thing to use because it provides additional functionality with syntax errors and automated features such as adding a class and main method that aren't available in other editors such as vim or nano.

2. Consider the following command of compiling C++ code:

```
g++ -std=c++14 -g -Wall source.cpp -o nameOfExecutable
```

What is the purpose of the flag `Wall`?

Can we ignore warning messages from the compilation?

The purpose of the flag `-Wall` is to display all of the warnings that are found when compiling a program. They can be ignored because warnings typically indicate inefficiencies in code, such as unused variables that take valuable memory, rather than errors that would result in the program not compiling or performing as intended.

3. Why use unnecessary parentheses when precedence will determine which operators are acted on first?

These parentheses improve the readability of code by offering an easier way to understand the order which the operators are being acted on even though they don't serve a specific function in code.

4. Are negative numbers true or false?

All negative numbers are true because all nonzero numbers are evaluated to true.

5. Why aren't changes to the value of function arguments reflected in the calling function?

This is because, unless otherwise specified, function arguments are sent as copies of the original value. This means that the function arguments are given their own temporary location in memory instead of pointing to the same memory location of the original reference. This means that any change to those function arguments

are made at a different memory reference, and therefore not reflected in the calling function.

6. How big is a C++ class object?

How big a C++ class object is, is dependent on the variables and functions stored inside. The size of the object will typically be a summation of all of the sizes of the individual variables and functions and possibly a bit more depending on overhead/padding the compiler gives the class.

7. How do you choose between `while` and `do ... while` ?

You would choose a `do...while` if you want it to run at least once, such as for inputting a number between a specific range, where it may need that input in the evaluation to continue looping. Whereas you would use a `while` any other time where it may not run at all since its evaluation to run or not occurs before the first run.

8. What is the significance of `while (1) ; for(;;)` ; Is it better to use `while (1)` or `for(;;)` ?

The `while (1); for(;;)` is significant because this loop will run indefinitely until the code hits a break; the `for(;;)` is better because it isn't doing any comparison and is instead just running whereas every time the `while(1)` is evaluating `1 == 1` every time, which takes more time.

9. What happens if I write to element 25 in a 24-member array?

You will get an `IndexOutOfBounds` error since there doesn't exist a memory reference to element 25 in a 24-member array. This means the array will overwrite an adjacent memory location it's not supposed to, creating unintended behaviors and issues with the program.

10. Explain different addressing modes

Each different addressing mode specified how to calculate memory addresses of an operand by using information held in registers and/or constants contained within a machine instruction or elsewhere

11. What is a stack (Chapter 6.11)? Using a C++ code fragment discuss how stack is used to store variables? (Text book page 279-280)

A stack is a data structure that implements the last-in, first-out (LIFO) ideology. This means that if something is added to the stack, it's added to the top of the list. And when something is removed it's always removed (or popped) from the top of the list in the stack.

A stack is used to store variables by creating stack frames. A stack frame is an entry on the stack and is created when a function such as "calculateAverage(int a, int b)." This function call would create a stack frame, which would reserve the memory for the variables, "a," and, "b." Then, if there was a variable called "average = ((a+b)/2);" the same stack frame would also reserve the memory for the variable. Then, when the function returns the average with, "return average;" the stack frame would be popped from the stack, therefore removing the variables as they should no longer exist outside the scope of the function.

12. What makes machine code difficult to deal with and maintain?

The problem with dealing and maintaining machine language is that modern microprocessors can contain more than 100 different instructions. And developing bit patterns can become cumbersome for each instruction.

13. What is a hazard (or stall)? What are the 3 types of hazards that can occur in a pipeline?

A hazard is a problem with the instruction pipeline which occurs when the next instruction can't execute in the following clock cycle, leading to potential incorrect computation results.

The 3 types of hazards are:

Read after write (RAW): This hazard occurs when something tries to read a source before something else writes to it.

Write after read (WAR): This hazard occurs when an instruction tries to write to a destination before another instruction can read it.

Write after write (WAW): This hazard occurs when an instruction writes to an operand before it is written by the previous instruction.

14. Present an example of a symbol table with at least 3 symbols in it. Explain the concept and use of a symbol table using your example (no more than 4 sentences).

