Course: Programming Fundamentals – ENSF 337

Lab #: Lab 3

Instructor: M. Moussavi

Student Name: Derek Braun, 30040032

Lab Section: B01

Date Submitted: Oct. 3, 2018

Exercise C: Problem Solving

```
/*
 *  lab3exe_C.c
 *  ENSF 337 Fall 2018 lab3 Exercise C
 *  Completed By: Derek Braun, 30040032
 *  Section: B01
 *
 *  In this program the implementatiom of function pascal_trangle is missing.
 *  Studtent must complete this function.
 */

#include <stdio.h>
#include <stdlib.h>

void pascal_triangle(int n);
/* REQUIRES: n > 0 and n <= 20
 PROMISES: displays a pascal_triangle. the first 5 line of the function's output
 should have the following format:
 row 0:  1
 row 1:  1   1
 row 2:  1   2   1
 row 3:  1   3   3   1
 row 4:  1   4   6   4   1
 */

int main() {
    int nrow;
    // These are ALL of the variables you need!
    printf("Enter the number of rows (Max 20): ");
    scanf("%d", &nrow);
    if(nrow <= 0 || nrow > 20) {
        printf("Error: the maximum number of rows can be 20.\n");
        exit(1);
    }

    pascal_triangle(nrow);
    return 0;
}

void pascal_triangle(int n) {
    int row = 0;
            int index = 0;
            int print_ind = 0;
    int previous[20] = {1};
    int current[20] = {0};

    while(row < n){
        while(index <= row){
            if(index == row || index == 0){
                current[index] = 1;
            }
            else{
                current[index] = previous[index-1] + previous[index];
            }
            index++;
        }
        index  = 0;
        printf("\nRow %d:\t", row);
        while(current[print_ind]){
            printf("\t%d", current[print_ind]);
            previous[print_ind] = current[print_ind];
            print_ind++;
        }
        print_ind = 0;
        printf("\n");
        row++;
    }
}
```

Output:

```
Enter the number of rows (Max 20): 9
Row 0:          1

Row 1:          1       1

Row 2:          1       2       1

Row 3:          1       3       3       1

Row 4:          1       4       6       4       1

Row 5:          1       5       10      10      5       1

Row 6:          1       6       15      20      15      6       1

Row 7:          1       7       21      35      35      21      7       1

Row 8:          1       8       28      56      70      56      28      8       1
```

Exercise D: Writing functions that work with arrays

```c
/* lab3exe_D.c
 * ENSF 337 Fall 2018, Lab 3 Exercise D
 * Completed By: Derek Braun, 30040032
 * Section: B01
 */

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

int substring(const char *s1, const char *s2);
/* REQUIRES
 * s1 and s2 are valid C-string terminated with '\0';
 * PROMISES
 * returns one if s2 is a substring of s1). Otherwise returns zero.
 */

void select_negatives(const int *source, int n_source,
                      int* negatives_only, int* number_of_negatives);
/* REQUIRES
 *   n_source >= 0.
 *   Elements source[0], source[1], ..., source[n_source - 1] exist.
 *   Elements negatives_only[0], negatives_only[1], ..., negatives_only[n_source - 1] exist.
 * PROMISES
 *   number_of_negatives == number of negative values in source[0], ..., source[n_source - 1].
 *   negatives_only[0], ..., negatives_only[number_of_negatives - 1] contain those negative values, in
 *   the same order as in the source array.                        */

int main(void)
{
    char s[] = "Knock knock! Who's there?";
    int a[] = { -10, 9, -17, 0, -15 };
    int size_a;
    int i;
    int negative[5];
    int n_negative;

    size_a = sizeof(a) / sizeof(a[0]);

    printf("a has %d elements:", size_a);
    for (i = 0; i < size_a; i++)
        printf("  %d", a[i]);
    printf("\n");
    select_negatives(a, size_a, negative, &n_negative);
    printf("\nnegative elements from array a are as follows:");
    for (i = 0; i < n_negative; i++)
        printf("  %d", negative[i]);
    printf("\n");

    printf("\nNow testing substring function....\n");
```

```
   printf("Answer must be 1. substring function returned: %d\n" , substring(s, "Who"));
   printf("Answer must be 0. substring function returned: %d\n" , substring(s, "knowk"));
   printf("Answer must be 1. substring function returned: %d\n" , substring(s, "knock"));
   printf("Answer must be 0. substring function returned: %d\n" , substring(s, ""));
   printf("Answer must be 1. substring function returned: %d\n" , substring(s, "ck! Who's"));
   printf("Answer must be 0. substring function returned: %d\n" , substring(s, "ck!Who's"));
   return 0;
}

int substring(const char *s1, const char* s2)
{
   int index = 0;
        int condition = 1;

        while(s1[index]){
                if(s1[index] == s2[0]){
                        for(int i = 0; s2[i]; i++){
                                if(s2[i] != s1[index + i]){
                                        condition = 0;
                                }
                        }
                        if(condition == 1){
                                return 1;
                        }
                        condition = 1;
                }
                index++;
        }

   return 0;
}

void select_negatives(const int *source, int n_source,
               int* negatives_only, int* number_of_negatives)
{
   int i = 0;
        int neg_ind = 0;
        while(i < n_source){
                if(source[i] < 0){
                        negatives_only[neg_ind] = source[i];
                        neg_ind++;
                }
                i++;
        }
   *number_of_negatives = neg_ind;

   return;
}
```

