York University
Electrical Engineering and Computer Science

EECS2031: Software Tools
SU2016
Assignment #7

Chapter 13: Exercises

1. The following function calls supposedly write a single new-line character, but some are incorrect. Identify which calls don't work and explain why.

```
(a) printf("%c", '\n');
(b) printf("%c", "\n");
(c) printf("%s", '\n');
(d) printf("%s", "\n");
(e) printf('\n');
(f) printf("\n");
(g) putchar('\n');
(h) putchar("\n");
(i) puts('\n');
(j) puts("\n");
(k) puts("");
```

(a) Correct
(b) Incorrect; use `"%c"` only to write a `char` value
(c) Incorrect; use `"%s"` only to write a string (a value of type `char *`)
(d) Correct
(e) Incorrect; `printf` requires a string, not a character
(f) Correct
(g) Correct
(h) Incorrect; `putchar` requires a character, not a string
(i) Incorrect; `puts` requires a string, not a character
(j) Incorrect; this writes two new-line characters
(k) Correct

2. Suppose that `p` has been declared as follows:

```
char *p = "abc";
```

Which of the following function calls are legal? Show the output produced by each legal call, and explain why the others are illegal.

```
(a) putchar(p);
(b) putchar(*p);
(c) puts(p);
(d) puts(*p);
```

(a) Illegal; `p` is not a character.
(b) Legal; output is `a`.
(c) Legal; output is `abc`.
(d) Illegal; `*p` is not a pointer.

5. (a) Write a function named `capitalize` that capitalizes all letters in its argument. The argument will be a null-terminated string containing arbitrary characters, not just letters. Use array subscripting to access the characters in the string. *Hint:* Use the `toupper` function to convert each character to upper-case.

(b) Rewrite the `capitalize` function, this time using pointer arithmetic to access the characters in the string.

(a)

```
void capitalize(char s[])
{
  int i;

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

(b)

```
void capitalize(char *s)
{
  while (*s != '\0') {
    *s = toupper(*s);
    s++;
  }
}
```

9. What will be the value of the string `s1` after the following statements have been executed?

```
strcpy(s1, "computer");
strcpy(s2, "science");
if (strcmp(s1, s2) < 0)
  strcat(s1, s2);
else
  strcat(s2, s1);
s1[strlen(s1)-6] = '\0';
```

"computers"

Chapter 13: Programming Projects

1. Write a program that finds the "smallest" and "largest" in a series of words. After the user enters the words, the program will determine which words would come first and last if the words were listed in dictionary order. The program must stop accepting input when the user enters a four-letter word. Assume that no word is more than 20 letters long. An interactive session with the program might look like this:

```
Enter word: dog
Enter word: zebra
Enter word: rabbit
Enter word: catfish
Enter word: walrus
```

```
Enter word: cat
Enter word: fish

Smallest word: cat
Largest word: zebra
```

Hint: Use two strings named `smallest_word` and `largest_word` to keep track of the "smallest" and "largest" words entered so far. Each time the user enters a new word, use `strcmp` to compare it with `smallest_word`; if the new word is "smaller," use `strcpy` to save it in `smallest_word`. Do a similar comparison with `largest_word`. Use `strlen` to determine when the user has entered a four-letter word.

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

#define WORD_LEN 20

void read_line(char str[], int n);

int main(void)
{
  char smallest_word[WORD_LEN+1],
       largest_word[WORD_LEN+1],
       current_word[WORD_LEN+1];

  printf("Enter word: ");
  read_line(current_word, WORD_LEN);
  strcpy(smallest_word, strcpy(largest_word, current_word));

  while (strlen(current_word) != 4) {
    printf("Enter word: ");
    read_line(current_word, WORD_LEN);
    if (strcmp(current_word, smallest_word) < 0)
      strcpy(smallest_word, current_word);
    if (strcmp(current_word, largest_word) > 0)
      strcpy(largest_word, current_word);
  }

  printf("\nSmallest word: %s\n", smallest_word);
  printf("Largest word: %s\n", largest_word);

  return 0;
}

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

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

16. Modify Programming Project 1 from Chapter 12 so that it includes the following function:

```
void reverse(char *message);
```

The function reverses the string pointed to by `message`. *Hint:* Use two pointers, one initially pointing to the first character of the string and the other initially pointing to the last character. Have the function reverse these characters and then move the pointers toward each other, repeating the process until the pointers meet.

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

#define MSG_LEN 80   /* maximum length of message */

void reverse(char *message);
int read_line(char str[], int n);

int main(void)
{
  char msg[MSG_LEN+1];

  printf("Enter a message: ");
  read_line(msg, MSG_LEN);

  reverse(msg);
  printf("Reversal is: %s\n", msg);

  return 0;
}

void reverse(char *message)
{
  char temp, *p = message + strlen(message) - 1;

  while (p > message) {
    temp = *message;
    *message++ = *p;
    *p-- = temp;
  }
}

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;
}
```

17. Modify Programming Project 2 from Chapter 12 so that it includes the following function:

```
bool is_palindrome(const char *message);
```

The function returns `true` if the string pointed to by `message` is a palindrome.

```
#include <ctype.h>
#include <stdbool.h>   /* C99 only */
#include <stdio.h>
#include <string.h>

#define MSG_LEN 80

bool is_palindrome(const char *message);
int read_line(char str[], int n);

int main(void)
{
  char msg[MSG_LEN+1];

  printf("Enter a message: ");
  read_line(msg, MSG_LEN);

  if (is_palindrome(msg))
    printf("Palindrome\n");
  else
    printf("Not a palindrome\n");

  return 0;
}

bool is_palindrome(const char *message)
{
  char *p = message + strlen(message) - 1;

  while (p > message) {
    while (message < p && !isalpha(*message)) message++;
    while (p > message && !isalpha(*p)) p--;
    if (toupper(*message++) != toupper(*p--))
      return false;
  }

  return true;
}

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;
}
```