Ray Mitchell, U99999999
MeanOldTeacher@MeanOldTeacher.com
C/C++ Programming I
Section 162461, Ray Mitchell
June 25, 2019
C1A6E0_Quiz.txt
Quiz Answers

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

C1A6E0 Explanations

In addition to the course book references cited below, these topics are also covered in the live lectures (in-class students) and the recorded lectures (online students).

1. **A** Note 5.11; *p* is an uninitialized automatic variable and as such represents a garbage memory address. If you dereference a garbage memory address and are extremely lucky it will result in a program crash or some other repeatable runtime error. But if you are extremely unlucky it will appear to work correctly.

2. **B** Notes 1.7, 6.14; The value of any legal expression having the form ++x is the same as the value of the expression *x + 1*. If *x* has an object pointer type, the sum is not necessarily the result of simply adding the two operands arithmetically. Instead, it is obtained by adding the pointer's value to the value of the product of the integral expression's value and the number of bytes in the type pointed to by the pointer. In the example below, where *sizeof(**int**)* has arbitrarily been defined to be 3, the sum is equal to 23:
   20 + (1 * sizeof(int)) == 20 + (1 * 3) == 20 + 3 == 23

3. **D** Notes 6.1, 6.5, 6.16; By the Right-Left Rule identifier *a* is declared as type "array of 9 **int**s". Any expression that has an array type and is not used in one of the special non-decaying situations is automatically converted to a pointer to the first element of that array. In the first argument *a* is not being used in one of those special situations so its data type is converted to a "pointer to an **int**". Variable *p* in the second argument is merely a "pointer to an **int**" and taking the address of a "pointer to an **int**" produces a pointer to it, that is, a "pointer to a pointer to an **int**".

4. **E** Notes 6.14, 6.16; The expressions in four of the answers are in one of the equivalent forms *p[i]*, *i[p]*, **(p + i)*, and **(i + p)*, where *p* represents an expression having an object pointer type and *i* represents an expression having an integer type. In this quiz question *p* is represented by *ax* and the total value of the integer part is 6. (Remember that addition is commutative.)

5. **A** Notes 6.14, 6.16, 7.1, & 7.7; The expression *buf* += "*cou*" + "*nt is:* " contains two string literals. A string literal is merely a character array containing a sequence of constant characters ending with the null character that is automatically placed there by the compiler. As with any array, a string literal will decay to a pointer to its first element if not used in one of the special non-decaying situations, which this is not. Thus, the two string literals each decay to type "pointer to **const char**". Since the code is attempting to add them together but the addition of two pointers is not one of the three legal arithmetic operations that may be performed on pointers, the compiler will generate an error and will not compile the code.

6. **C** Note 7.3; The input line is: this code is slower
   a. The "\n" in the *scanf* control string reads and throws away characters from the remaining input string up to but not including the first character that is not a whitespace. That character is the "*t*" in "*this*".
   b. The literal letter *t* following the "\n" in the *scanf* control string must exactly match the next character from the remaining input. If it does, which it does, that character is thrown away.
   c. The "*%31[^e]*" in the *scanf* control string reads characters from the remaining input string up to but not including the first character that is an "e", up to a maximum of 31 characters. All characters read are stored in array *a* and terminated with the null character, '\0'. Thus, the string "*his cod*" is stored in array *a*.
   d. The "\n" in the *scanf* control string reads and throws away characters from the remaining input string up to but not including the first character that is not a whitespace. That character is "*e*".
   e. The "*%31[a-s]*" in the *scanf* control string reads characters from the remaining input string up to but not including the first character that is not in the range "*a-s*" and is not a

space, up to a maximum of 31 characters. All characters read are stored in array *b* and terminated with the null character, '*\0*'. Thus, the string "*e is slo*" is stored in array *b*.

f. The "*\n*" in the *scanf* control string reads and throws away characters from the remaining input string up to but not including the first character that is not a whitespace. That character is "*w*".

g. The "*%31[^e]*" in the *scanf* control string reads characters from the remaining input string up to but not including the first character that is a "*e*", up to a maximum of 31 characters. All characters read are stored in array c and terminated with the null character, '*\0*'. Thus, the string "*w*" is stored in array c.

