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 constrant is used
   - `e)` The first argument in `printf` expects pointer but character constrant (an integer) is used isntead
   - `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 Squences 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;
}
```

a) Have it skip white space beore beginning to store input characters

b) Have it stop reading at the first white-space character

c) Have it stop reading at the first new-line character, then store the new-line character in the string

d) Have it leave behind characters that it doesn't have room to store

The solution to this problem is:

a)

```
#include <ctype.h>
#include <stdbool.h>

...

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

    while ((ch = getchar()) != '\n')
        if (isspace(ch) && non_space_char_exists){
            continue;
        }

        if (i < n)
            str[i++] = ch;
            non_space_char_exists = true;
    str[i] = '\0';
    return i;
}
```

b)

```c
#include <ctype.h>

...

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

    while ((ch = getchar()) != '\n')
        if (isspace(ch)){
            break;
        }

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

c)

```c
#include <ctype.h>

...

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

    while ((ch = getchar()) != '\n')
        if (ch == '\n'){
            break;
        }

        if (i < n)
            str[i++] = ch;

    str[i] = '\n';
    str[i+1] = '\0';
    return i;
}
```

d)

```c
#include <ctype.h>

...

int read_line(char str[], int n)
{
    int ch, i = 0;
    int n = strlen(str) + 1;

    do {
        ch = getchar();

        if (!ch) {
            break;
```

```
            }

            str[i++] = ch;

        } while (i < (n - 1));

        str[i] = '\0';
        return i;
    }
```

Correct Solution

- c)

```
    #include <ctype.h>

    ...

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

        do {
            ch = getchar()

            if (ch == '\n'){
                break;
            }

            if (i < n)
                str[i++] = ch;

        } while (ch !== '\n');

        str[i] = '\0';
        return i;
    }
```

- d)

```
    #include <ctype.h>

    ...

    int read_line(char str[], int n)
    {
        int ch, i = 0;
        int n = strlen(str) + 1;

        do {
            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 `censor` that modifies a string by replacing every occurence 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:

```c
#include <string.h> \\ strlen

void censor(char s[]) {
    char *p;

    if (strlen(s) < 3) {
        return;
    }

    for (p = &s[2]; p < s + strlen(s); p++) {
        if (tolower(*p) == 'o' &&
            tolower(*(p-1)) == 'o' &&
            tolower(*(p-2)) == 'f') {

            *p = '*';
            *(p-1) = '*';
            *(p-2) = '*';
        }
    }

}
```

Correct Solution

```c
#include <string.h> \\ strlen

void censor(char s[]) {
    if (strlen(s) < 3) {
        return;
    }

    for (char *p = &s[2]; *p != '\0'; p++) {
        if (tolower(*p) == 'o' &&
            tolower(*(p-1)) == 'o' &&
            tolower(*(p-2)) == 'f') {

            *p = *(p-1) = *(p-2) = '*';
        }
    }

}
```

```
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 statments 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:

```
int strcmp (char *s, char *t) {
    char *p = s, *q = t;

    while (*p == *q) {
        if (*p == '\0') {
            return 0;
        }
        p++;
        q++;
    }

    return *p - *q;
}
```

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:

```
void get_extension(const char *file_name, char *extension) {
    int i = strlen(file_name) - 1;

    for (; i >= 0; i--) {
        if (*(file_name+i) == '.') {
            break;
        }
    }
    strcpy(extension, file_name + (i+1));
}
```