

CSC 209 Review 6 Solution

August 24, 2020

1 Exercises

1. I need to write which of the supplied function calls don't work and explain why.

- b) String format in `printf` expects character constant, but string literal is used
- c) String format in `printf` expects string but character constant is used
- e) The first argument in `printf` expects pointer but character constant (an integer) is used instead
- h) The first argument in `putchar` expects a character, but string literal (a pointer to character) is used
- i) The first argument in `puts` expects a pointer to character, but character constant (an integer) is used

Notes

- **putchar**
 - **Syntax:** `int putchar(int char)`
 - Writes a character (an unsigned char) specified by the argument `char` to stdout.
 - Does not append a new line to the output
 - Is similar to `printf` but for character
- **puts**
 - **Syntax:** `int puts(const char *str)`
 - Writes a string to stdout up to but not including the null character
 - Appends a newline character to the output.
 - Is similar to `printf` but for string
- **Character Constant**
 - **Syntax:** `' ... '`

- Is represented by an integer

- **String Literal**

- **Syntax:** " . . . "
- Has a sequence of characters inside
- Ends with `\0`
- Is represented by a pointer

Example

"When you come to a fork in the road, take it"

- **Escape Sequences in String Literal**

- A common example is `'\n'`
 - * causes the cursor to advance to the next line

2. First, I need to write which of the provided function calls are legal, and write the output produced

The solution to the first part is:

- b) [output: a]
- c) [output: abc]

Second, I need to write which of the following function calls are illegal, and explain why.

The solution to the second part is:

- a) `purchar` expects a character constant (an integer) but a value of type pointer to `char` is used
- d) `puts` expects a variable of type pointer to `char`, but a variable of type pointer to `char` is used

3. I need to write the values of `i`, `j`, `k` in the function

```
scanf("%d%s%d", &i, s, &j)
```

if the user enters `12abc34 56def78`.

The solution to this problem is:

- `i` - 12
- `j` - abc34

- k - 56

4. I need to modify the following `read_line` function in the following ways:

```
int read_line(char str[], int n)
{
    int ch, i = 0;

    while ((ch = getchar()) != '\n')
        if (i < n)
            str[i++] = ch;
    str[i] = '\0';
    return i;
}
```

- Have it skip white space before beginning to store input characters
- Have it stop reading at the first white-space character
- Have it stop reading at the first new-line character, then store the new-line character in the string
- Have it leave behind characters that it doesn't have room to store

The solution to this problem is:

```
a) #include <ctype.h>
2  #include <stdbool.h>
3
4  ...
5
6  int read_line(char str[], int n)
7  {
8      int ch, i = 0;
9      bool non_space_char_exists = false;
10
11     while ((ch = getchar()) != '\n')
12         if (isspace(ch) && non_space_char_exists){
13             continue;
14         }
15
16         if (i < n)
17             str[i++] = ch;
18             non_space_char_exists = true;
19     str[i] = '\0';
20     return i;
21 }
```

```
b) #include <ctype.h>
2
3   ...
4
5   int read_line(char str[], int n)
6   {
7       int ch, i = 0;
8
9       while ((ch = getchar()) != '\n')
10          if (isspace(ch)){
11              break;
12          }
13
14          if (i < n)
15              str[i++] = ch;
16      str[i] = '\0';
17      return i;
18  }
```

```
c) #include <ctype.h>
2
3   ...
4
5   int read_line(char str[], int n)
6   {
7       int ch, i = 0;
8
9       while ((ch = getchar()) != '\n')
10          if (ch == '\n'){
11              break;
12          }
13
14          if (i < n)
15              str[i++] = ch;
16
17      str[i] = '\n';
18      str[i+1] = '\0';
19      return i;
20  }
```

```
d) #include <ctype.h>
2
3   ...
4
5   int read_line(char str[], int n)
6   {
7       int ch, i = 0;
8       int n = strlen(str) + 1;
9
10      do {
11          ch = getchar();
12
13          if (!ch) {
14              break;
15          }
16      }
```

```
15         }
16
17         str[i++] = ch;
18
19     } while (i < (n - 1));
20
21     str[i] = '\0';
22     return i;
23 }
```