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

i. Finally, the *printf* prints the value of variable x, followed by a blank space, followed by the strings stored in arrays *a*, *b*, and c, respectively, with no blank space between those strings. The complete output is: *3 his code is slow*

```c
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 162461, Ray Mitchell
// June 25, 2019
// C1A6E1_main.c
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// This file contains function main, which determines string lengths by calling
// the strlen and MyStrlen functions.
//

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_SIZE 256 // input buffer size

size_t MyStrlen(const char *s1);

//
// Function main displays the results of calling the strlen and MyStrlen
// functions to determine the lengths of user input strings.
//
int main(void)
{
   char s1[MAX_SIZE];

   printf("Enter a string: ");
   fgets(s1, MAX_SIZE, stdin);
   s1[strcspn(s1, "\n")] = '\0';    // eliminate trailing newline
   size_t length = strlen(s1);

   // Display messages describing the relationship between the strings.
   printf("strlen(\"%s\") returned %zu\n", s1, length);
   printf("MyStrlen(\"%s\") returned %zu\n", s1, MyStrlen(s1));

   return EXIT_SUCCESS;
}
```

```
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming I
5   // Section 162461, 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     while (*s1)
28        ++s1;
29     // Return the number of characters inspected, not including the '\0'.
30     return (size_t)(s1 - START);
31  }
```

```
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 162461, Ray Mitchell
// June 25, 2019
// C1A6E2_main.c
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// This file contains function main, which compares strings using the strcmp and
// MyStrcmp functions.
//

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_SIZE 256 // input buffer size

int MyStrcmp(const char *s1, const char *s2);

//
// Display the results of calling the strcmp and MyStrcmp functions to determine
// the relationship between user input strings.
//
int main(void)
{
   char s1[MAX_SIZE];
   printf("Enter a string: ");
   fgets(s1, MAX_SIZE, stdin);
   s1[strcspn(s1, "\n")] = '\0';    // eliminate trailing newline (if any)

   char s2[MAX_SIZE];
   printf("Enter another string: ");
   fgets(s2, MAX_SIZE, stdin);
   s2[strcspn(s2, "\n")] = '\0';    // eliminate trailing newline (if any)

   // Display messages describing the relationship between the strings.
   printf("strcmp(\"%s\", \"%s\") returned %d\n", s1, s2, strcmp(s1, s2));
   printf("MyStrcmp(\"%s\", \"%s\") returned %d\n", s1, s2, MyStrcmp(s1, s2));

   return EXIT_SUCCESS;
}
```

```
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming I
5   // Section 162461, 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 values
18  // only the sign need match that returned by strcmp:
19  //    <0  if the string represented by pointer <s1> is lexicographically less
20  //        than the string represented by pointer <s2>
21  //    ==0 if the string represented by pointer <s1> is lexicographically equal
22  //        to the string represented by pointer <s2>
23  //    >0  if the string represented by pointer <s1> is lexicographically greater
24  //        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 *s1 - *s2;
33  }
```

```
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming I
5   // Section 162461, 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  //
14
15  #include <stdio.h>
16  #include <stdlib.h>
17  #include <string.h>
18
19  #define MAX_SIZE 256    // size of input buffers
20
21  char *GetSubstring(const char source[], int start, int count, char result[]);
22
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 pointer
27  // 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
35
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));
42
43     return EXIT_SUCCESS;
44  }
```

```
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming I
5   // Section 162461, 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 of
12  // specified length from a specified position in another string.
13  //
14
15  //
16  // Function GetSubstring copies a substring from the string in <source> into the
17  // array in <result>, terminating it with a null character.  <start> specifies
18  // the 0-based index in <source> where the substring to be copied starts and
19  // <count> represents the number of characters to copy.  If <start> represents
20  // an index beyond the end of the string in <source> only a null character is
21  // placed in the array in <result>.  If the value of <count> is such that the
22  // substring's length would extend beyond the end of the string in <source>,
23  // copying ends at the end of the string in <source>.  It is assumed that the
24  // array in <result> is large enough to hold the copied substring.  The original
25  // 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 point if it is 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  }
```