Course: ENSF 619- Fall 2020

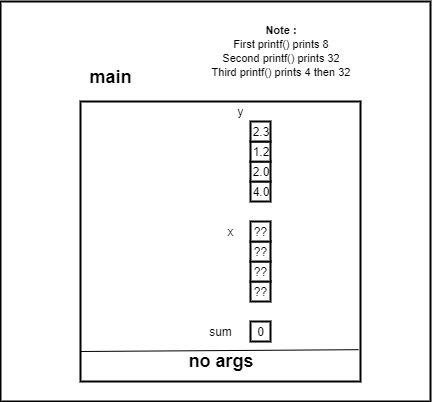
Lab: 2

Student Name: Ziad Chemali

Submission Date: October 2, 2020

# 1) Exercise: A

## a) Point-1

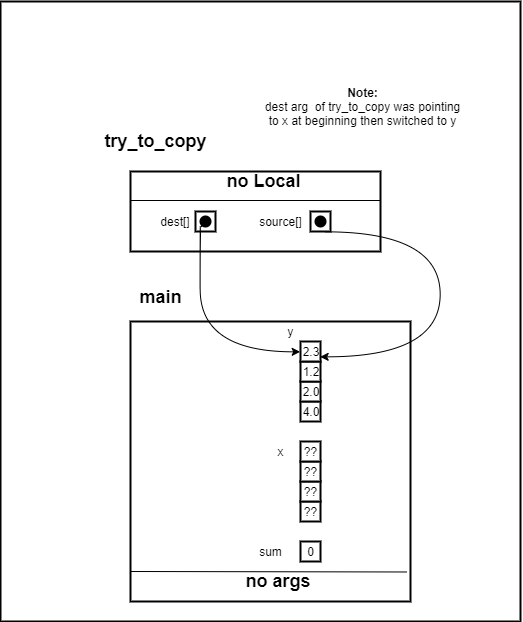


Size of 4 bytes

Size of 32 bytes

Size of 32 bytes

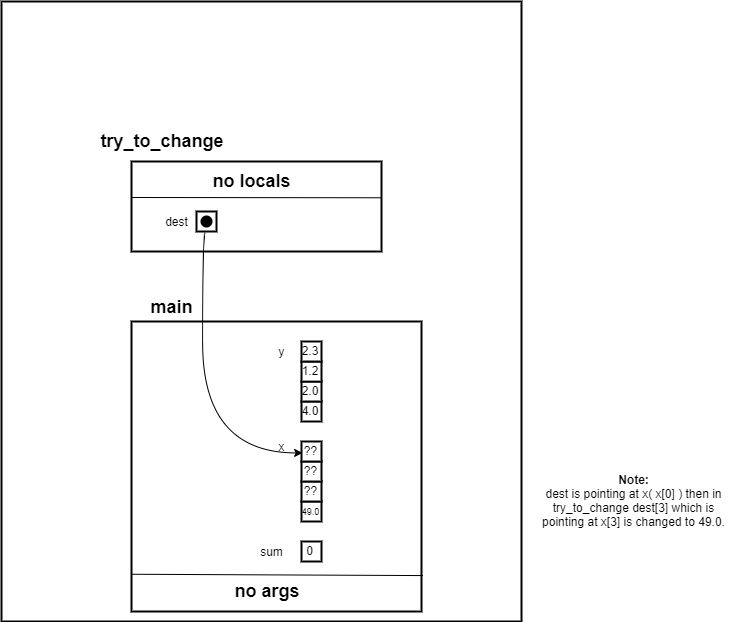
## b)Point-2:



Size of 4 bytes

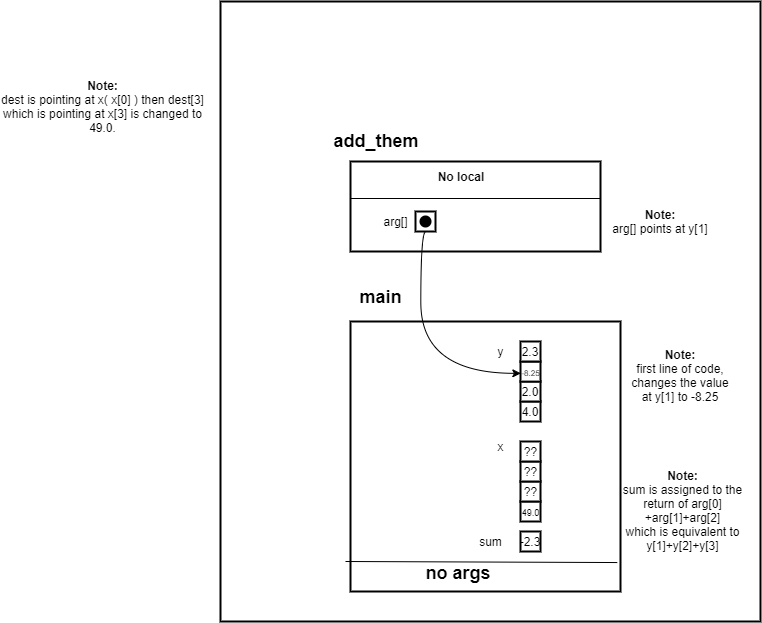
Size of 4 bytes

## c) Point-3:



Size of 4 bytes

## d) Point-4:



Size of 4 bytes

# 2) Exercise: B

## 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)

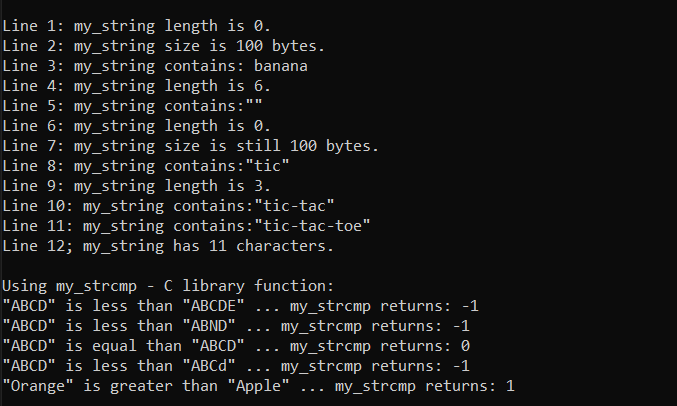
return -1;

else

return 0;

}

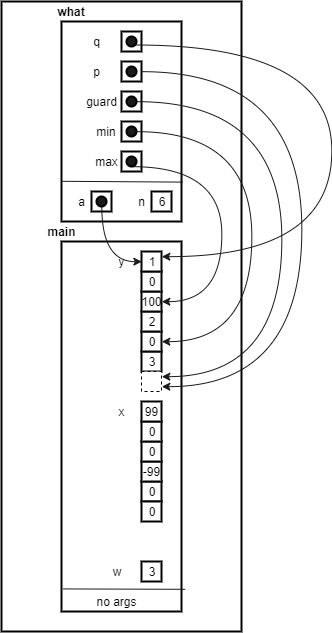
## Output:



# 3) Exercise: C

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

## 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)

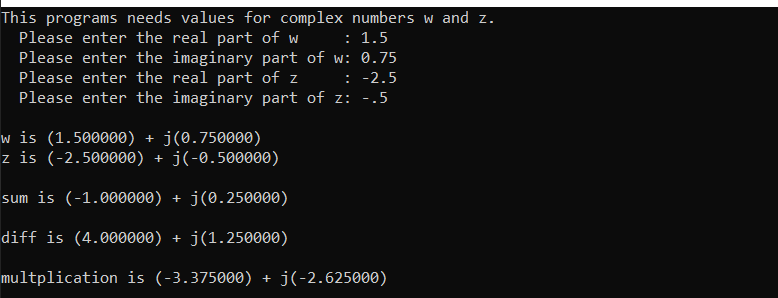
{

product->real = (pz1->real \* pz2->real) - (pz1->imag \* pz2->imag);

product->imag = (pz1->real \* pz2->imag) + (pz1->imag \* pz2->real);

}

## b) Output:



# 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:

