Assignment 6
C/C++ Programming I

1
2
3
4 **C1A6 General Information**

5
6
What is *size_t* ?

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

13
"Lexicographical" String Comparison

14 A string is defined as a sequence of characters ending with a null terminator character. Two strings are
15 considered equal if they are the same length and all of their corresponding characters are equal. The
16 term "corresponding characters" refers to the 1st character in one string compared to the 1st character
17 in the other string, the 2nd character in one string compared to the 2nd character in the other string,
18 etc. Two strings are not equal if they are different lengths or if any corresponding characters are not
19 equal. For non-equal strings the string considered to be greater is not necessarily the longer string.
20 Instead, which is greater is determined entirely by the relative numeric values of the first two
21 corresponding characters that are not equal. For example, the greatest of strings "Heat" and "Hi" is "Hi"
22 because the second two corresponding characters ('e' and 'i') are not equal and the value of 'i' is
23 greater than the value of 'e' (ASCII character set assumed). This method of comparing strings is known
24 as "lexicographical" (dictionary) comparison.
25
26
Inputting an Entire (Possibly Empty) User Line in C

27 All "C-style" strings end with the null terminator character, `'\0'`, and an "empty" string contains **only**
28 that character. Because of the sometimes inconsistent behavior of the `scanf` function between
29 compilers I recommend against using it to read empty user input lines. Instead, both empty and non-
30 empty user lines can be reliably read using the `fgets` function. `fgets` reads and stores an entire input
31 line as a string, including the newline character that terminates that line. Often, however, that newline
32 character is not wanted and must be removed, and a common way to do it is to simply overwrite it.
33 Because there are occasionally cases in which there is not a newline character this must be taken into
34 consideration. The simplest way to overwrite the newline character, if any, is to use the standard library
35 `strcspn` function as illustrated below. I recommend that you look it up and learn how it works rather
36 than simply using it:

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

40
41 In C++ the `getline` function should be used instead of `fgets`. It has the advantage of automatically
42 discarding the newline character, thereby eliminating the need to remove it manually.

43
44
45 **Get a Consolidated Assignment 6 Report (optional)**
46 If you would like to receive a consolidated report containing the results of the most recent version of
47 each exercise submitted for this assignment:
48 `Send an empty-body email to the assignment checker with the subject line **C1A6_162461_U09339367**
49 and no attachments.
50 Inspect the report carefully since it is what I will be grading. You may resubmit exercises and report
51 requests as many times as you wish before the assignment deadline.

Page 1 of 1 of C1A6 General Information

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 <u>are not</u> trick questions and there is only one correct answer, but basing an answer on runtime results 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. The following is intended to display *12345* What is the most serious problem?

   ```
   int arr[5] = { 1, 2, 3, 4, 5 };
   int *p, *pa = arr;
   for (*p = 15; *p > 10; --*p)
       cout << *pa++;
   ```
   (Note 6.6)
   A. A garbage address is accessed.
   B. The syntax --*p is not portable.
   C. *pa++ should be *(pa++)
   D. *pa++ is not valid in C++
   E. There is no serious problem.

2. If **char**s are 8 bits and **int**s are 24 bits, predict the value of *p* after:

   ```
   int *p = (int *)20;
   ++p;
   ```
   (Note 6.14)
   A. 22
   B. 23
   C. 24
   D. 25
   E. none of above

3. For **int** *a[9], *p = a;* the data types passed to function *f* by *f(a, &p)* left-to-right are:
   (Note 6.16)
   A. "array of 9 **int**s", "address of **int**"
   B. "array of 9 **int**s", "pointer to **int**"
   C. "pointer to **int**", "pointer to array of 9 **int**s"
   D. "pointer to **int**", "pointer to pointer to **int**"
   E. "pointer to array of 9 **int**s", "pointer to **int**"

4. For **float** *ax[29];* which one does not access element 6 or is syntactically incorrect?
   (Notes 6.14, 6.16)
   A. *(printf("syntax") + ax)
   B. *(ax + 6)
   C. ax[2 * 2 + 2]
   D. 6[ax]
   E. (*(ax) + 6)

