Ray Mitchell, U99999999
MeanOldTeacher@MeanOldTeacher.com
C/C++ Programming I
Section 146359, Ray Mitchell
June 25, 2019
C1A6E0_Quiz.txt
Quiz Answers

1. D
2. D
3. C
4. B
5. E
6. A

1 **C1A6E0 Explanations**
2
3 In addition to the course book references cited below, these topics are also covered in the live lectures
4 (in-class students) and the recorded lectures (online students).
5
6   1. **D** Notes 6.14, 6.16, 7.1, & 7.7; The expression *buf += " of th" + "is is: "* contains two string literals.   A
7   string literal is merely a character array containing a sequence of constant characters ending
8   with the null character that is automatically placed there by the compiler. As with any array, a
9   string literal will decay to a pointer to its first element if not used in one of the special non-
10   decaying situations, which this is not. Thus, the two string literals each decay to type "pointer to
11   **const char**".   Since the code is attempting to add them together but the addition of two
12   pointers is not one of the three legal arithmetic operations that may be performed on pointers,
13   the compiler will generate an error and will not compile the code.
14
15   2. **D** Notes 1.7, 6.14; The value of any legal expression having the form *++x* is the same as the value
16   of the expression *x + 1*. If *x* has an object pointer type the sum is not necessarily the result of
17   simply adding the two operands arithmetically. Instead, it is obtained by adding the pointer's
18   value to the value of the product of the integral expression's value and the number of bytes in
19   the type pointed to by the pointer. In the example below, where *sizeof(**int**)* has arbitrarily been
20   defined to be 5, the sum is equal to 25:
21   20 + (1 * sizeof(int))   ==   20 + (1 * 5)   ==   20 + 5   ==   25
22
23   3. **C** Notes 6.14, 6.16; The expressions in four of the answers are in one of the equivalent forms *p[i]*,
24   *i[p]*, **(p + i)*, and **(i + p)*, where *p* represents an expression having an object pointer type and *i*
25   represents an expression having an integer type. In this quiz question *p* is represented by *bc* and
26   the total value of the integer part is *8*. (Remember that addition is commutative.)
27
28   4. **B** Note 5.11; *p* is an uninitialized automatic variable and as such represents a garbage memory
29   address. If you dereference a garbage memory address and are extremely lucky it will result in
30   a program crash or some other repeatable runtime error. But if you are extremely unlucky it will
31   appear to work correctly.
32
33   5. **E** Notes 6.1, 6.5, 6.16; By the Right-Left Rule identifier *p* is declared as type "array of 9 **int**s". Any
34   expression that has an array type and is not used in one of the special non-decaying situations is
35   automatically converted to a pointer to the first element of that array. In the first argument *p* is
36   not being used in one of those special situations so its data type is converted to a "pointer to an
37   **int**". In the second argument *p* is being used in one of those special situations so its data type
38   remains unchanged. Thus, the second argument is a "pointer to an array of 9 **int**s".
39
40   6. **A** Note 7.3; The input line is: one two three
41   a. The "\n" in the *scanf* control string reads and throws away characters from the remaining
42   input string up to but not including the first character that is not a whitespace. That
43   character is the "*t*" in "*two*".
44   b. The "*%31[a-t]*" in the *scanf* control string reads characters from the remaining input string
45   up to but not including the first character that is not in the range "*a-t*" and is not a
46   space, up to a maximum of 31 characters. All characters read are stored in array *a* and
47   terminated with the null character, '\0'. Thus, the string "*t*" is stored in array *a*.
48   c. The "\n" in the *scanf* control string reads and throws away characters from the remaining
49   input string up to but not including the first character that is not a whitespace. That
50   character is "*w*".
51   d. The "*%31[^t]*" in the *scanf* control string reads characters from the remaining input string
52   up to but not including the first character that is a "*t*", up to a maximum of 31 characters.
53   All characters read are stored in array *b* and terminated with the null character, '\0'.
54   Thus, the string "*wo* " is stored in array *b*.

1   e. The *" "* (blank space) in the *scanf* control string reads and throws away characters from
2   the remaining input string up to but not including the first character that is not a
3   whitespace. That character is the *"t"* in *"three"*.

4   f. The *"%31s"* in the *scanf* control string reads and throws away characters from the
5   remaining input string up to but not including the first character that is not a whitespace,
6   up to a maximum of 31 characters. That character is the *"t"* in *"three"*. It then reads
7   characters from the remaining input string up to but not including the first whitespace.
8   All characters read are stored in array *c* and terminated with the null character, *'\0'*.
9   Thus, the string *"three"* is stored in array *c*.

