**Course: ENSF 614–Fall2021**

**Lab #: Lab 2**

**Student Names: Graydon Hall, Jared Kraus**

**Submission Date: 2021-09-27**

# Exercise A

Table

Description automatically generated with low confidenceDiagram

Description automatically generatedDiagram

Description automatically generatedDiagram

Description automatically generated

# Exercise B

|  |
| --- |
| */\* File Name: my\_lab2exe\_B.c*  *\* Lab # and Assignment #: Lab #2 Exercise B*  *\* Lab section: 1*  *\* Completed by: Graydon Hall and Jared Kraus*  *\* Submission Date: 2021-09-27*  *\*/*  int my\_strlen(const char \*s);  */\*  Duplicates my\_strlen from <string.h>, except return type is int.*  *\*  REQUIRES*  *\*     s points to the beginning of a string.*  *\*  PROMISES*  *\*     Returns the number of chars in the string, not including the*  *\*     terminating null.*  *\*/*  void my\_strncat(char \*dest, const char \*source, int n);  */\*  Duplicates my\_strncat from <string.h>, except return type is void.*  *\*  REQUIRES*  *\*     source and dest point to the beginning of the strings.*  *\*  PROMISES*  *\*     Appends sttring pointed to by source, to the string pointed to by dest*  *\*     up to n characters from source.*  *\*/*  int my\_strcmp(const char\* str1, const char\* str2);  */\*  Duplicates my\_strcmp from <string.h>, except return type is void.*  *\*  REQUIRES*  *\*     str1 and str2 point to the beginning of the strings.*  *\*  PROMISES*  *\*     to return 0 when str1 and str2 are identical. Otherwise, returns the ASCII*  *\*     value differences of the first two characters that are different.*  *\**  *\*/*  #include <stdio.h>  #include <string.h>  int main(void)  {      char str1[7] = "banana";      const char str2[] = "-tacit";      const char\* str3 = "-toe";    */\* point 1 \*/*      char str5[] = "ticket";      char my\_string[100]="";      int bytes;      int length;    */\* using my\_strlen C libarary function \*/*      length = (int) my\_strlen(my\_string);      printf("\nLine 1: my\_string length is %d.", length);    */\* using sizeof operator \*/*      bytes = sizeof (my\_string);      printf("\nLine 2: my\_string size is %d bytes.", bytes);    */\* using strcpy C libarary function \*/*      strcpy(my\_string, str1);      printf("\nLine 3: my\_string contains: %s", my\_string);        length = (int) my\_strlen(my\_string);      printf("\nLine 4: my\_string length is %d.", length);        my\_string[0] = '\0';      printf("\nLine 5: my\_string contains:\"%s\"", my\_string);        length = (int) my\_strlen(my\_string);      printf("\nLine 6: my\_string length is %d.", length);        bytes = sizeof (my\_string);      printf("\nLine 7: my\_string size is still %d bytes.", bytes);    */\* my\_strncat append the first 3 characters of str5 to the end of my\_string \*/*      my\_strncat(my\_string, str5, 3);      printf("\nLine 8: my\_string contains:\"%s\"", my\_string);        length = (int) my\_strlen(my\_string);      printf("\nLine 9: my\_string length is %d.", length);        my\_strncat(my\_string, str2,  4);      printf("\nLine 10: my\_string contains:\"%s\"", my\_string);    */\* my\_strncat append ONLY up ot '\0' character from str3 -- not 6 characters \*/*      my\_strncat(my\_string, str3, 6);      printf("\nLine 11: my\_string contains:\"%s\"", my\_string);        length = (int) my\_strlen(my\_string);      printf("\nLine 12; my\_string has %d characters.", length);        printf("\n\nUsing my\_strcmp - C library function: ");        printf("\n\"ABCD\" is less than \"ABCDE\" ... my\_strcmp returns: %d",             my\_strcmp("ABCD", "ABCDE"));      printf("\n\"ABCD\" is less than \"ABND\" ... my\_strcmp returns: %d",             my\_strcmp("ABCD", "ABND"));      printf("\n\"ABCD\" is equal than \"ABCD\" ... my\_strcmp returns: %d",             my\_strcmp("ABCD", "ABCD"));      printf("\n\"ABCD\" is less than \"ABCd\" ... my\_strcmp returns: %d",             my\_strcmp("ABCD", "ABCd"));      printf("\n\"Orange\" is greater than \"Apple\" ... my\_strcmp returns: %d\n",             my\_strcmp("Orange", "Apple"));        return 0;  }  int my\_strcmp(const char\* str1, const char\* str2){         int i = 0;         while (1) {       */\* Stop looping when we reach the null-character. \*/*                if((str1[i] == '\0') && (str2[i] == '\0')){  *// got to end of both strings and they were always equal*                       return 0;                }                if(str1[i] != str2[i]){                       return (int) (str1[i] - str2[i]);                }                i++;         }  }  int my\_strlen(const char \*s){         int i = 0;         while (s[i] != '\0') {       */\* Stop looping when we reach the null-character. \*/*                i++;         }         return i;  }  void my\_strncat(char \*dest, const char \*source, int n){         int i = 0;         int j = 0;         while (dest[i] != '\0') {       */\* Stop looping when we reach the null-character. \*/*                i++;         }         while ((source[j] != '\0' && (j<n))) {       */\* Stop looping when we reach the null-character. \*/*                dest[i] = source[j];                i++;                j++;         }         dest[i] = '\0';  } |

**Program Output:**

**Text

Description automatically generated**

# Exercise C

Diagram

Description automatically generated

# Exercise E:

|  |
| --- |
| */\* File Name: my\_lab2exe\_E.c*  *\* Lab # and Assignment #: Lab #2 Exercise E*  *\* Lab section: 1*  *\* Completed by: Graydon Hall and Jared Kraus*  *\* Submission Date: 2021-09-27*  *\*/*  #include "lab2exe\_E.h"  struct cplx cplx\_add(struct cplx z1, struct cplx z2)  {    struct cplx result;    result.real = z1.real + z2.real;    result.imag = z1.imag + z2.imag;    return result;  }  void cplx\_subtract(struct cplx z1, struct cplx z2,  struct cplx \*difference){    (\*difference).real = z1.real - z2.real;    (\*difference).imag = z1.imag - z2.imag;    }  void  cplx\_multiply(const struct cplx \*pz1, const struct cplx \*pz2, struct cplx \*product){    (\*product).real = (\*pz1).real\*(\*pz2).real - (\*pz1).imag\*(\*pz2).imag;    (\*product).imag = (\*pz1).real\*(\*pz2).imag + (\*pz1).imag\*(\*pz2).real;  } |