Assignment 6
C/C++ Programming I

C1A6 General Information

Assignment 6 consists of FOUR (4) exercises:

C1A6E0 C1A6E1 C1A6E2 C1A6E3

All requirements are in this document.

Related examples are in a separate file.

Get a Consolidated Assignment 6 Report (optional)

If you would like to receive a consolidated report containing the results of the most recent version of each exercise submitted for this assignment:

`Send an empty-body email to the assignment checker with the subject line **C1A6_174273_U09845800** and no attachments.

Inspect the report carefully since it is what I will be grading. You may resubmit exercises and report requests as many times as you wish before the assignment deadline.

C1A6 General Information, continued

What is *size_t* ?

size_t (note 2.12) is an implementation-defined data type typically used to count things related to memory, such as the amount of storage needed for an object or the number of objects available or to be processed. The actual type that **size_t** represents is at the discretion of the compiler designer and may be any of **unsigned char**, **unsigned short**, **unsigned int**, **unsigned long**, or **unsigned long long**. Any assumption on the part of the applications programmer about which one of these types it is, is not portable.

"Lexicographical" String Comparison

A string (C-string) is defined as a sequence of characters ending with a null character, `'\0'`. Two strings are considered equal only if they are the same length and their corresponding characters are equal. The term "corresponding characters" refers to the first character in one string compared to the first character in the other string, the second character in one string compared to the second character in the other string, etc.

The longer string is not necessarily the greater. Instead, the greater is determined entirely by the relative numeric values of the first two non-equal corresponding characters. For example, the greatest of strings `"Heat"` and `"Hi"` is `"Hi"` because the second two corresponding characters (`'e'` and `'i'`) are not equal and the value of `'i'` is greater than that of `'e'` (ASCII character set assumed). This method of comparing strings is known as "lexicographical" (dictionary) comparison.

Inputting an Entire (Possibly Empty) User Line in C
Eliminating the `fgets` *newline* Character

All "C-style" strings end with the null character, `'\0'`. An "empty" string contains **only** that character. Because of the sometimes-inconsistent behavior of the `scanf` function between different compilers, I recommend against using it to read empty user input lines. Instead, both empty and non-empty lines can be read reliably using the `fgets` function in C and the `getline` function in C++.

`getline` discards the newline character that terminates an input line but `fgets` keeps it. The simplest way to eliminate an unwanted newline character is to overwrite it with a null character using the standard library `strcspn` function technique illustrated below:

```
char buffer[BUF_SIZE];
fgets(buffer, BUF_SIZE, stdin);
buffer[strcspn(buffer, "\n")] = '\0';
```

How it works: The `strcspn` function searches the string in its first argument for the first of any characters in the string in its second argument, including each string's terminating null character. It then returns the index in the first string of the first character it found from the second string. For example,

```
strcspn(buffer, "\n")
```

searches `buffer` for the newline and null characters and returns the index in `buffer` of which character it found first. Thus,

```
buffer[strcspn(buffer, "\n")] = '\0';
```

will overwrite the first newline or null character in `buffer` with a null character.

C1A6E0 *(6 points total - 1 point per question – No program required)*

Assume language standards compliance and any necessary standard library support unless stated otherwise. These are not trick questions and there is only one correct answer. Basing an answer on actual testing is risky. Place your answers in a plain text "quiz file" named **C1A6E0_Quiz.txt** formatted as:

a "Non-Code" Title Block, an empty line, then the answers:
1. A
2. C
etc.