10   g. The *scanf* function returns a count of the number of its arguments for which input values
11   were successfully matched assigned, which in this case is 3 (*a*, *b*, and *c*). Thus, *3* is
12   assigned into variable *x*.

13   h. Finally, the *printf* prints the value of variable *x*, followed by a blank space, followed by
14   the strings stored in arrays *a*, *b*, and *c*, respectively, with no blank space between those
15   strings. The complete output is: *3 two three*

```c
1    //
2    // Ray Mitchell, U99999999
3    // MeanOldTeacher@MeanOldTeacher.com
4    // C/C++ Programming I
5    // Section 146359, Ray Mitchell
6    // June 25, 2019
7    // C1A6E1_main.c
8    // Windows 10 Professional
9    // Visual Studio 2019 Professional
10   //
11   // This file contains function main, which determines string lengths by
12   // calling the strlen and MyStrlen functions.
13   //
14
15   #include <stdio.h>
16   #include <stdlib.h>
17   #include <string.h>
18
19   #define MAX_SIZE 256 // input buffer size
20
21   size_t MyStrlen(const char *s1);
22
23   //
24   // Function main displays the results of calling the strlen and MyStrlen
25   // functions to determine the lengths of user input strings.
26   //
27   int main(void)
28   {
29       char s1[MAX_SIZE];
30       int length;
31
32       printf("Enter a string: ");
33       fgets(s1, MAX_SIZE, stdin);
34       length = (int)strlen(s1);
35       if (length > 0 && s1[length - 1] == '\n')
36           s1[--length] = '\0';                 // eliminate trailing newline
37
38       // Display messages describing the relationship between the strings.
39       printf("strlen(\"%s\") returned %d\n", s1, length);
40       printf("MyStrlen(\"%s\") returned %d\n", s1, (int)MyStrlen(s1));
41
42       return EXIT_SUCCESS;
43   }
```

```c
 1    //
 2    // Ray Mitchell, U99999999
 3    // MeanOldTeacher@MeanOldTeacher.com
 4    // C/C++ Programming I
 5    // Section 146359, Ray Mitchell
 6    // June 25, 2019
 7    // C1A6E1_MyStrlen.c
 8    // Windows 10 Professional
 9    // Visual Studio 2019 Professional
10    //
11    // This file contains function MyStrlen, which duplicates the syntax and
12    // functionality of the standard library strlen function.
13    //
14
15    #include <stddef.h>
16    //
17    // Function MyStrlen duplicates the syntax and functionality of the standard
18    // library strlen function.  Specifically, a count of the number of characters
19    // in the string represented by pointer <s1> (not including the null terminator
20    // character) is returned.
21    //
22    size_t MyStrlen(const char *s1)
23    {
24        const char * const START = s1;
25
26        // Inspect the characters in <s1> until '\0' is reached.
27        for (; *s1; ++s1)
28            ;
29        // Return the number of characters inspected, not including the '\0'.
30        return (size_t)(s1 - START);
31    }
```

```
 1   //
 2   // Ray Mitchell, U99999999
 3   // MeanOldTeacher@MeanOldTeacher.com
 4   // C/C++ Programming I
 5   // Section 146359, Ray Mitchell
 6   // June 25, 2019
 7   // C1A6E2_main.c
 8   // Windows 10 Professional
 9   // Visual Studio 2019 Professional
10   //
11   // This file contains function main, which compares strings using the strcmp
12   // and MyStrcmp functions.
13   //
14
15   #include <stdio.h>
16   #include <stdlib.h>
17   #include <string.h>
18
19   #define MAX_SIZE 256 // input buffer size
20
21   int MyStrcmp(const char *s1, const char *s2);
22
23   //
24   // Display the results of calling the strcmp and MyStrcmp functions to
25   // determine the relationship between user input strings.
26   //
27   int main(void)
28   {
29       char s1[MAX_SIZE];
30       printf("Enter a string: ");
31       fgets(s1, MAX_SIZE, stdin);
32       s1[strcspn(s1, "\n")] = '\0';    // eliminate trailing newline (if any)
33
34       char s2[MAX_SIZE];
35       printf("Enter another string: ");
36       fgets(s2, MAX_SIZE, stdin);
37       s2[strcspn(s2, "\n")] = '\0';    // eliminate trailing newline (if any)
38
39       // Display messages describing the relationship between the strings.
40       printf("strcmp(\"%s\", \"%s\") returned %d\n", s1, s2, strcmp(s1, s2));
41       printf("MyStrcmp(\"%s\", \"%s\") returned %d\n", s1, s2, MyStrcmp(s1, s2));
42
43       return EXIT_SUCCESS;
44   }
```

```
 1   //
 2   // Ray Mitchell, U99999999
 3   // MeanOldTeacher@MeanOldTeacher.com
