Programming Fundamental – ENSF 337
Lab 6
M. Moussavi
Jay Chuang
B01
October 29, 2019

EXERCISE D

```c
//  lab6exe_D.c
//  Jay Chuang
//  Lab6 Exercise D - Fall 2019

/* The purpose of this exercise is to practice dynamic allocation of c-strings
 * on the memory. Also showing you a tiny step towards concept of data
 * abstraction that is a bigger topic taught later in this course (in C++).
 *
 * Note: Users of the struct String must be aware of it restrictions: Functions
 * in this program require the instances of String contain a valid c-string.
 * Notice that an array of characters with one element that contains a '\0' is
 * considered as an empty (but valid) c-string.
 */

#include "lab6exe_D.h"

void test_copying(void);
void test_appending(void);
void test_truncating(void);

int main(void) {

#if 0
    test_copying();
#endif

#if 1
    test_truncating();
#endif

#if 1
    test_appending();
#endif

    return 0;
}

void create_empty_string (String *str) {

    if(str -> dynamic_storage != NULL)
        free(str -> dynamic_storage);

    str -> dynamic_storage = malloc (sizeof(char) * 1);
```

```
    if(str ->dynamic_storage == NULL) {
        printf("malloc failed ...\n");
        exit(1);
    }

    str -> dynamic_storage[0] = '\0';
    str -> length = 0;
}

void String_cpy(String *dest, const char* source) {

    if(dest -> dynamic_storage != NULL){
        free(dest->dynamic_storage);
        dest ->dynamic_storage = NULL;
    }

    if(source != NULL || source [0] != '\0' ) {
        // allocate storate space equal to length of source plus one for '\0'
        dest -> dynamic_storage = malloc(strlen(source)+1);
        if(dest -> dynamic_storage == NULL){
            printf("malloc failed: Memory was unavailable...\n");
            exit(1);
        }

        strcpy(dest -> dynamic_storage , source);
        dest -> length = (int)strlen(source);
    }
}

void String_copy(String *dest, const String* source) {
    if(dest -> dynamic_storage != NULL){
        free(dest->dynamic_storage);
        dest->dynamic_storage = NULL;
    }

    if(source ->dynamic_storage != NULL) {
        // allocate storate space equal to length of source plus one for '\0'
        dest -> dynamic_storage = malloc(strlen(source->dynamic_storage)+1);
        if(dest -> dynamic_storage == NULL){
            printf("malloc failed: Memory was unavailable...\n");
            exit(1);
        }

        strcpy(dest -> dynamic_storage , source ->dynamic_storage);
        dest -> length = source -> length;
```

```
    }
}

void display_String(const String* s){
    if(s -> length > 0)
        printf("%s      %zu\n", s->dynamic_storage, s -> length);
    else
        printf("%s      %zu\n", "String is empty", s -> length);
}


void String_append(String *dest, const String* source){
    int lengthD = strlen(dest -> dynamic_storage);
    int lengthS = strlen(source -> dynamic_storage);

    char temp[lengthD+1];
    for(int i = 0; i < lengthD; i++)
        temp[i] = dest -> dynamic_storage[i];
    temp[lengthD] = '\0';

    free(dest -> dynamic_storage);
    dest -> dynamic_storage = NULL;

    dest -> dynamic_storage = calloc(lengthD + lengthS + 1, sizeof(char));
    if(dest -> dynamic_storage == NULL){
        printf("malloc failed: Memory was unavailable...\n");
        exit(1);
    }

    for(int i = 0; i < lengthD; i++)
        dest -> dynamic_storage[i] = temp[i];

    if(source -> dynamic_storage[0] != '\0' || source -> dynamic_storage != NULL)
    {
        for(int i = 0; i < (lengthS); i++)
            dest -> dynamic_storage[lengthD+i] = source -> dynamic_storage[i];
    }
    dest -> dynamic_storage[lengthD + lengthS] = '\0';
    dest -> length = lengthD + lengthS;
}

void String_truncate(String *dest, int new_length){
    if(new_length <= dest->length){
        char* temp;
        temp = dest -> dynamic_storage;
```

```
        free(dest -> dynamic_storage);
        dest -> dynamic_storage = NULL;

        dest -> dynamic_storage = malloc(new_length+1);
        if(dest -> dynamic_storage == NULL){
            printf("malloc failed: Memory was unavailable...\n");
            exit(1);
        }

        for(int i = 0; i < new_length; i++)
            dest -> dynamic_storage[i] = temp[i];

        dest -> dynamic_storage[new_length] = '\0';
        dest -> length = new_length;
    }
}

void test_copying(void){
    printf("\nTesting String_cpy and String_copy started: \n");

    String st1 = {NULL, 0};
    String st2 = {NULL, 0};
    String st3 = {NULL, 0};
    String st4 = {NULL, 0};

    // The following four lines creates instances of STring with valid
    // c-strings of length zero. Means it allocates one element for the
    //dynamic_storage and initializes that element with '\0'.
    create_empty_string(&st1);
    create_empty_string(&st2);
    create_empty_string(&st3);
    create_empty_string(&st4);

    display_String(&st1);     // displays: String is empty      0
    display_String(&st2);     // displays: String is empty      0
    display_String(&st3);     // displays: String is empty      0
    display_String(&st4);     // displays: String is empty      0

    //copies "William Shakespeare" int the string_stirage in object st1
    String_cpy(&st1, "William Shakespeare");

    // Must display: William Shakespeare      19
    display_String(&st1);
```

```
    String_cpy(&st2, "Aaron was Here!!!!");

    // Must display: Aaron was Here!!!!      18
    display_String(&st2);

    String_cpy(&st3, "But now he is in Italy");

    // Must display: But now he is in Italy      22
    display_String(&st3);

    //copies the  c-string in st4 into the string_storage in object st1
    String_copy(&st1, &st4);

    // Must display: String is empty      0
    display_String(&st1);

    String_cpy(&st2, "");
    // Must display: String is empty      0
    display_String(&st2);

    String_copy(&st2, &st3);
    // Must display: But now he is in Italy      22
    display_String(&st2);

    create_empty_string(&st2);

    // Must display: String is empty      0
    display_String(&st1);

    printf("\nTesting String_cpy and String_copy finished...\n");
    printf("-------------------------------------------------------------\n");
}

void test_appending(void) {
    printf("\nTesting String_append started: \n");

    String st1 = {NULL, 0};
    String st2 = {NULL, 0};
    String st3 = {NULL, 0};
    String st4 = {NULL, 0};

    create_empty_string(&st1); // creates an empty object with a valid c-string
    create_empty_string(&st2); // creates an empty object with a valid c-string
    create_empty_string(&st3); // creates an empty object with a valid c-string
    create_empty_string(&st4); // creates an empty object with a valid c-string
```

```
    String_cpy(&st1, "Aaron was Here. ");

    // Must display: Aaron was Here.       16
    display_String(&st1);

    String_cpy(&st2, "He left a few minutes ago.");

    // Must display: He left a few minutes ago.      26
    display_String(&st2);

    String_append(&st4, &st3);

    // Must display: String is empty      0
    display_String(&st4);

    String_append(&st1, &st2);

    // Must display: Aaron was Here. He left a few minutes ago.      42
    display_String(&st1);

    create_empty_string(&st1);

    // Must display: String is empty      0
    display_String(&st1);

    String_cpy(&st1, "GET THE BALL ROLLING");

    // Must display: GET THE BALL ROLLING      20
    display_String(&st1);

    String_cpy(&st2, "!");
    String_append(&st1, &st2);

    // Must displays: GET THE BALL ROLLING!      21
    display_String(&st1);

    String_append(&st1, &st4);

    // Must display: GET THE BALL ROLLING!      21
    display_String(&st1);

    printf("\nTesting String_append finished...\n");
    printf("-------------------------------------------------------------\n");
}
```

```c
void test_truncating (void) {
    printf("\nTesting String_truncate started: \n");

    String st1 = {NULL, 0};
    String_cpy(&st1, "Computer Engineering.");

    // Must display: Computer Engineering.      21
    display_String(&st1);

    String_truncate(&st1, 8);

    // Must display: Computer      8
    display_String(&st1);

    String_truncate(&st1, 3);

    // Must displays: Com      3
    display_String(&st1);

    String_truncate(&st1, 7);

    // Must display: Com      3
    display_String(&st1);

    String_truncate(&st1, 1);

    // Must display: C      1
    display_String(&st1);

    String_truncate(&st1, 0);

    // Must display: String is empty      0
    display_String(&st1);

    String_cpy(&st1, "Truncate done Successfully.");

    // Must display: Truncate done Successfully.      27
    display_String(&st1);

    printf("\nTesting String_truncate finished... \n");
    printf("-------------------------------------------------------------\n");
}
```