Symbol (variable)	Address (Hex)	Size (bytes)
k	0x452D78	2
email	0xFBC100	4
_terminate	0x000001	8

A symbol table allows people to associate descriptive names with computer memory addresses. Since computers only care about addresses, a symbol table allows for a program to map symbols with a corresponding address. Such as the email symbol in the above table points to the address 0xFBC100 with a size of 4 bytes, which means when a programmer uses the symbol email in a program, the computer will come to the symbol table and find that address and size to read the correct data from memory.

15. Consider the following code fragment:

```
#include <iostream>

using namespace std;

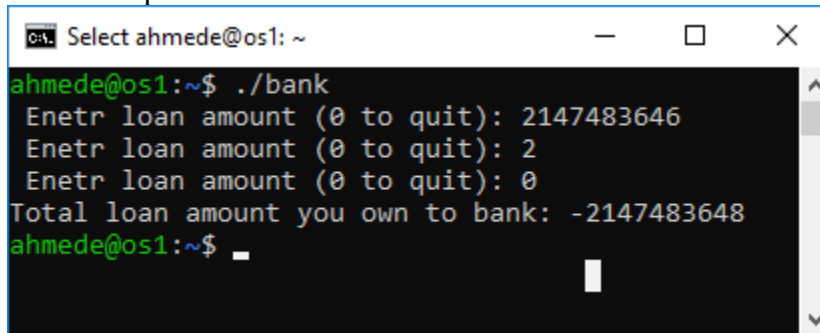
int main() {

    int sum = 0, debit = 0;

    do {
        std::cout << " Enter loan amount (0 to quit): ";
        std::cin >> debit;
        if (debit > 0) {
            sum += debit;
        }
    } while (debit > 0);
    std::cout << "Total loan amount you own to bank: " << sum << std::endl;
```

```
return 0;  
}
```

And a sample run screen shot:



```
ahmede@os1:~$ ./bank  
Enter loan amount (0 to quit): 2147483646  
Enter loan amount (0 to quit): 2  
Enter loan amount (0 to quit): 0  
Total loan amount you own to bank: -2147483648  
ahmede@os1:~$
```

Use sufficient comments in the code (`// /* */`)

Re-run the code with some other suitable input value to demonstrate NumericOverflow.

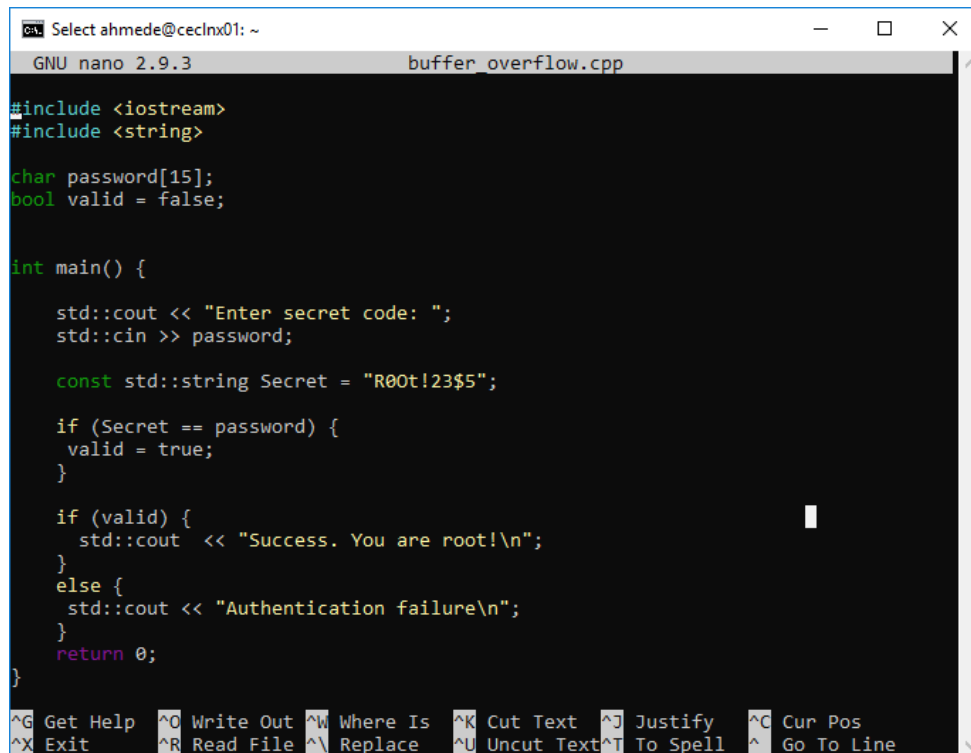
Attach a screen shot of the run

```
[hilgerbj@os1:~/cse278/Homework9$ ./bank  
[ Enter loan amount (0 to quit): 2147483645  
[ Enter loan amount (0 to quit): 6  
[ Enter loan amount (0 to quit): 0  
Total loan amount you own to bank: -2147483645  
hilgerbj@os1:~/cse278/Homework9$
```

Explain why and how *numeric* overflow might occur.

