NWEN 241 Assignment 2

"On Strings and Pointers"

Release Date: **18 March 2019**

Submission Deadline: **1 April 2019, 23:59**

In this assignment, you will be given 11 tasks where you will apply your knowledge about strings and pointers.

The assignment is divided into 3 parts. In Part I (Tasks 1–5), you will be asked to answer questions about pointers. In Part II (Tasks 6–8) and Part III (Tasks 9–11), you continue the implementation several basic text editor operations. You will implement some of the operations in pure C (Part II), and some in C++ (Part III).

You only need to submit the required implementations. You do not need to write a `main()` function, but you would need one if you want to test your code. **Sample code showing an example on how you can test your code are provided under the `files` directory in the archive that contains this file.**

Full marks is 100.

Instructions and Submission Guidelines:

- For Parts II and III, you should provide appropriate comments to make your source code readable. If your code does not work and there are no comments, you may lose all the marks. See the marking criteria at the end of this document for details about the marks for commenting.
- For Parts II and III, you should follow a consistent coding style when writing your source code. There is a short discussion about coding style below. See the marking criteria at the end of this document for details about the marks for coding style.
- Submit the required files to the Assessment System (`https://apps.ecs.vuw.ac.nz/submit/NWEN241/Programming_Assignment_2`) on or before the submission deadline.

- Late submissions (up to 48 hours from the submission deadline) will be accepted but will be penalized. No submissions will be accepted 48 hours after the submission deadline.

Coding Style for Parts II and III

Coding style (also known as coding standard) refers to the use of appropriate indentation, proper placement of braces, proper formatting of control constructs, etc. Following a particular coding style consistently will make your source code more readable.

There are many coding standards available (search "C/C++ coding style"). Most of these standards are dense and will take you many days (even weeks) to read and understand. If you want to follow a *lightweight* coding style, consult the Linux kernel coding style (`https://www.kernel.org/doc/html/v4.10/process/coding-style.html`). You only need to read sections 1–3 of this document.

Note that you do not have to follow every recommendation you can find in a coding style document. If you change, for instance the tab size from 8 to 4, that is fine. You just have to apply that style consistently.

Part I: Pointer Concepts

This part will test your conceptual knowledge of pointers. Your answers should be submitted in a plain text file named `part1.txt`.

Task 1.

Basics [10 Marks]

1. **(2 Marks)** Declare a pointer to a floating-point quantity.
2. **(2 Marks)** Declare a prototype for a function `func1` that accepts two integer arguments and returns a pointer to a long integer.
3. **(2 Marks)** Declare a prototype for a function `func2` that accepts a character pointer argument and returns a pointer to a character. The function must not allow the modification of the input argument.
4. **(2 Marks)** Declare a one-dimensional array of integer pointers with 20 elements.

5. **(2 Marks)** Declare an array of C strings whose initial values are `"cyan"`, `"magenta"`, `"yellow"`, and `"black"`.

Task 2.

Completion [5 Marks]

A C/C++ program contains the following statements:

```
1  char u, v = 'A';
2  char *pu, *pv = &v;
3
4  *pv = v + 1;
5  u = *pv + 1;
6  pu = &u;
```

Each character occupies 1 byte of memory. If the value assigned to `u` is stored in (decimal) address `1100` and the value assigned to `v` is stored in (decimal) address `1101`, then

1. **(1 Mark)** What is the numeric value of the expression `&u`?
2. **(1 Mark)** What is the numeric value of the expression `&v`?
3. **(1 Mark)** What value is assigned to `pv` after the completion of line 2?
4. **(1 Mark)** What is the numeric value of the expression `*pv` after the completion of line 4?
5. **(1 Mark)** What value is assigned to `u` after the completion of line 5?

Task 3.

Completion [5 Marks]

Consider the following C++ code snippet:

```
int int1 = 26;
int int2 = 45;
int *int1Ptr = &int1;
int *int2Ptr = &int2;

*int1Ptr = 89;
```

```
*int2Ptr = 62;
int1Ptr = int2Ptr;
*int1Ptr = 80;
int1 = 57;

std::cout << int1 << "   " << int2 << endl;
std::cout << *int1Ptr << "   " << *int2Ptr << endl;
```

1. **(1 Mark)** What is its output?
2. **(4 Marks** Explain individually the printed values.

Task 4.

Challenge [3 Marks]

A C/C++ program has the following statements:

```
short a[] = {1, 2, 4, 8, 16, 32};
short *pa = a;
short **ppa = &pa;
```

Suppose each short integer quantity occupies 2 bytes of memory. If the array `a` is at (decimal) address 1102, `pa` is at (decimal) address 1114, and `ppa` is at (decimal) address 1118, then

1. **(1 Mark)** What is the numeric value of the expression `a`?
2. **(1 Mark)** What is the numeric value of the expression `ppa`?
3. **(1 Mark)** What is the numeric value of the expression `*ppa + 2`?

Task 5.

Challenge [2 Marks]

Consider the following C++ code snippet:

```
std::string seasons[4] = {"Winter", "Spring", "Summer",
   "Fall"};
std::string *strPtr;
strPtr = new string[5];
```

```
for (int i = 0; i < 4; i++)
    strPtr[i] = seasons[i];
```

1. **(1 Mark)** Write a C++ code snippet that outputs the contents of the array to which `strPtr` points using pointer notation.
2. **(1 Mark)** Write a C++ code snippet that deallocates the memory space occupied by the array to which `strPtr` points.

Part II: Pure C Programming

You may only use the Standard C Library to perform the tasks in this part.