Output:

```
a has 5 elements:  -10  9  -17  0  -15

negative elements from array a are as follows:  -10  -17  -15

Now testing substring function....
Answer must be 1. substring function returned: 1
Answer must be 0. substring function returned: 0
Answer must be 1. substring function returned: 1
Answer must be 0. substring function returned: 0
Answer must be 1. substring function returned: 1
Answer must be 0. substring function returned: 0
```

Exercise E: More Practice with Strings

```
/* File: palindrome.c
 *  ENSF 337 - Fall 2018
 *  Exercise E - Lab 3
 *  Completed By: Derek Braun, 30040032
 *  Section: B01
 *  Abstract: The program receives a string (one or more words) and indicates
 *  if the string is a palindrome or not. Plaindrome is a phrase that spells the
 *  same from both ends
 */

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define SIZE 100


/* function prototypes*/
int is_palindrome (const char *str);
/* REQUIRES: str is pointer to a valid C string.
 * PROMISES: the return value is 1 if the string a is palindrome.*/


void strip_out(char *str);
/* REQUIRES: str points to a valid C string terminated with '\0'.
 * PROMISES: strips out any non-alphanumerical characters in str*/

int main(void)
{
    int p =0;
    char str[SIZE], temp[SIZE];

    fgets(str, SIZE, stdin);

    /* Remove end-of-line character if there is one in str.*/
    if (str[strlen(str) - 1] == '\n')
        str[strlen(str) - 1] = '\0';

    strcpy(temp,str);

    /* This loop is infinite if the string "done" never appears in the
     * input.  That's a bit dangerous, but OK in a test harness where
     * the programmer is controlling the input. */

    while(strcmp(str, "done") !=0) /* Keep looping unless str matches "done". */
    {

#if 1
        strip_out(str);

        p = is_palindrome(str);
#endif

        if(!p)
            printf("\n \"%s\" is not a palindrome.\n", temp);
        else
            printf("\n \"%s\" is a palindrome.\n", temp);

        fgets(str, SIZE, stdin);

        /* Remove end-of-line character if there is one in str.*/
        if(str[strlen(str) - 1] == '\n')
            str[strlen(str) - 1]= '\0';
        strcpy(temp, str);
    }

    return 0;
}

void strip_out(char *str){
            int index = 0;
            while(index < strlen(str)){
                        while(!isalnum(str[index]) && index < strlen(str)){
                                    for(int i = 0; i < strlen(str)-index; i++){
                                                str[index+i] = str[index+i+1];
                                    }
                        }
                        index++;
    }
}

int is_palindrome(const char *str){
    int index = 0;
    int condition = 1;
    while(index < strlen(str)){
        if(tolower(str[index]) != tolower(str[strlen(str)-1-index])){
            condition = 0;
        }
        index++;
    }
    return condition;
}
```

Output:

```
Draw nine men $$ inward.
 "Draw nine men $$ inward." is a palindrome.

Eva, Can I stab *Bats in a Cave?
 "Eva, Can I Stab *Bats In A Cave?" is a palindrome.

A Santa Lived As a Devil At NASA.
 "A Santa Lived As a Devil At NASA." is a palindrome.

Palindrome &&?
 "Palindrome &&?" is not a palindrome.

done
```