CSC 209 Review 6 Solution

August 25, 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

<u>Notes</u>

- **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 <u>integer</u>

- **String Literal**

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

    <u>**Example**</u>

    "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;
```

```
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 `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:

```
#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

```
#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 <u>upto</u> 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

11

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

    for (n = 0; *s != '\0'; s++)              while (*s++)
        n++;              =                       n++;
    return 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 **strlen** are equal

```
char *strcat(char *s1, const char *s2)
{
    char *p = s1;                          char *strcat(char *s1, const char *s2)
                                           {
    while (*p != '\0')                         char *p = s1;
        p++;
    while (*s2 != '\0') {                       while (*p)
        *p = *s2;                                  p++;
        p++;              =                    while (*p++ = *s2++)
        s2++;                                      ;
    }                                          return s1;
    *p = '\0';                             }
    return s1;
}
```

15. a) I need to find the value of `f("abcd","babc")` given the provided function `f`.

    The solution to this problem is: 3

    b) I need to find the value of `f("abcd","bcd")` given the provided function `f`.

    The solution to this problem is: 0

    c) I need to write what value `f` returns when two strings `s` and `t` are passed.

    The answer to this problem is: it returns the number of consequtive characters in `s` that's also in `t`.

16. I need to use the techniques of section 13.6 to condense the function `count_spaces` in section 13.4

    The solution to this problem is:

```
1  int count_spaces(const char s[])
2  {
3      int count = 0;
4
5      do {
6          if (*s == ' ') {
7              count++;
8          }
9      } while (*s++);
10
11     return count;
12 }
```

17. I need to write the following function

    bool test_extension (const char *file_name, const char *extension)

    satisfying the following requiremnts

    • `test_extension` should return `true` if the file's extension matches the string pointed to by `extension`.
    • `test_extension` should ignore the case of letters
    • `test_extension` should use "search for the end of a string" idiom
    • `test_extension` should use `toupper` to make the process case-insensitive

    The solution to this problem is:

```
1    bool test_extension (const char *file_name , const char *extension)
2    {
3        while (*file_name ++ != '.')
4                ;
5
6
7        while (toupper(*file_name ++) == toupper(*extension++)) {
8            if (*file_name == '\0' && *extension == '\0') {
9                return true;
10           }
11       }
12
13       return false;
14   }
```

18. I need to write the function

    void remove_filename(char *url)

    satisfying the following requirements

    - remove_filename shold modify the string in url by removing the file name and the preceding slash

      http://www.knking.com/index.html → http://www.knking.com/
    - remove_filename should use "search for the end of a string" idiom

    The solution to this problem is:

```
1    void remove_filename(char *url)
2    {
3        char *p = url;
4
5        while (*p++)
6                ;
7
8        while (p-- > url) {
9            if (*p == '/') {
10               *p = '\0';
11               return;
12           }
13       }
14   }
```

2   Programming Exercise

1. I need to write a program that finds the "smallest" and "largest" in a series of words.

   The requirement for this program are:

   - The program should ask the user to enter the words
   - The program should determine which words comes first and last as if the words are listed in dictionary order
   - The program must stop accepting input when the user enters a four-letter word
   - Assume program has no word more than 20 letters long
   - The program should use `smallest_word` and `largest_word` to keep trak of the "smallest" and "largest" words so far.
   - The program should use `strcmp` to compare with `smallest_word`
   - The program should also use `strcmp` to comapre with `largest_word`
   - The program should use `strlen` to determine when the user entered a four-letter word

   The solution to this problem is included in file `question_19.c`.

2. a) I need to improve the `remind.c` with the following requirements:

      - The program should print an error message and ignore a reminder if the corresponding day is negative or larger than 31 using `continue` statement

      The solution to this problem is included in file `question_20_a.c`.

   b) I need to improve the `remind.c` with the following requirements:

      - The user should enter a day, a 24-hour time, and a reminder.
      - The program should sort the printed reminder list first by day and then by time

      The solution to this problem is included in file `question_20_b.c`.

      Notes
      - Learned that scanf yields the unread characters

```
    printf("Enter day and reminder: ");
    scanf("%2d", &day);                          enter '31 Exercise at night'
    if (day == 0)
      break;                                      31 is stored in 'day'
    sprintf(day_str, "%2d", day);
    read_line(msg_str, MSG_LEN);

    for (i = 0; i < num_remind; i++)
      if (strcmp(day_str, reminders[i]) < 0)
        break;
    for (j = num_remind; j > i; j--)
      strcpy(reminders[j], reminders[j-1]);

    strcpy(reminders[i], day_str);
    strcat(reminders[i], msg_str);

    num_remind++;
  }

  printf("\nDay Reminder\n");
  for (i = 0; i < num_remind; i++)
    printf(" %s\n", reminders[i]);

  return 0;
}

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

  while ((ch = getchar()) != '\n')               'Exercise at night' is
    if (i < n)                                    read here
      str[i++] = ch;
  str[i] = '\0';
  return i;
}
```