5. Predict the output:

   ```
   char test[] = "weird characters";
   string buf("Character");
   buf += " cou" + "nt is: ";
   string buf2(test + 6, test + 16);
   cout << buf << buf2.size() << ' ' << buf2;
   ```
   (Notes 6.14, 6.16, 7.1, & 7.7)
   A. none – adding 2 pointers won't compile.
   B. *Character count is: 10 characters*
   C. *Character count is: 10 weird characters*
   D. *count is:*
   E. implementation dependent output

6. Predict the output if <u>*this code is slower*</u> is entered by the user:

   ```
   char a[32], b[32], c[32];
   int x =
   scanf("\nt%31[^e]\n%31[a-s ]\n%31[^e]",
       a, b, c);
   printf("%d %s%s%s", x, a, b, c);
   ```
   (Note 7.3)
   A. *3 his code is slower*
   B. *3 this code is slower*
   C. *3 his code is slow*
   D. garbage
   E. implementation dependent output

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C1A6E0_162461_U09339367** and with your quiz file <u>attached</u>.

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

1 **C1A6E1 *(4 points – C Program)***

2 Exclude any existing source code files that may already be in your IDE project and add two new ones,
3 naming them **C1A6E1_MyStrlen.c** and **C1A6E1_main.c**. <u>Do not</u> use **#include** to include either of these
4 files in each other or in any other file. However, you may use it to include any appropriate header file(s)
5 you need.
6
7 File **C1A6E1_MyStrlen.c** must contain a function named **MyStrlen** that has the same syntax and
8 functionality as the standard library **strlen** function. If you are not familiar with **strlen** look it up in
9 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 <u>only one variable</u> 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. <u>not</u> assign anything to **s1** (do not do: **s1** = *something*), but you may increment it.
   5. <u>not</u> call any functions, use any macros, or display anything.
   6. <u>not</u> use the **sizeof** operator (it would not help anyway).

20 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 ?

31 Manually re-run your program several times, testing with at least the following 4 strings (the last string is
32 empty):
   1. a
   2. HELLO
   3. C/C++ Programming I
   4. *(an empty string)*

Submitting your solution

40 `Send an empty-body email to the assignment checker with the subject line **C1A6E1_162461_U09339367**
41 and with both source code files <u>attached</u>.

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

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

1 ## C1A6E2 *(4 points – C Program)*

2 Exclude any existing source code files that may already be in your IDE project and add two new ones,
3 naming them **C1A6E2_MyStrcmp.c** and **C1A6E2_main.c**. <u>Do not</u> use `#include` to include either of these
4 files in each other or in any other file. However, you may use it to include any appropriate header file(s)
5 you need.
6
7 File **C1A6E2_MyStrcmp.c** must contain a function named `MyStrcmp` that has the same syntax and
8 functionally as the standard library `strcmp` function. If you are not familiar with `strcmp` look it up in your
9 IDE's help, the course book, any good C textbook, or online. `MyStrcmp` must:
10   1. Have the syntax (prototype): **int MyStrcmp(const char *s1, const char *s2);**
11   2. Return:
12   a. <u>any</u> value < 0 if the string in **s1** is lexicographically less than the string in **s2**.
13   b. 0 if the string in **s1** is equal to the string in **s2**.
14   c. <u>any</u> value > 0 if the string in **s1** is lexicographically greater than the string in **s2**.
15   The values returned by `strcmp` and `MyStrcmp` do not have to be the same for non-equal strings.
16   3. <u>Not</u> use any variables other than its two formal parameters **s1** and **s2**.
17   4. <u>Not</u> call any functions, use any macros, or display anything.
18   5. <u>Not</u> use the **sizeof** operator (it would not help anyway).
19
20 File **C1A6E2_main.c** must contain function `main`, which must:
21   1. Use two separate user prompts to obtain two strings (both of which may be empty or contain
22   spaces).
23   2. Call `strcmp` to compare the two strings.
24   3. Call `MyStrcmp` to compare the two strings.
25   4. Display the relationship between the two strings as determined by both `strcmp` and `MyStrcmp` in
26   the following 2-line format, where `ABCXYZ` and `DEF` are the strings in this example and where the
27   question marks represent the integral decimal numeric values returned by the functions. Be sure
28   to enclose the strings in double-quotes:
29
```
        strcmp("ABCXYZ", "DEF") returned ?
        MyStrcmp("ABCXYZ", "DEF") returned ?
```
30
31
32 Manually re-run your program several times, testing with at least the following 4 string pairs (the last pair
33 consists of two empty strings):
34   1. `a` and `B`
35   2. `HE` and `HELLO`
36   3. `HE` and `EHLLO`
37   4. *(an empty string)* and *(an empty string)*
38
39
40 ## Submitting your solution

41 `Send an empty-body email to the assignment checker with the subject line **C1A6E2_162461_U09339367**
42 and with both source code files <u>attached</u>.

