Course: ENSF 619- Fall 2020

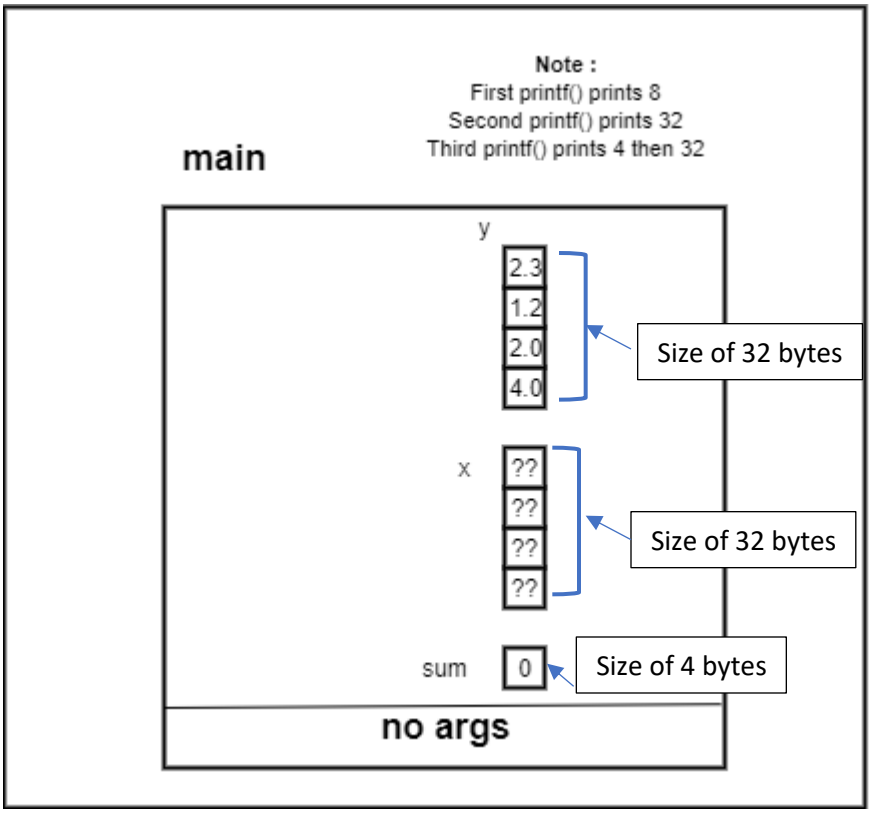
Lab: 2