Correct Solution

- c)

```
1  #include <ctype.h>
2
3  ...
4
5  int read_line(char str[], int n)
6  {
7      int ch, i = 0;
8
9      do {
10         ch = getchar()
11
12         if (ch == '\n'){
13             break;
14         }
15
16         if (i < n)
17             str[i++] = ch;
18
19     } while (ch != '\n');
20
21     str[i] = '\0';
22     return i;
23 }
```

- d)

```
1  #include <ctype.h>
2
3  ...
4
5  int read_line(char str[], int n)
6  {
7      int ch, i = 0;
8      int n = strlen(str) + 1;
9
10     do {
11         ch = getchar();
```

```
12
13         if (ch == '\n') {
14             break;
15         }
16
17         str[i++] = ch;
18
19     } while (i < (n - 1));
20
21     str[i] = '\0';
22     return i;
23 }
```

Notes

- Learned that `getchar()` always ends with `\n`

5. a) I need to write a function named `capitalize` that capitalizes all letters in its argument.

The requirement for this function is:

- Array subscripting must be used to access each character in string

The solution to this problem is:

```
1  #include <ctype.h> // toupper
2
3  void capitalize(char *s)
4  {
5
6      for (int i = 0; s[i] != '\0'; i++) {
7          s[i] = toupper(s[i]);
8      }
9  }
```

Notes

- **Accessing the Characters in a String**
 1. Using array subscripting

Example

```
int count_spaces(const char s[])
{
    int count = 0, i;

    for (i = 0; s[i] != '\0'; i++)
        if (s[i] == ' ')
            count++;
    return count;
}
```

2. Using pointer

Example

```
int count_spaces(const char *s)
{
    int count = 0;

    for (; *s != '\0'; s++)
        if (*s == ' ')
            count++;
    return count;
}
```

- b) I need to write a function named `capitalize` that capitalizes all letters in its argument.

The requirement for this function is:

- pointer must be used to access each character in string

The solution to this problem is:

```
1  #include <ctype.h> // toupper
2
3  void capitalize(char *s)
4  {
5      char *p = s;
6      while (*p != '\0') {
7          *p = toupper(*p);
8          p++;
9      }
10 }
```

6. I need to write a function `sensor` that modifies a string by replacing every occurrence of `foo` with `***`.

The additional requirement of this function are:

- I need to make the function as short as possible without sacrificing clarity.

The solution to this problem is:

```
1 #include <string.h> \\ strlen
2
3 void sensor(char s[]) {
4     char *p;
5
6     if (strlen(s) < 3) {
7         return;
8     }
9
10    for (p = &s[2]; p < s + strlen(s); p++) {
11        if (tolower(*p) == 'o' &&
12            tolower(*(p-1)) == 'o' &&
13            tolower(*(p-2)) == 'f') {
14
15            *p = '*';
16            *(p-1) = '*';
17            *(p-2) = '*';
18        }
19    }
20 }
21 }
```

Correct Solution

```
1 #include <string.h> \\ strlen
2
3 void sensor(char s[]) {
4     if (strlen(s) < 3) {
5         return;
6     }
7
8     for (char *p = &s[2]; *p != '\\0'; p++) {
9         if (tolower(*p) == 'o' &&
10             tolower(*(p-1)) == 'o' &&
11             tolower(*(p-2)) == 'f') {
12
13             *p = *(p-1) = *(p-2) = '*';
14         }
15     }
16 }
17 }
```


18

7. I need to identify from the provided statements that which is not equivalent to others.

The solution to this problem is:

- d) All of the other statements are about making `str` null or empty.