43 *See the course document titled "How to Prepare and Submit Assignments" for additional exercise*
44 *formatting, submission, and assignment checker requirements.*
45
46
47 **Hints:**
48 See note 7.2 for an example of using a pointer to walk through a string and note 7.6 for some examples
49 of string comparisons. No special case is needed for empty strings. The value obtained by subtracting
50 the values of the two characters currently being compared is the most straightforward value to return
51 when a return is required.   Merely simultaneously step through both strings character-at-a-time,
52 comparing the corresponding characters in each. Return when the first pair of non-equal characters is
53 encountered or when a null terminator character, `'\0'`, is reached in either string.

© 1992-2021 Ray Mitchell   Page 1 of 1 of C1A6E2

1 **C1A6E3** *(6 points – C Program)*

2 Exclude any existing source code files that may already be in your IDE project and add two new ones,
3 naming them **C1A6E3_GetSubstring.c** and **C1A6E3_main.c**. <u>Do not</u> use **#include** to include either of
4 these files in each other or in any other file. However, you may use it to include any appropriate header
5 file(s) you need.
6
7 File **C1A6E3_GetSubstring.c** must contain a function named `GetSubstring` whose purpose is to create
8 a new string of characters by copying them from an existing string. Its syntax (prototype) is:

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

10 where `source` represents the string from which to copy the characters, `start` is the index in `source` of
11 the first character to copy, `count` is the number of characters to copy, and `result` represents an array
12 into which the characters are to be copied. For example, if the string in `source` is **investments**, the start
13 index is **2**, and the character count is **4**, the characters `vest` will be copied from `source` into `result`
14 and a `'\0'` will be appended.
15
16 Function `GetSubstring` must:
17   1.   Handle the following three situations:
18   a. If `start` is within the string in `source` and `count` does not extend beyond the end of it:
19   Copy `count` characters into the `result` array and append a `'\0'`.
20   b. If `start` is within the string in `source` but `count` does extend beyond the end of it:
21   Copy all characters remaining in `source` into the `result` array and append a `'\0'`.
22   c. If `start` is beyond the end of the string in `source`:
23   Store only a `'\0'` in the `result` array.
24   2.   Return a pointer to the first element of the `result` array.
25   3.   Use only <u>one</u> variable other than formal parameters `source`, `start`, `count`, and `result`; it
26   must be an automatic variable of type "pointer to **char**".
27   4.   <u>Not</u> call any functions, use any macros, or display anything.
28   5.   <u>Not</u> use the **sizeof** operator (it would not help anyway).
29   6.   <u>Not use index or pointer offset expressions</u> like `pointer[i]` and `*(pointer + i)`. Compact or
30   moving pointer expressions like `*pointer++` and `pointer++` are more appropriate. If you have
31   trouble with this it may help to write the program using index notation first, then convert to
32   compact or moving pointers.
33
34 File **C1A6E3_main.c** must contain function `main`, which:
35   1.   Prompts the user to enter a sequence of 0 or more arbitrary printable characters (which may
36   include spaces) then stores them as a string in a 256-element character array named `source`.
37   2.   Prompts the user again to enter a space-separated start index and character count on the
38   same line then stores them in type **int** variables named `start` and `count`, respectively.
39   3.   Calls `GetSubstring(source, start, count, result)`, where `result` is a 256-element
40   character array, and displays the results of the extraction in the following format, where the
41   quotes, commas, and the literal word **extracts** are all required. This output is for the example
42   provided in the description of the `GetSubstring` function above:

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

44 <u>The pointer returned by `GetSubstring`, not the `result` array itself, must be used to display the</u>
45 <u>extracted substring.</u>
46
47 Test your program several times, using at least the 7 user entry sets shown on the next page.
48
49 **Submitting your solution**

50 `Send an empty-body email to the assignment checker with the subject line **C1A6E3_162461_U09339367**
51 and with both source code files <u>attached</u>.

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

---

1 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 <u>for parameter declarations only</u> 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.

Page 2 of 2 of C1A6E3