

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)

```

1  #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)

```

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) {
14             break;
15         }
16
17         str[i++] = ch;

```

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

```
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 solution to this problem is:

Notes

- **Accessing the Characters in a String**

1. Using array subscripting

Example

2. Using pointer

Example