Notes

- `*str = 0` makes pointer NULL
- `strcpy`
 - **Syntax:** `char *strcpy (char *s1, const char *s2)`
 - Copies string `s2` to the string `s1`
- `strcat`
 - **Syntax:** `char *strcat(char *s1, const char *s2)`
 - appends the contents of the string `s2` to the end of the string `s1`

Example

```
strcpy(str1, "abc");  
strcat(str1, "def"); /* str1 now contains "abcdef" */
```

8. I need to write the value of the string `str` after the following execution of statements

```
strcpy(str, "tire-bouchon");  
strcpy(&str[4], "d-or-wi");  
strcat(str, "red?");
```

The solution to this problem is: `tired-or-winred?`

Correct Solution

The solution to this problem is: `tired-or-wired?`

Notes

- `strcpy` always copies upto the first null character.
 - The pointer stops and points at the first null character after `strcpy`

9. I need to write the value of the string `s1` after the executing the provided statements:

The solution to this problem is: `computers`

Correct Solution

The solution to this problem is: `computers\0`

Notes

- `strcmp`
 - **Syntax:** `int strcmp(const char *s1, const char *s2)`
 - * Compares string `s1` and `s2`
 - * Returns
 - 0 - if `s1` and `s2` are identical
 - >0 - if ASCII value of first unmatched character in `s1` is greater than `s2`
 - <0 - if ASCII value of first unmatched character in `s1` is less than `s2`

10. I need to write what's wrong with the provided function.

The solution to this problem is that the pointer `*q` hasn't allocated memory, and because of that, the function call `strcpy(q,p)` would result in segmentation fault.

11. Here I need to modify `strcmp` function to use pointer arithmetic.

The solution to this problem is:

```
1  int strcmp (char *s, char *t) {
2      char *p = s, *q = t;
3
4      while (*p == *q) {
5          if (*p == '\0') {
6              return 0;
7          }
8          p++;
9          q++;
10     }
11
12     return *p - *q;
13 }
```

12. I need to write the following function

```
void get_extension(const char *file_name, char *extension)
```

satisfying the following requirements:

- If the file name doesn't have an extension, empty string should be stored instead
- `get_extension` should be kept as simple as possible by using `strlen` and `strcpy`

The solution to this problem is:

```
1  void get_extension(const char *file_name, char *extension) {
2      int i = strlen(file_name) - 1;
3
4      for (; i >= 0; i--) {
5          if (*(file_name+i) == '.') {
6              break;
7          }
8      }
9      strcpy(extension, file_name + (i+1));
10 }
```

13. I need to write the following function

```
void build_index_url (const char *domain, char *index_url);
```

satisfying the following requirements:

- `build_index_url` should add "http://www." to the beginning of the string
- `build_index_url` should add "/index.html" to the end of the string
- `build_index_url` should store the result in `index_url`

- `build_index_url` should be kept as simple as possible by using `strcat` and `strcpy`

The solution to this problem is:

```

1 void build_index_url (const char *domain, char *index_url) {
2     strcat(index_url, "http://www.");
3     strcat(index_url, domain);
4     strcat(index_url, "/index.html");
5 }

```

14. I need to write the output of the provided function.

The solution to this problem is: **Grinch**

Notes

- `--*p` means decrement `*p` first; the value of expression is `*p` after decrement
- Idioms are good to know. It's used by a lot of C programmers.
- **String Idioms - Searching for the end of string**
 - The following function `strlen` are equal

```

size_t strlen(const char *s)
{
    size_t n;

    for (n = 0; *s != '\0'; s++)
        n++;
    return n;
}

```

=

```

size_t strlen(const char *s)
{
    size_t n = 0;

    while (*s++)
        n++;
    return n;
}

```

It takes advantage of the fact that the end of string is `*s = '\0' = 0`

- **String Idioms - Copying string**
 - The following function `strcat` are equal

```

char *strcat(char *s1, const char *s2)
{
    char *p = s1;

    while (*p != '\0')
        p++;
    while (*s2 != '\0') {
        *p = *s2;
        p++;
        s2++;
    }
    *p = '\0';
    return s1;
}

```

=

```

char *strcat(char *s1, const char *s2)
{
    char *p = s1;

    while (*p)
        p++;
    while (*p++ = *s2++)
        ;
    return s1;
}

```