EXERCISE D OUTPUT

```
jaych@DESKTOP-DILG265 /cygdrive/c/ensf337/lab6
$ ./a.exe

Testing String_truncate started:
Computer Engineering.      21
Computer      8
Com      3
Com      3
C      1
String is empty      0
Truncate done Successfully.      27

Testing String_truncate finished...
------------------------------------------------------------

Testing String_append started:
Aaron was Here.       16
He left a few minutes ago.      26
String is empty      0
Aaron was Here. He left a few minutes ago.      42
String is empty      0
GET THE BALL ROLLING      20
GET THE BALL ROLLING!      21
GET THE BALL ROLLING!      21

Testing String_append finished...
------------------------------------------------------------
```

EXERCISE E

```c
/*
 *  File: lab6_exe_A.c
 *  ENSF 337 Fall 2019 - lab 6, Exercise E
 */
#include <stdio.h>
#include <stdlib.h>

// This is a simple C program that is supposed to create an array of type double,
// (dyanamically on the heap), filling it with some arbitrary numbers and then
// using the array as needed. But the program doesn't do any thing useful becaues
// some flaws in the function main function and improper design of the function
// create_array.

void create_array(double **x, unsigned long n);
void populate_array(double *array, int n);

int main(void) {
    printf("\nProgram started...\n");
```

```c
    double *array = NULL;
    int n = 20;
    create_array(&array, n);

    if( array != NULL) {
        populate_array(array, n);

        // displays half of the values of the array
        for(int i = 0; i < n/2; i++){
            printf("%f\n", *array++);
        }

        // According to C standard, the program's behaviour, after the following
        // call to the function free is considered "Undefined" and needs to be fi
xed.
        free(array);
    }

    printf("Program terminated...\n");
    return 0;
}

// THE FOLLOWING FUNCTION IS NOT PROPERLY DESINGED AND NEEDS TO BE FIXED
void create_array(double **x, unsigned long n) {
    *x = malloc(n *sizeof(double));

    if(x == NULL){
        printf("Sorry Memory Not Available. Program Terminated.\n");
        exit(1);
    }
}

void populate_array(double *array, int n) {
    int i;
    for(i = 0; i < n; i++)
        array[i] = (i + 1) * 100;
}
```

EXERCISE E OUTPUT

```
jaych@DESKTOP-DILG265 /cygdrive/c/ensf337/lab6
$ ./a.exe

Program started...
100.000000
200.000000
300.000000
400.000000
500.000000
600.000000
700.000000
800.000000
900.000000
1000.000000
Program terminated...
```