A numeric overflow occurs because integers only have a certain range of numbers they can be since they have a set size in memory. So, when a value that's too large for an integer value to hold, a numeric overflow error occurs. For integers what usually happens is that if the integer is too large, it'll "wrap" around from the highest positive to the highest negative number possible for an int.

16. Experimenting with *buffer* overflow. Consider the following code fragment:



```
GNU nano 2.9.3 buffer_overflow.cpp

#include <iostream>
#include <string>

char password[15];
bool valid = false;

int main() {

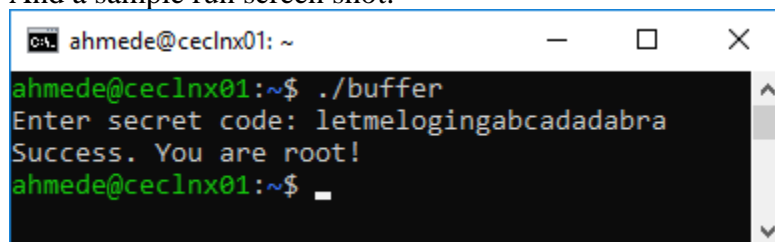
    std::cout << "Enter secret code: ";
    std::cin >> password;

    const std::string Secret = "R00t!23$5";

    if (Secret == password) {
        valid = true;
    }

    if (valid) {
        std::cout << "Success. You are root!\n";
    }
    else {
        std::cout << "Authentication failure\n";
    }
    return 0;
}
```

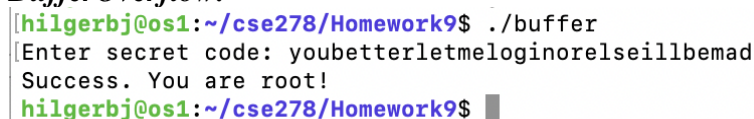
And a sample run screen shot:



```
ahmede@ceclnx01: ~
ahmede@ceclnx01:~$ ./buffer
Enter secret code: letmelogingabacadadabra
Success. You are root!
ahmede@ceclnx01:~$
```

Use sufficient comments in the code (`// /* */`)

Re-run the code with some other suitable input value to demonstrate BufferOverflow.



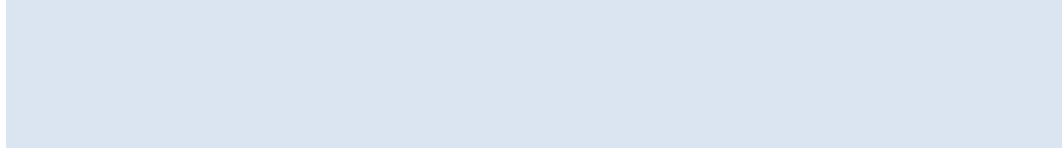
```
[hilgerbj@os1:~/cse278/Homework9$ ./buffer
[Enter secret code: youbetterletmeloginorelseillbemad
Success. You are root!
hilgerbj@os1:~/cse278/Homework9$
```

Attach a screen shot of the run

Explain why this has been a successful login still the password is incorrect.

It was a successful login because the string that was inputted was much larger than what was allocated for it in memory, which means that it overwrites information in adjacent memory locations, corrupting memory. In this case, it probably overwrote the root key string in the program, causing the unintentional error that they inputted the correct key.

Due before: 11:59 PM (before Midnight) on Friday May 8, 2020



Submission

- No late assignments will be accepted!
- This work is to be done individually
- The submission file will be saved with the name ***HW9_yourMUID.pdf***
- Assignment is due before Midnight Friday May 8, 2020.
- On or before the due time, drop the *electronic copy* of your work in the *canvas*
- **Don't forget to Turn in the files! HW9_yourMUID.pdf & HW9_yourMUID*.cpp**