Student Name: Ziad Chemali

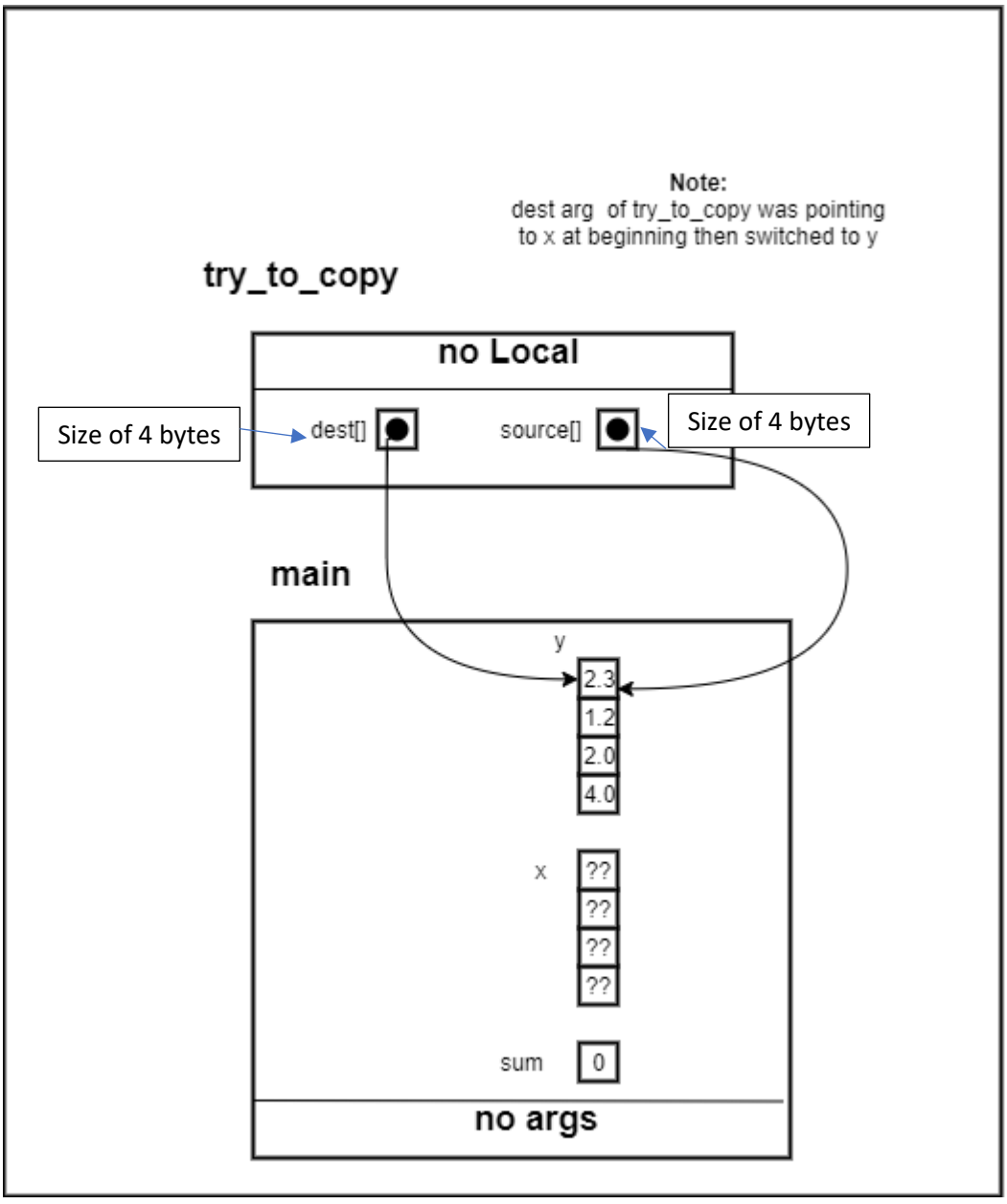
Submission Date: October 2, 2020