 4   // C/C++ Programming I
 5   // Section 146359, Ray Mitchell
 6   // June 25, 2019
 7   // C1A6E2_MyStrcmp.c
 8   // Windows 10 Professional
 9   // Visual Studio 2019 Professional
10   //
11   // This file contains function MyStrcmp, which duplicates the syntax and
12   // functionality of the standard library strcmp function.
13   //
14
15   //
16   // Duplicate the syntax and functionality of the standard library strcmp
17   // function.  The values returned are as indicated below.  For nonzero
18   // values only the sign need match that returned by strcmp:
19   //    <0  if the string represented by pointer <s1> is lexicographically
20   //        less than the string represented by pointer <s2>
21   //    ==0 if the string represented by pointer <s1> is lexicographically
22   //        equal to the string represented by pointer <s2>
23   //    >0  if the string represented by pointer <s1> is lexicographically
24   //        greater than the string represented by pointer <s2>
25   //
26   int MyStrcmp(const char *s1, const char *s2)
27   {
28      // Compare the strings in <s1> and <s2>.
29      for (; *s1 == *s2 && *s1; ++s1, ++s2)
30         ;
31      // Return a value indicating the relationship between the strings.
32      return (int)*s1 - (int)*s2;
33   }
```

```c
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming I
5   // Section 146359, Ray Mitchell
6   // June 25, 2019
7   // C1A6E3_main.c
8   // Windows 10 Professional
9   // Visual Studio 2019 Professional
10  //
11  // This file contains function main, which displays information extracted from
12  // a user-input string by another function.
13  //

15  #include <stdio.h>
16  #include <stdlib.h>
17  #include <string.h>

19  #define MAX_SIZE 256    // size of input buffers

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

23  //
24  // The user is prompted for a string, an index within that string, and a
25  // character count.  These are passed to function GetSubstring which extracts
26  // the specified substring and returns a pointer to it.  main uses this
27  // pointer to display the substring.
28  //
29  int main(void)
30  {
31      char source[MAX_SIZE];
32      printf("Enter a string: ");
33      fgets(source, MAX_SIZE, stdin);
34      source[strcspn(source, "\n")] = '\0';    // eliminate trailing newline

36      char result[MAX_SIZE];
37      int start, count;
38      printf("Enter a space-separated starting index and character count: ");
39      scanf("%d %d", &start, &count);
40      printf("\"%s\", %d, %d, extracts \"%s\"\n", source, start, count,
41          GetSubstring(source, start, count, result));

43      return EXIT_SUCCESS;
44  }
```

```c
 1  //
 2  // Ray Mitchell, U99999999
 3  // MeanOldTeacher@MeanOldTeacher.com
 4  // C/C++ Programming I
 5  // Section 146359, Ray Mitchell
 6  // June 25, 2019
 7  // C1A6E3_GetSubstring.c
 8  // Windows 10 Professional
 9  // Visual Studio 2019 Professional
10  //
11  // This file contains function GetSubstring, which extracts a substring
12  // of specified length from a specified position in another string.
13  //
14
15  //
16  // Function GetSubstring copies a substring from the string in <source> into
17  // the array in <result>, terminating it with a null character.  <start>
18  // specifies the 0-based index in <source> where the substring to be copied
19  // starts and <count> represents the number of characters to copy.  If <start>
20  // represents an index beyond the end of the string in <source> only a null
21  // character is placed in the array in <result>.  If the value of <count> is
22  // such that the substring's length would extend beyond the end of the string
23  // in <source>, copying ends at the end of the string in <source>.  It is
24  // assumed that the array in <result> is large enough to hold the copied
25  // substring.  The original value of parameter <result> is returned.
26  //
27  char *GetSubstring(const char source[], int start, int count, char result[])
28  {
29      char *savedTarget = result;      // save pointer for return
30
31      while (*source && start--)       // find start if it's in source
32          ++source;
33      while (*source && count--)       // copy substring
34          *result++ = *source++;
35      *result = '\0';                  // null terminate substring
36      return savedTarget;              // pointer to substring
37  }
```