1. For ***char*** *ix[20];* which one does not access *element 19* or is syntactically incorrect? (Note 6.16)
   A. *(19 + ix)
   B. *ix + 19
   C. *(ix + 18 + **sizeof(char**))
   D. (9 + 10)[ix]
   E. ix[2 * 10 - 1]

2. If **char**s are 8 bits and **int**s are 48 bits, predict the value of *p* after:
   ```
   int *p = (int *)20;
   ++p;
   ```
   (Notes 6.14, 6.16)
   A. 24
   B. 25
   C. 26
   D. 27
   E. none of above

3. The following is intended to display *12345* What is the most serious problem?
   ```
   short array[7] = { 1, 2, 3, 4, 5 };
   short *ps, *ptr = array;
   for (*ps = 0; *ps < 5; (*ps)++)
       printf("%hd ", *ptr++);
   ```
   (Note 6.6)
   A. *ps* is a "null pointer".
   B. Possible runtime error with **ps = 0* or **ps < 5* or **ps++*
   C. The array is too large.
   D. *%hd* in the *printf* should be *%d* or *%i*
   E. **ptr++* should be ***(short)***ptr++*

4. For ***int*** *c[3];* the data types passed to function *test* by *test(&c, c)* left-to-right are: (Note 6.16)
   A. "reference to **int**", "array of 3 **int**s"
   B. "address of **int**", "array of 3 **int**s"
   C. "pointer to **int**", "array of 3 **int**s"
   D. "pointer to array of 3 **int**s", "pointer to **int**"
   E. "reference to **int**", "pointer to **int**"

5. Predict the output if *Little Orphan Annie* is entered by the user:
   ```
   char a[32], b[32], c[32];
   int x =
     scanf("\n%31[^t]\n%31[a-z ] %31s",
         a, b, c);
   printf("%d %s%s%s", x, a, b, c);
   ```
   (Note 7.3)
   A. *3 Little Orphan Annie*
   B. *3 Little*
   C. *3 Little Orphan*
   D. *3 Orphan*
   E. implementation dependent output

6. Predict the output:
   ```
   char test[] = "Strange characters";
   string bf("Length");
   bf += " o" + "f this is: ";
   string bf2(test + 6, test + 16);
   cout << bf << bf2.length() << ' ' << bf2;
   ```
   (Notes 6.14, 6.16, 7.1, & 7.7)
   A. *Length of this is: 10 characters*
   B. *Length of this is: 10 Strange characters*
   C. garbage
   D. implementation dependent output
   E. none – adding 2 pointers won't compile.

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C1A6E0_174273_U09845800** and with your quiz file attached.

See the course document titled "How to Prepare and Submit Assignments" for additional exercise formatting, submission, and assignment checker requirements.

C1A6E1 *(4 points – C Program)*

Exclude any existing source code files that may already be in your IDE project and add two new ones, naming them **C1A6E1_MyStrlen.c** and **C1A6E1_main.c**. Do not use `#include` to include either of these files in each other or in any other file. However, you may use it to include any appropriate header file(s) you need.

File **C1A6E1_MyStrlen.c** must contain a function named `MyStrlen` that has the same syntax and functionality as the standard library `strlen` function. If you are not familiar with `strlen` look it up in your IDE's help, the course book, any good C textbook, or online. `MyStrlen` must:

1. have the syntax (prototype): `size_t MyStrlen(const char *s1);`
2. return a count of the number of characters in the string in `s1`, not including the null terminator.
3. use only one variable other than its formal parameter `s1`. That variable must be of type "**const** pointer to **const char**" and must be initialized to the value of formal parameter `s1` when declared, for example:
   ```
   const char * const START = s1;
   ```
4. not assign anything to `s1` (do not do: `s1 = something`), but you may increment it.
5. not call any functions, use any macros, or display anything.
6. not use the **sizeof** operator (it would not help anyway).

File **C1A6E1_main.c** must contain function `main`, which must:

1. prompt the user to enter a string (which may be empty or contain spaces).
2. call `strlen` to determine that string's length.
3. call `MyStrlen` to determine that string's length.
4. display the string and its length as determined by both `strlen` and `MyStrlen` in the following 2-line format, where `ABC` is the string used in this example and where the question marks represent the integral decimal numeric values returned by the functions. By sure to enclose the string in double-quotes:
   ```
   strlen("ABC") returned  ?
   MyStrlen("ABC") returned ?
   ```

Manually re-run your program several times, testing with at least the following 4 strings (the last string is empty):

1. `a`
2. `HELLO`
3. `C/C++ Programming I`
4. *(an empty string)*

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C1A6E1_174273_U09845800** and with both source code files attached.

