Course: ENSF 614 – Fall 2023
Lab 2:
Instructor: M. Moussavi
Student Name: Emmanuel Alafonye
Submission Date: September 27, 2023

Lab2exe_A

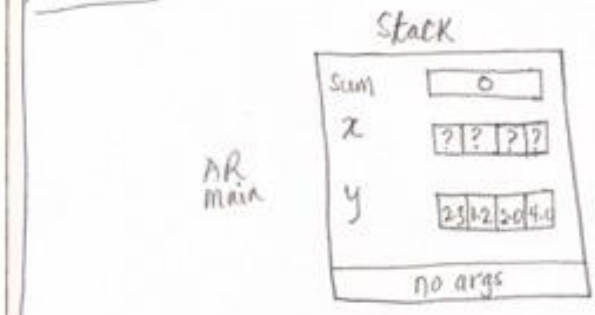
```
/*
 *  File Name: lab2exe_A.cpp
 *  Assignment: ENSF 614 Lab 2, exercise A
 *  Created by Mahmood Moussavi
 *  Completed by: Emmanuel Alafonye
 *  Submission Date: Sept 27, 2023.
 */

/*****************************************************************
 One objective of this program is to use sizeof operator to find the number of
 bytes of memory alloacted for simple varibles, pointers, and arrays.

 The second objective is is to demonstrate how array notations in a function
 argument is still treated as a pointer.

 *****************************************************************/
#include <iostream>
using namespace std;

void try_to_change(double* dest);
void try_to_copy(double dest[], double source[]);
double add_them (double a[5]);

int main(void){
   double sum = 0;
   double x[4];
   double y[] = {2.3, 1.2, 2.0, 4.0};
   cout << " sizeof(double) is " << (int) sizeof(double) << " bytes.\n";
   cout << " size of x in main is: " << (int) sizeof(x) << " bytes.\n";
   cout << " y has " << (int) (sizeof(y)/ sizeof(double)) << " elements and its size is: " <<  (int) sizeof(y) << "
bytes.\n";

   /* Point one */

   try_to_copy(x, y);

   try_to_change(x);

   sum = add_them(&y[1]);
   cout << "\n sum of values in y[1], y[2] and y[3] is: " << sum << endl;

   return 0;
}


void try_to_copy(double dest[], double source[])
{
   dest = source;
```

```
   /* point two*/

   return;
}

void try_to_change(double* dest)
{
   dest [3] = 49.0;

   /* point three*/
   cout << "\n sizeof(dest) in try_to_change is "<< (int)sizeof(dest) << " bytes.\n";
   return;
}


double add_them (double arg[5])
{
   *arg = -8.25;

   /* point four */
   cout << "\n sizeof(arg) in add_them is " << (int) sizeof(arg) << " bytes.\n";
   cout << "\n Incorrect array size computation: add_them says arg has " << (int) (sizeof(arg)/sizeof(double))
<<" element.\n";

   return arg[0] + arg[1] + arg[2];
}
```

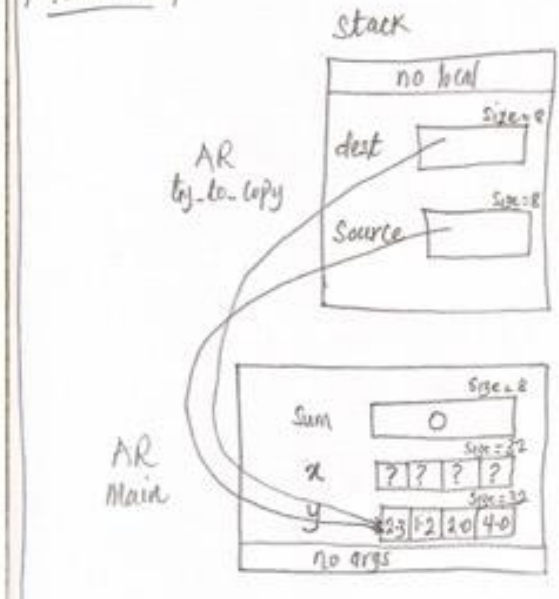
Output:

```
sizeof(double) is 8 bytes.
 size of x in main is: 32 bytes.
 y has 4 elements and its size is: 32 bytes.

 sizeof(dest) in try_to_change is 8 bytes.

 sizeof(arg) in add_them is 8 bytes.

 Incorrect array size computation: add_them says arg has 1 element.

 sum of values in y[1], y[2] and y[3] is: -2.25
Program ended with exit code: 0
```