Task 6.

Basics [15 Marks]

In this task, you will create a C header that will declare the functions that you will implement in Tasks 7 and 8. You may declare constants and user-defines types needed by your function implementations.

Save the header file as `editor2.h`.

Task 7.

Completion [15 Marks]

In this task, you will implement a function that counts the number of lines from a given editing buffer. The editing buffer is an array of characters, and is passed to the function using pointer notation. The size of the buffer should also be passed to the function. The function will not modify the contents of the buffer, and this should be indicated in the function header (and prototype).

The function should be named `editor_count_lines`. It must return the number of lines in the buffer. A *line* is defined as consisting of the characters: (i) from the start of the buffer until a newline character; or (ii) immediately after a newline character until another newline character.

The arguments should be ordered as follows:

1. Editing buffer

2. Size of editing buffer

In implementing the function, you must use pointer notation in traversing the buffer.

Save the function implementation in `editor2.c`. You may implement other functions (needed by your implementation) in `editor2.c`.

Task 8.

Challenge [7.5 Marks]

In this task, you will implement a function that will search for multiple occurrences of a given search string from a given editing buffer. The search string is a null-terminated array of characters, and is passed to the functions using pointer notation. The editing buffer is an array of characters, and is passed to the function using pointer notation. The size of the buffer should also be passed to the function. The function will not modify the search string and contents of the buffer, and this should be indicated in the function header (and prototype).

Two other arguments are passed to the function: an array of integers (passed using pointer notation) and the size of the integer array. This array will be used for storing the *positions* of the found occurrences of search string in the buffer. Position refers to the index of the first letter of the search string in the buffer.

To clarify the use of the integer array, suppose that the search string is found at positions 10, 24, 56, and 100, then the values of integer array elements at indices 0, 1, 2, and 3 should be 10, 24, 56, and 100, respectively. Note that the function will only be able to search at most K occurrences, where K is the size of the integer array.

The function should be named `editor_search_multiple`. It must return the number of elements in the integer array which has valid positions. In the example above where the search string is found at positions 10, 24, 56, and 100, the function must return 4.

The arguments should be ordered as follows:

1. Editing buffer
2. Size of editing buffer
3. String to be searched

4. Integer array
5. Size of integer array

In implementing the function, you must use pointer notation in traversing the search string, buffer and integer array.

Save the function implementation in `editor2.c`. You may implement other functions (needed by your implementation) in `editor2.c`.

Part III: C++ Programming

You may use the C++ Standard Library to perform the tasks in this part.

Task 9.

Basics [15 Marks]

In this task, you will create a C++ header file that will contain declaration of a class for implementing Tasks 10 and 11. You may declare constants and user-defines types needed by your implementations.

The class should be named `EditorUtilities` and should be defined within `editor2` namespace.

Save the header file as `editor2.hh`.

Task 10.

Completion [15 Marks]

In this task, you will implement a member function of `EditorUtilities` for counting the number of words in a given editing buffer. The member function should be public and callable even without the instance of `EditorUtilities`.

The editing buffer is an array of characters, and is passed to the function using pointer notation. The size of the buffer should also be passed to the function. The function will not modify the contents of the buffer, and this should be indicated in the member function header (and prototype).

The function should be named `countWords`. It must return the number of words in the buffer. A *word* is defined as consisting of non-whitespace characters. Words are separated by whitespace charaters. In C/C++, the whitespace characters include:

- Space: `' '`
- Horizontal tab: `'\t'`
- Newline: `'\n'`
- Vertical tab: `'\v'`
- Form feed: `'\f'`
- Carriage return: `'\r'`

The arguments should be ordered as follows:

1. Editing buffer
2. Size of editing buffer

In implementing the function, you must use pointer notation in traversing the buffer.

Save the implementation in `editor2.cc`. You may implement other member functions (needed by your implementation) in `editor2.cc`.

Task 11.

Challenge [7.5 Marks]

In this task, you will implement a member function of `EditorUtilities` that will search for multiple occurrences of a given search string from a given editing buffer, **ignoring case**. The member function should be public and callable even without the instance of `EditorUtilities`.

This is a variation of Task 6, hence, all the points in that task applies to this as well, except for the following:

- When searching for the occurrence of search string, case should be ignored. This means that if the search string is `"Hello"`, then `"Hello"`, `"hello"`, `"heLLo"`, etc. are considered as occurrences.
- The search string argument is of type `std::string`, not an array of characters.
- The member function should be named `searchMultipleNoCase`.

In implementing the function, you must use pointer notation in traversing the search string, buffer and integer array.

Save the implementation in `editor2.cc`. You may implement other member functions (needed by your implementation) in `editor2.cc`.

Marking Criteria for Tasks 7, 8, 9, and 11:

Criteria	Weight	Expectations for Full Marks
"Compilability"	10%	Source code compiles without warnings
Commenting	10%	Source code contains sufficient and appropriate comments
Coding Style	10%	Source code is formatted, readable and uses a coding style consistently
Use of Pointers	30%	Uses pointers correctly and as specified
Correctness	40%	Handles all possible cases correctly
	100%	

Marking Criteria for Tasks 6 and 9:

Criteria	Weight	Expectations for Full Marks
Commenting	10%	Source code contains sufficient and appropriate comments
Coding Style	10%	Source code is formatted, readable and uses a coding style consistently
Correctness	40%	Addresses all specifications and correctly uses syntax in the declarations and/or definitions
Completeness	30%	Declaration and/or definition of all required functions (in Task 6) and class and member functions (in Task 9)
	100%	