See the course document titled "How to Prepare and Submit Assignments" for additional exercise formatting, submission, and assignment checker requirements.

Hints:

Do you know what **size_t** is? If not consider reviewing note 2.12. Do you know what pointer subtraction is? If not consider reviewing note 6.14. For an example of using a pointer to walk through a string see notes 6.17 and 7.2. No special case is needed for an empty string. Set the extra pointer variable you are allowed to declare equal to the parameter pointer variable then increment one of these pointers as you step through the input string looking for the null terminator character, `'\0'`. When you find it, subtract the two pointers to find the string length and return that difference. Type cast the return expression to **size_t** to avoid a compiler warning. Most library functions that compute values, including `strlen`, do no printing.

C1A6E2 *(4 points – C Program)*

Exclude any existing source code files that may already be in your IDE project and add two new ones, naming them **C1A6E2_MyStrcmp.c** and **C1A6E2_main.c**. Do not use `#include` to include either of these files in each other or in any other file. However, you may use it to include any appropriate header file(s) you need.

File **C1A6E2_MyStrcmp.c** must contain a function named `MyStrcmp` that has the same syntax and functionally as the standard library `strcmp` function. If you are not familiar with `strcmp` look it up in your IDE's help, the course book, any good C textbook, or online. `MyStrcmp` must:

1. Have the syntax (prototype): `int MyStrcmp(const char *s1, const char *s2);`
2. Return:
    a. any value < 0 if the string in `s1` is lexicographically less than the string in `s2`.
    b. 0 if the string in `s1` is equal to the string in `s2`.
    c. any value > 0 if the string in `s1` is lexicographically greater than the string in `s2`.

   The values returned by `strcmp` and `MyStrcmp` do not have to be the same for non-equal strings.
3. Not use any variables other than its two formal parameters `s1` and `s2`.
4. Not call any functions, use any macros, or display anything.
5. Not use the **sizeof** operator (it would not help anyway).

File **C1A6E2_main.c** must contain function `main`, which must:

1. Use two separate user prompts to obtain two strings (both of which may be empty or contain spaces).
2. Call `strcmp` to compare the two strings.
3. Call `MyStrcmp` to compare the two strings.
4. Display the relationship between the two strings as determined by both `strcmp` and `MyStrcmp` in the following 2-line format, where `ABCXYZ` and `DEF` are the strings in this example and where the question marks represent the integral decimal numeric values returned by the functions. Be sure to enclose the strings in double-quotes:

```
strcmp("ABCXYZ", "DEF") returned ?
MyStrcmp("ABCXYZ", "DEF") returned ?
```

Manually re-run your program several times, testing with at least the following 4 string pairs (the last pair consists of two empty strings):

1. `a` and `B`
2. `HE` and `HELLO`
3. `HE` and `EHLLO`
4. *(an empty string)* and *(an empty string)*

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C1A6E2_174273_U09845800** and with both source code files attached.

See the course document titled "How to Prepare and Submit Assignments" for additional exercise formatting, submission, and assignment checker requirements.

Hints:

See note 7.2 for an example of using a pointer to walk through a string and note 7.6 for some examples of string comparisons. No special case is needed for empty strings. The value obtained by subtracting the values of the two characters currently being compared is the most straightforward value to return when a return is required. Merely simultaneously step through both strings character-at-a-time, comparing the corresponding characters in each. Return when the first pair of non-equal characters is encountered or when a null terminator character, `'\0'`, is reached in either string.

C1A6E3 *(6 points – C Program)*

Exclude any existing source code files that may already be in your IDE project and add two new ones, naming them **C1A6E3_GetSubstring.c** and **C1A6E3_main.c**. Do not use `#include` to include either of these files in each other or in any other file. However, you may use it to include any appropriate header file(s) you need.

