Course: Programming Fundamentals – **ENCM 339**
Lab #: Lab 6
Instructor: S. Norman
Student Name: **Mitchell Sawatzky**
Lab Section: **B02**
Date Submitted: **Oct 27, 2015**

Exercise C

Lab6ExC.c

```
// File: Lab6-F15-exC.c
// Lab 6 - Exercise C - Fall 2015

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

typedef struct IntVector{

    // an integer pointer that is supposed to point to a dynamically allocated
    // array. To be safely used, should be always pointing to a dynamically
    // allocated space or to be set to NULL.
    int* vector_storage;

    // represents the size of the array that vector_storage points to. Will be
    // zero if vector_storage is a NULL-pointer
    int size;

} IntVector;

void display_array(const IntVector *arr);
/*
 * PROMISES:
 * if array is not empty (vector_storage != NULL), displays the integer values
 * in vector_storage[0] to vector_storage[size-1], one value per line.
 */

IntVector* read_binary_file (void);
/*
 * PROMISES:
 * - opens a binary file stream. Terminates the program if unable to access
 *   and open the stream,
 * - dynamically creates an object of IntVector,
 * - reads a number as stream of bytes (sizeof(int)) from binary file
 *   (one at a time), expands the size of the storage on the heap, as needed and
 *   places that number into the newly allocated element of the array,
 * - at the end it closes the file.
 * PLEASE NOTICE: the memory management policy used in this small exercise
 * is not an efficient policy. A better policy could be to reallocate the
 * memory in a bigger block (say n elements) and when the n elements are used,
 * then call the function realloc to expand the size of the array with again
 * of the same size block.
 */

void save_table(const IntVector *arr,  int k);

/* REQUIRES:
 *  arr pointing to an object of IntVector.
 *  0 < k <= arr->size.
 *
 * PROMISES:
 *  The existing integer numbers in an array that arr->vector_stroage points to,
 *  should be saved into a text file called table.txt, in a tabular format with
 *  k columns.
 *  Each number must be saved in a field of 5. For example if the value in one of
 *  the elements of arr->vector->storage is 45, it will be printed as: ***45
 *  (assume each asterisk represents a blank space).
 */

int main(void) {

    IntVector *intArr;
    intArr = read_binary_file();
```

```
    if(intArr != NULL)
        display_array(intArr);
    else
        return 0;

    int col = 11;
    save_table(intArr, col);

    free(intArr->vector_storage);
    free (intArr);

    return 0;
}

void display_array(const IntVector* array) {
    int i;
    for (i =0; i < array ->size; i++ )
        printf("%10d\n", array ->vector_storage[i]);
}

IntVector* read_binary_file(void) {
    char filename[100];
    IntVector *arr = malloc(sizeof (IntVector));
    if(arr == NULL){
        printf("malloc faild: Memory was unavailable...\n");
        exit(1);
    }
    FILE *fp;
    printf("\nEnter the name of binary file to open and to read from: ");
    fgets(filename, 100, stdin);

    if(filename[strlen(filename)-1] == '\n')
      filename[strlen(filename)-1] = '\0';


    if((fp = fopen (filename, "rb")) == NULL){
        fprintf(stdout, "Sorry cannot open the binary file %s.\n", filename);
        exit(1);
    }

    arr -> size = 0;
    int num;
    unsigned long v = fread(&num, sizeof(int), 1, fp);
    if(v == 1){
        arr->vector_storage = malloc(sizeof(int));
        if(arr->vector_storage == NULL){
            printf("malloc failed: Memory was unavailable...\n");
            exit(1);
        }
    }
    else return NULL;

    while(1) {
        arr -> vector_storage[arr -> size] = num;
        (arr -> size)++;


        unsigned long v = fread(&num, sizeof(int), 1, fp);
        if(v < 1) break;
        // reallocate memory and check if it was a successful operation
        if((arr->vector_storage = realloc(arr -> vector_storage, sizeof(int) * (arr -> size
+ 1))) == NULL){
            printf("realloc faild: Memory was unavailable...\n");
            exit(1);
        }

    }

    fclose(fp);
```

```
    return arr;
}

void save_table(const IntVector *arr, int k){
    /* REQUIRES:
     *  arr pointing to an object of IntVector.
     *  0 < k <= arr->size.
     *
     * PROMISES:
     *  The existing integer numbers in an array that arr->vector_stroage points to,
     *  should be saved into a text file called table.txt, in a tabular format with
     *  k columns.
     *  Each number must be saved in a field of 5. For example if the value in one of
     *  the elements of arr->vector->storage is 45, it will be printed as: ***45
     *  (assume each asterisk represents a blank space).
     */

     char filename[10] = "table.txt";
     int i, j;
     FILE* fp = fopen(filename, "w");
     if (fp == NULL) {
         printf("Sorry, could not save text file table.txt");
         exit(1);
     }
     for (i = 0, j = 0; i < arr->size; i++, j++) {
         if (j == k) {
             fwrite("\n", 1, 1, fp);
             j = 0;
         }
         size_t n = fprintf(fp, "%5d", arr->vector_storage[i]);
         if (n != 5) {
             printf("Sorry, could not save text file table.txt");
             exit(1);
         }
     }
     fclose(fp);
}
```

table.txt

```
    8   50   74   59   31   73   45   79   24   10   41
   66   93   43   88    4   28   30   41   13    4   70
   10   58   61   34  100   79   17   36   98   27   13
   68   11   34   80   50   80   22   68   73   94   37
   86   46   29   92   95   58    2   54    9   45   69
   91   25   97   31    4   23   67   50   25    2   54
   78    9   29   34   99   82   36   14   66   15   64
   37   26   70   16   95   30    2   18   96    6    5
   52   99   89   24    6   83   53   67   17   38   39
   45
```

Exercise D

Lab6ExD.c

```
// lab6-ExD.c
//  Lab6 Exercise D - Fall 2015

/* 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 "Lab6-ExD.h"
```

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

int main(void) {

#if 0
    test_copying();
#endif

#if 0
    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){
     char* old_storage = dest->dynamic_storage;
     dest->dynamic_storage = malloc((dest->length + source->length + 1) * sizeof(char));
     int i,j;
     for (i = 0; i < dest->length; i++)
        dest->dynamic_storage[i] = old_storage[i];
     for (j = 0; j < source->length; j++)
        dest->dynamic_storage[i+j] = source->dynamic_storage[j];
     dest->dynamic_storage[i+j] = '\0';
     dest->length += source->length;
     free(old_storage);
}

void String_truncate(String *dest, int new_length){
    if (new_length >= dest->length)
        return;
    char* old_storage = dest->dynamic_storage;
    dest->dynamic_storage = malloc((new_length + 1) * sizeof(char));
    int i;
    for (i = 0; i < new_length; i++) {
        dest->dynamic_storage[i] = old_storage[i];
    }
    dest->dynamic_storage[i] = '\0';
    dest->length = new_length;
    free(old_storage);
}


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");
}

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");
}
```

Terminal Output:

```
Mitchell@ttys000 13:06 {127} [lab6]$ ./test.out
```

```
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...
------------------------------------------------------------
```