1) Exercise: A

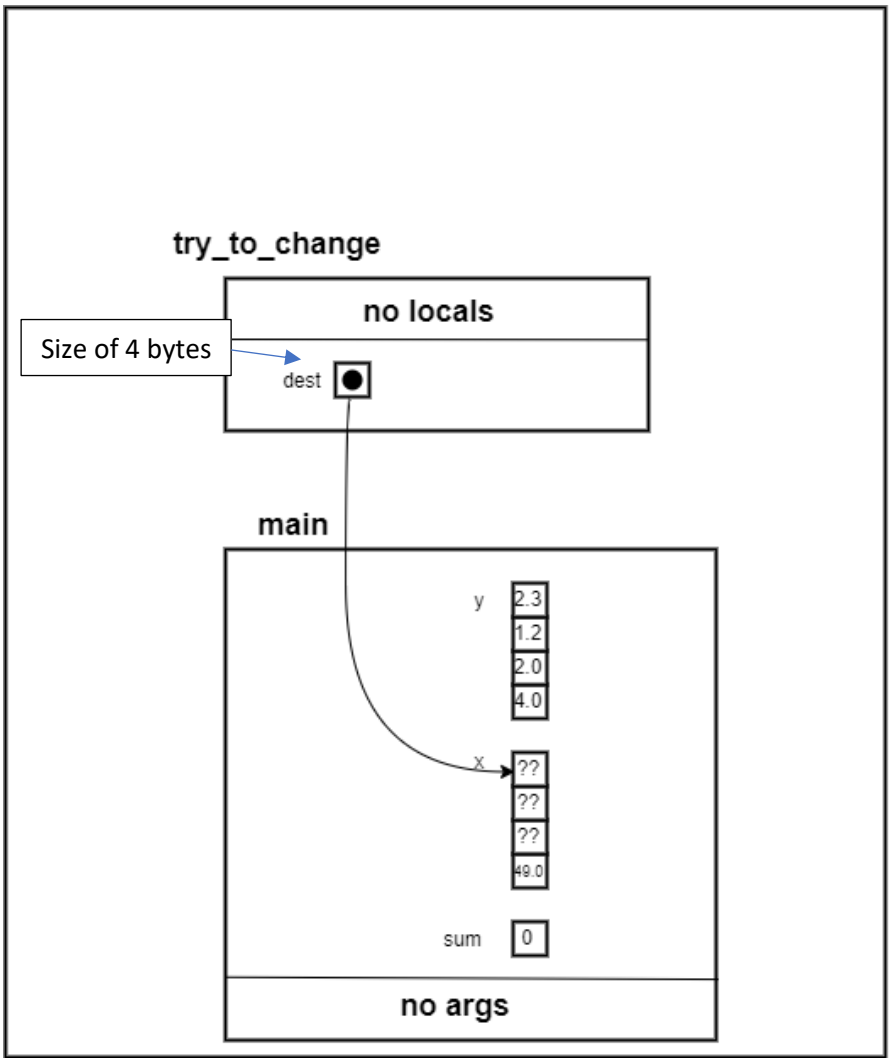
a) Point-1



b)Point-2:

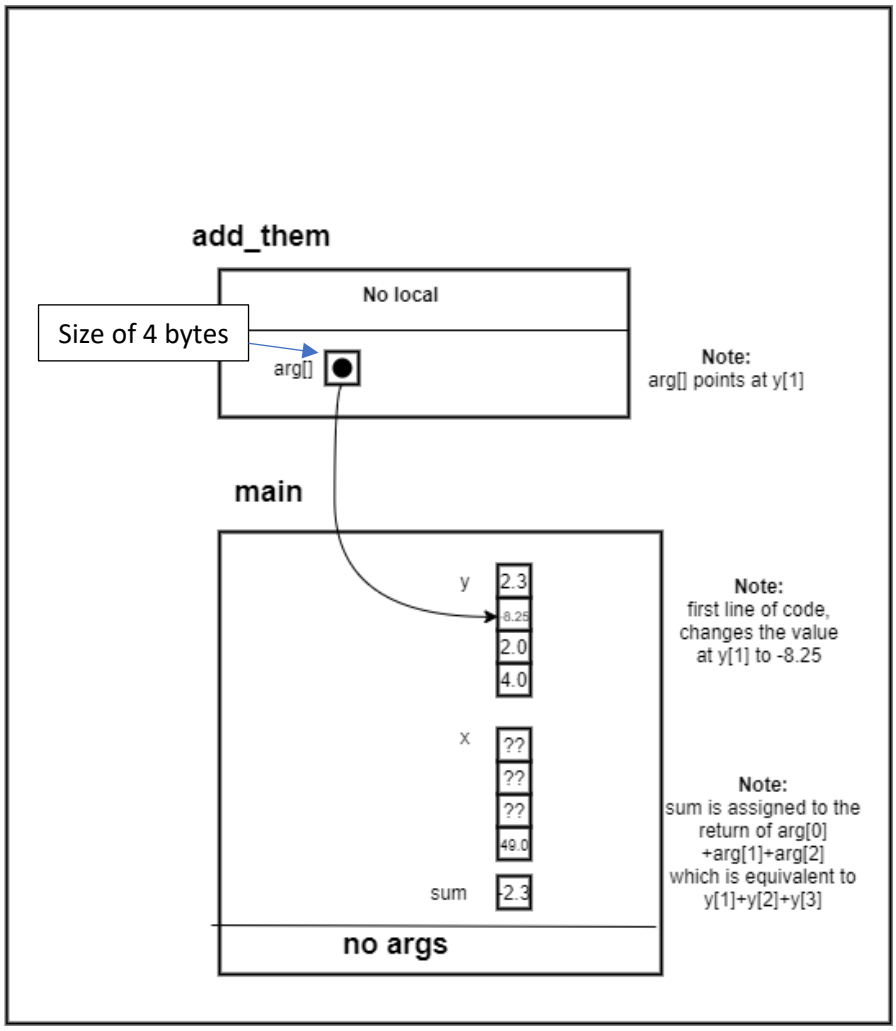

Note:
dest arg of try_to_copy was pointing
to x at beginning then switched to y
try_to_copy
no Local
Size of 4 bytes
dest[]
source[]
Size of 4 bytes
main
y
2.3
1.2
2.0
4.0
x
??
??
??
??
sum
0
no args

c) Point-3:



Note:
dest is pointing at x(x[0]) then in try_to_change dest[3] which is pointing at x[3] is changed to 49.0.

d) Point-4:



2) Exercise: B

a) Code:

```
/*
 *  File Name: lab2exe_B.c
 *  Lab:2
 *  Completed by: Ziad Chemali
 *  Submission Date: 02,10,2020
 *
 *  ENSF 619 Fall 2020 Lab 2 Exercise B
 *
 */

int my_strlen(const char *s);
/*  Duplicates 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);
/*  Duplicates strncat from <string.h>, except return type is void.
 */
int my_strcmp(const char* str1, const char* str2);

#include <stdio.h>
#include <string.h>
#pragma warning(disable:4996)

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_strlen(const char* s) {
    int i = 0;

    while (*s!='\0'){
        i++;
        s += sizeof(char);


    }

    return i;
}
void my_strncat(char* dest, const char* source, int size) {
    int k = 0;
    while(dest [k]!='\0')
    {

        k++;
    }


    for (int i = 0;i < size;i++) {


        dest[k+i] = source[i];

    }
    dest[k + size] = '\0';
}
int my_strcmp(const char* str1, const char* str2) {

    int value_str1 = 0;
    int value_str2 = 0;
    int k = 0;
    while (str1[k] != '\0')
    {
        value_str1 += (int)str1[k];
```

