**ECE 15200: Programming for Engineers**

**Purdue University Northwest, ECE Department**

Laboratory 8: Pointers

**Instructions**:

* Submit only C++ source files (\*.cpp) for all the problems through Brightspace.
* Name each file following the format ***Lastname\_*Lab*X*\_p*Y*.cpp**, replace *Lastname, X,* and *Y* with your last name, lab #, and problem #, respectively.
* Put your name, assignment number, and date on the top of each source file (\*.cpp) as multi-line comment given below:

/\*

Class: ECE15200

Author: [Your Name]

Assignment: Lab [No.]

Date: [MM]/[DD]/[YY]

\*/

Remove the brackets after updating the information in them.

* PLEASE WORK ALONE. If any plagiarism is found, you will get ZERO. Never hesitate to discuss with the instructor/TA if stuck in any assignment problem.

**Problem 1. (**Lastname\_Lab8\_p1.cpp**)** Write a C++ program to accept five integer values from the keyboard. The five values will be stored in an array using a pointer. Then display the elements of the array on the screen [**20 points**].

**Problem 2**. **(**Lastname\_Lab8\_p2.cpp**)** Write a program that stores the following numbers in the array named miles: 15, 22, 16, 18, 27, 23, and 20. Use a pointer to copy the data stored in miles to another array named dist and then display the values in the dist array [**20 points**].

**Problem 3**. **(**Lastname\_Lab8\_p3.cpp**)** Write a function void reverse(char s[]) that reverses a character string. For example, “Harry” becomes “yrraH”. Write the main() function that will call reverse() and display the reversed string [**20 points**].

**Problem 4**. **(**Lastname\_Lab8\_p4.cpp**)** Write a program that stores a string “C plus plus is a simple programming language” in an array named str. Use a pointer \*Pt to count the number of character ‘p’ in the string [**20 points**].

**Problem 5**. **(**Lastname\_Lab8\_p5.cpp**)** Write a program that declares three one-dimensional arrays named miles, gallons, and mpg. Each array should be declared in main() and should be capable of holding ten double-precision numbers. The numbers that should be stored in miles are 240.5, 300, 189.6, 310.6, 280.7, 206.9, 199.4, 160.3, 177.4, and 192.3. The numbers should be stored in gallons are 10.3, 15.6, 8.7, 14, 16.3, 15.7, 14.9, 10.7, 8.3, and 8.4. Each element of the mpg array should be calculated as the corresponding element of the miles array divided by the equivalent element of the gallons array. (e.g., mpg[0] = miles[0]/gallons[0]). Use ***pointers*** when calculating and displaying the elements of the mpg array [**20 points**].

**(Optional Cybersecurity awareness Part 1: Integer overflow continued.)**

**Problem 6**. (Lastname\_Lab8\_p6.cpp). We have already studied and used array. There is an issue with C/C++ in accessing array elements. C/C++ does not check accessing array elements if it goes beyond the size of an array. For example, if we declare an array, x of size 10, we should only be able to access elements from x[0] to x[9], however if you access or modify an array element with index 10 or beyond (e.g. x[11]), there is no restriction from C/C++. This problem is called **buffer overflow**, where an array is accessed or modified beyond the space allocated to it. Buffer overflow may happen due to improper input validation or lack of index checking while accessing array elements. One of the consequences of buffer overflow is that it can access or overwrite data stored after the array elements in memory. This data could have sensitive information or return address of a function. If the program accesses this data due to buffer overflow then it could be a data breach. If a program overwrites data due to buffer overflow, then it could manipulate the sensitive information or return address of a function, which may crash the program. Buffer overflow is a very critical and known security vulnerability. Although there is huge familiarity with buffer overflow, programmers still make mistakes that may lead to buffer overflow. As a programmer, we should write robust code so that this problem should not happen.

|  |  |  |
| --- | --- | --- |
| Memory Address | Data | Values |
| 1504 | A[0] | 20 |
| 1508 | A[1] | 30 |
| 1512 | A[2] | 40 |
| 1516 | A[3] | 50 |
| 1520 | A[4] | 60 |
| 1524 | B[0] | 12 |
| 1528 | B[1] | 24 |
| 1532 | B[2] | 36 |
| 1536 | B[3] | 48 |
| 1540 | B[4] | 60 |
| 1544 | C | 0 |

Let us understand buffer overflow through an example. Figure on right shows the memory snapshot of a program having two integer arrays, A & B and a variable C stored adjacent to each other. Here, we have assumed the addresses for A, B, and C for illustration purpose. In a robust program, we should only be able to access or modify elements A[0] to A[4], and B[0] to B[4] in arrays A and B, respectively. However, if there is an issue in the program, it can access or overwrite values using arrays at addresses beyond the address space allocated to them. For example, if a program performs following assignment (may be due to lack of input validation or index checking):

B[5] = 10;

In this case, variable C will be changed from 0 to 10, as B[5] points to location 1544, which is allocated to C. The above statement causes buffer overflow.

To make it clear, we have given you a program, ece152\_cybersec2\_1.cpp that has buffer overflow vulnerability. The program is mimicking login to a system. If input values for username and password match with the stored values for them, then the system gives access to the user otherwise not. We have kept username admin, and password @dm1n152. If someone enters them correctly, access will be granted. If any of entered values does not match then access should be denied. That is how it should be, right? Otherwise, anyone can access your computer, email or social media account without knowing your password. Now, run the given program in <https://replit.com/languages/cpp> with following pair of inputs for username and password:

1. admin, @dm1n152
2. admin, admin123
3. root, @dm1n152
4. admin, aaaaaaaaaaaaaaaaaaa

Write your observation for each case of input. Do you find any serious security issues in any of the cases? If yes, can you explain and fix the program so that it should not happen. Submit the revised code as Lastname\_Lab8\_p6.cpp. Provide your observations and explanation as comments in the submitted CPP file.

**Problem 7**. (Lastname\_Lab8\_p7.cpp) When developing a large software, no single developer writes everything. If a team is building a program, they may not check each other’s work thoroughly for errors that can break or cause a weird behavior. One such case is mismatch in the parameters sent to a function and the parameters the function expects. Similarly, there could be also a mismatch in the return variable. We have given program ece152\_cybersec2\_2.cpp to illustrate this idea. Run this program with the following values for withdraw: 1000, 2000, 33000. Do you observe anything wrong in any output? If yes, fix the program so that it should not have such issues. Submit your revised code with name as Lastname\_Lab8\_p7.cpp.