File **C1A6E3_GetSubstring.c** must contain a function named `GetSubstring` whose purpose is to create a new string of characters by copying them from an existing string. Its syntax (prototype) is:

```
char *GetSubstring(const char source[], int start, int count, char result[]);
```

where `source` represents the string from which to copy the characters, `start` is the index in `source` of the first character to copy, `count` is the number of characters to copy, and `result` represents an array into which the characters are to be copied. For example, if the string in `source` is **investments**, the start index is `2`, and the character count is `4`, the characters `vest` will be copied from `source` into `result` and a `'\0'` will be appended.

Function `GetSubstring` must:

1. Handle the following three situations:
   a. If `start` is within the string in `source` and `count` does not extend beyond the end of it:
      Copy `count` characters into the `result` array and append a `'\0'`.
   b. If `start` is within the string in `source` but `count` does extend beyond the end of it:
      Copy all characters remaining in `source` into the `result` array and append a `'\0'`.
   c. If `start` is beyond the end of the string in `source`:
      Store only a `'\0'` in the `result` array.
2. Return a pointer to the first element of the `result` array.
3. Use only one variable other than formal parameters `source`, `start`, `count`, and `result;` it must be an automatic variable of type "pointer to **char**".
4. Not call any functions, use any macros, or display anything.
5. Not use the **sizeof** operator (it would not help anyway).
6. Not use index or pointer offset expressions like `pointer[i]` and `*(pointer + i)`. Compact or moving pointer expressions like `*pointer++` and `pointer++` are more appropriate. If you have trouble with this it may help to write the program using index notation first, then convert to compact or moving pointers.

File **C1A6E3_main.c** must contain function `main`, which:

1. Prompts the user to enter a sequence of 0 or more arbitrary printable characters (which may include spaces) then stores them as a string in a 256-element character array named `source`.
2. Prompts the user again to enter a space-separated start index and character count on the same line then stores them in type **int** variables named `start` and `count`, respectively.
3. Calls `GetSubstring(source, start, count, result)`, where `result` is a 256-element character array, and displays the results of the extraction in the following format, where the quotes, commas, and the literal word **`extracts`** are all required. This output is for the example provided in the description of the `GetSubstring` function above:

```
"investments", 2, 4, extracts "vest"
```

The pointer returned by `GetSubstring`, not the `result` array itself, must be used to display the extracted substring.

Test your program several times, using at least the 7 user entry sets shown on the next page.

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C1A6E3_174273_U09845800** and with both source code files attached.

See the course document titled "How to Prepare and Submit Assignments" for additional exercise formatting, submission, and assignment checker requirements.

Recommended Program Test Sets

source	start	count	You should get
This is really fun	2	800	is is really fun
This is really fun	261	9	*(an empty string)*
This is really fun	0	12	This is real
one two three	5	87	wo three
one two three	18	7	*(an empty string)*
one two three	6	5	o thr
one two three	0	3	one
(an empty string)	3	23	*(an empty string)*

Hints:

All hints assume the prototype for `GetSubstring` is:

```
char *GetSubstring(const char source[], int start, int count, char result[]);
```

- Remember that for parameter declarations only the forms **type** `name[]` and **type** `*name` are functionally equivalent and both mean "`name` is a pointer to **type**".

- A common error is to mistakenly return a pointer to the end of the extracted substring.

- If you are doing `source + start` or `start + count` you are on the wrong track.

- An optimal solution for `GetSubstring` will contain the statement `*result++ = *source++;`

- The following 2-loop algorithm is recommended but not required. Note that the loops are not nested. If it is not clear what this algorithm is doing you should draw a step-by-step diagram:
  1. Save a copy of `result`.
  2. **First loop**
     Loop through each successive character in `source` until either the end of the string is found or the offset specified by `start` is reached. Increment `source` and decrement `start` as you proceed as appropriate.
     End first loop
  3. **Second loop**
     Copy successive characters from `source` (as updated in step 2) into `result` until the null terminator character is reached (don't copy it) or until `count` characters have been copied, whichever comes first. Increment `source` and `result` as you proceed as appropriate.
     End second loop
  4. Copy a null terminator character, `'\0'`, into `*result`.
  5. Return the copy of the original value of `result` saved in step 1.