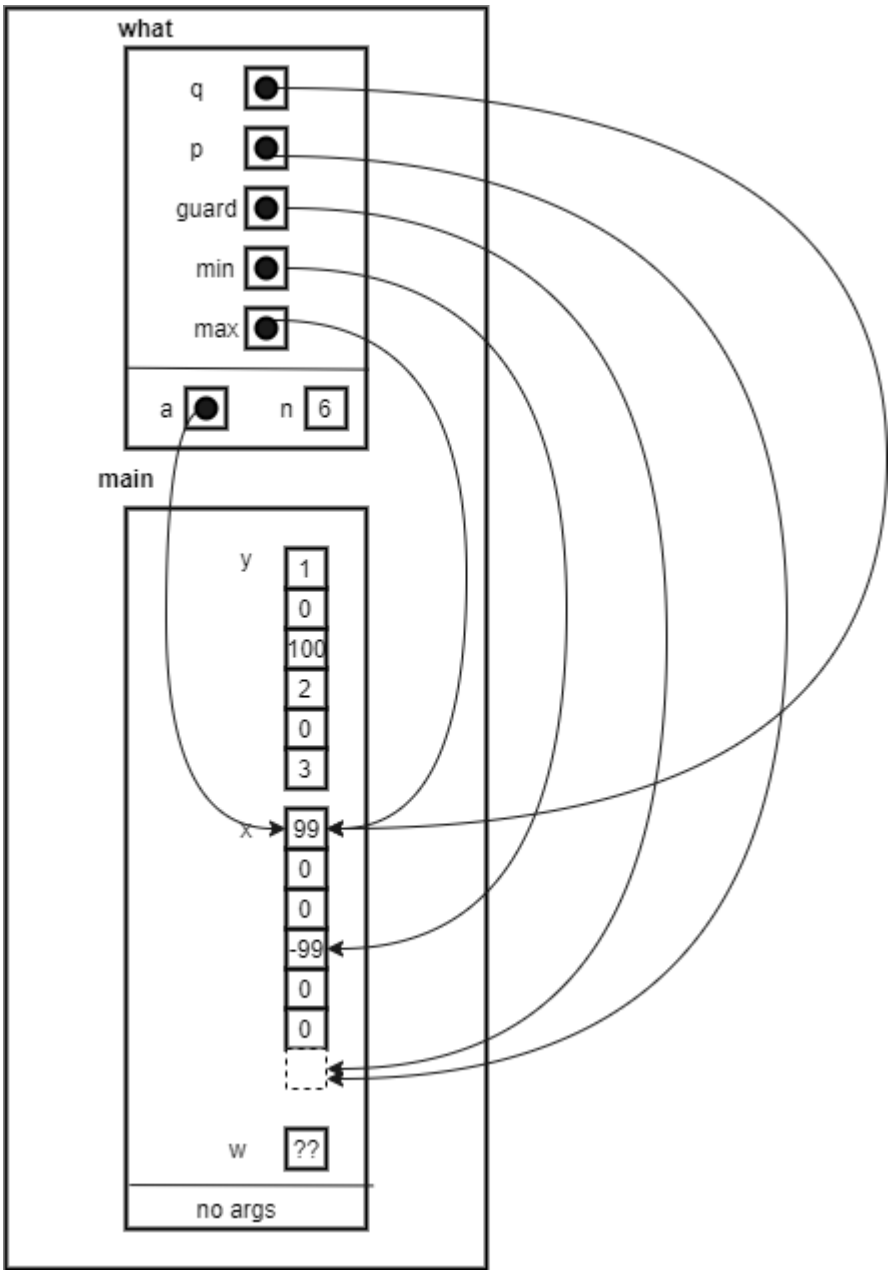
```
        k++;
    }
    k = 0;
    while (str2[k] != '\0')
    {
        value_str2 += (int)str2[k];
        k++;
    }
    if (value_str1 > value_str2)
        return 1;
    else if (value_str1 < value_str2)
        return -1;
    else
        return 0;

}
```

b) Output:

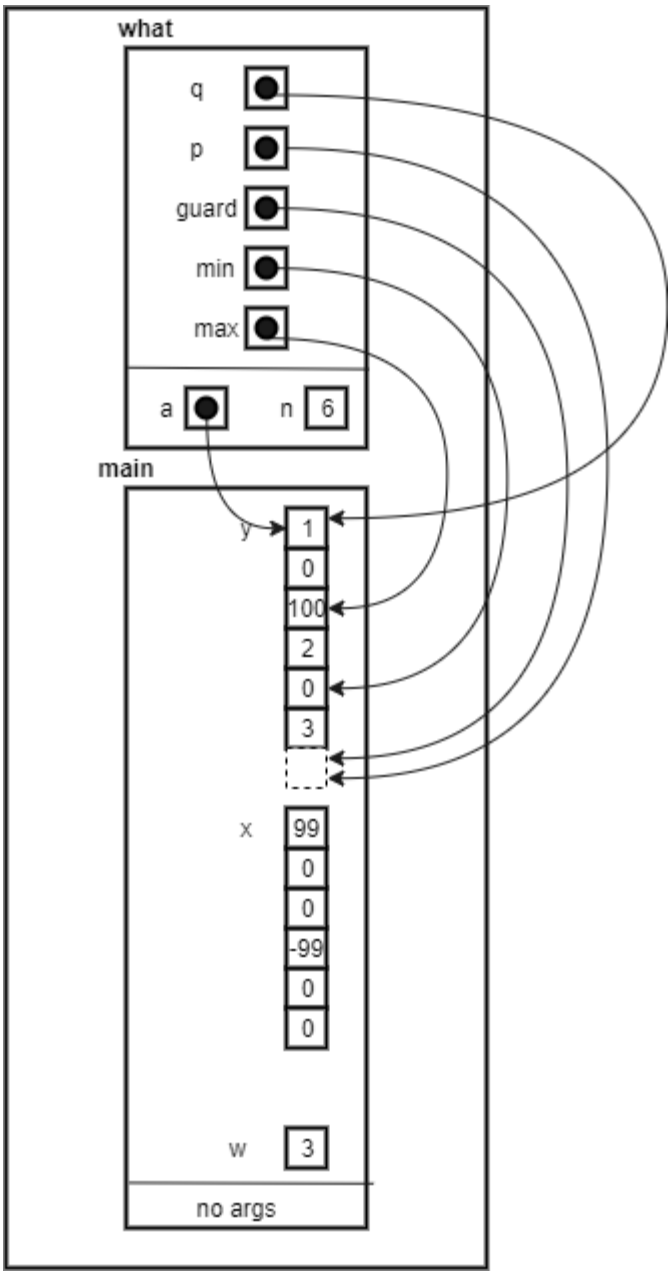
```
Line 1: my_string length is 0.
Line 2: my_string size is 100 bytes.
Line 3: my_string contains: banana
Line 4: my_string length is 6.
Line 5: my_string contains:""
Line 6: my_string length is 0.
Line 7: my_string size is still 100 bytes.
Line 8: my_string contains:"tic"
Line 9: my_string length is 3.
Line 10: my_string contains:"tic-tac"
Line 11: my_string contains:"tic-tac-toe"
Line 12; my_string has 11 characters.

Using my_strcmp - C library function:
"ABCD" is less than "ABCDE" ... my_strcmp returns: -1
"ABCD" is less than "ABND" ... my_strcmp returns: -1
"ABCD" is equal than "ABCD" ... my_strcmp returns: 0
"ABCD" is less than "ABCd" ... my_strcmp returns: -1
"Orange" is greater than "Apple" ... my_strcmp returns: 1
```

3) Exercise: C

a) AR for **first call** of what function until //point one:



b) AR for **Second call** of what function until //point one



4) Exercise E:

a) Code:

```
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) {
	struct cplx result;
	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);
}
```

b) Output:

```
This programs needs values for complex numbers w and z.
  Please enter the real part of w     : 1.5
  Please enter the imaginary part of w: 0.75
  Please enter the real part of z     : -2.5
  Please enter the imaginary part of z: -.5

w is (1.500000) + j(0.750000)
z is (-2.500000) + j(-0.500000)

sum is (-1.000000) + j(0.250000)

diff is (4.000000) + j(1.250000)

multplication is (-3.375000) + j(-2.625000)
```

4) Exercise F:

a) Code:

```
double distance(const struct point* p1, const struct point* p2)
{

    return sqrt(pow(p1->x-p2->x,2)+ pow(p1->y - p2->y, 2)+ pow(p1->z - p2->z, 2));
}
```

b) Output:

```
Size of struct-point in our Linux lab is: 32 bytes.
Size of strcut-point pointer in our Linux lab is: 4 bytes.
Size of strcut that stp points to is:  32 bytes.

Point: A1 <2.30, 4.50, 0.00>
Point: C1 <12.30, 14.50, 56.00>
Point: D1 <125.90, 130.00, 97.00>
The distance between sigma and omega is:     167.11
The distance between sigma and theta is:     292.70
```