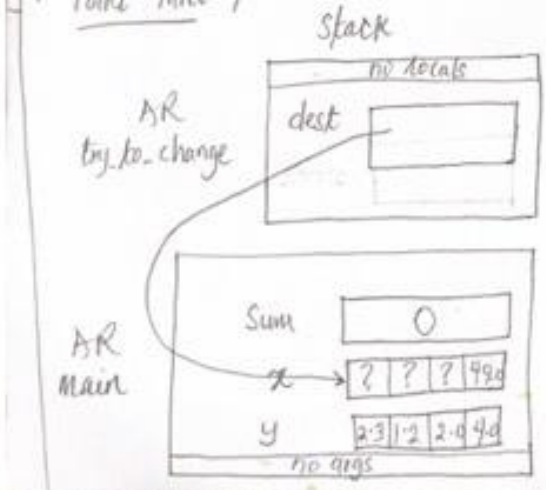
/* Point One */



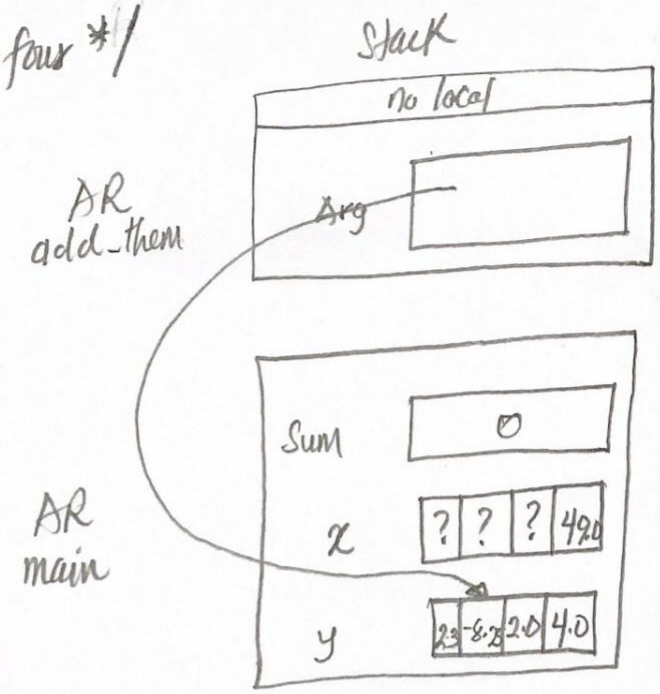
/* Point Two */



/* Point Three */



/*Point four*/



Lab2exe_B

```
/*
 *  File Name: lab2exe_B.cpp
 *  Assignment: ENSF 614 Lab 2, exercise B
 *  Created by Mahmood Moussavi
 *  Completed by: Emmanuel Alafonye
 *  Submission Date: Sept 27, 2023.
 */
#include <iostream>
#include <cstring>
using namespace std;

int my_strlen(const char *s);

void my_strncat(char *dest, const char *source, int n);

int my_strcmp(const char *s1, const char *s2);
/*
 *  Duplicates strcmp from <cstring>.
 *  Compares two strings lexicographically.
 *
 *  REQUIRES
 *    s1 and s2 point to the beginning of strings.
 *  PROMISES
 *    Returns 0 if s1 is equal to s2.
 *    Returns a positive integer if s1 is greater than s2.
 *    Returns a negative integer if s1 is less than s2.
 */

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; // Let is initialized.

   /* using my_strlen custom function */
   length = my_strlen(my_string);
   cout << "\nLine 1: my_string length is " << length;

   /* using sizeof operator */
   bytes = sizeof (my_string);
   cout << "\nLine 2: my_string size is " << bytes << " bytes.";
```

```
/* using strcpy library function */
strcpy(my_string, str1);
cout << "\nLine 3: my_string contains: " << my_string;

length = my_strlen(my_string);
cout << "\nLine 4: my_string length is " << length << ".";

my_string[0] = '\0';
cout << "\nLine 5: my_string contains:\"" << my_string << "\"";

length = my_strlen(my_string);
cout << "\nLine 6: my_string length is " <<  length << ".";

bytes = sizeof (my_string);
cout << "\nLine 7: my_string size is still " << bytes << " bytes.";

/* my_strncat appends the first 3 characters of str5 to the end of my_string */
my_strncat(my_string, str5, 3);
cout << "\nLine 8: my_string contains:\"" << my_string << "\"";

length = my_strlen(my_string);
cout << "\nLine 9: my_string length is " << length << ".";

my_strncat(my_string, str2,  4);
cout << "\nLine 10: my_string contains:\"" << my_string << "\"";

/* my_strncat appends ONLY up to '\0' character from str3 -- not 6 characters */
my_strncat(my_string, str3, 6);
cout << "\nLine 11: my_string contains:\"" << my_string << "\"";

length = my_strlen(my_string);
cout << "\nLine 12; my_string has " << length << " characters.";

cout << "\n\nUsing my_strcmp - Custom function: ";

cout << "\n\"ABCD\" is less than \"ABCDE\" ... my_strcmp returns: " <<
my_strcmp("ABCD", "ABCDE");

cout << "\n\"ABCD\" is less than \"ABND\" ... my_strcmp returns: " <<
my_strcmp("ABCD", "ABND");

cout << "\n\"ABCD\" is equal than \"ABCD\" ... my_strcmp returns: " <<
my_strcmp("ABCD", "ABCD");

cout << "\n\"ABCD\" is less than \"ABCd\" ... my_strcmp returns: " <<
my_strcmp("ABCD", "ABCd");

cout << "\n\"Orange\" is greater than \"Apple\" ... my_strcmp returns: " <<
my_strcmp("Orange", "Apple") << endl;
```

```
    return 0;
}

int my_strlen(const char *s) {
    int length = 0;
    while (*s != '\0') {
        length++;
        s++;
    }
    return length;
}

void my_strncat(char *dest, const char *source, int n) {
    int dest_len = my_strlen(dest);
    int i;
    for (i = 0; i < n && source[i] != '\0'; i++) {
        dest[dest_len + i] = source[i];
    }
    dest[dest_len + i] = '\0';
}

int my_strcmp(const char *s1, const char *s2) {
    while (*s1 != '\0' && *s2 != '\0') {
        if (*s1 != *s2) {
            return (*s1 - *s2);
        }
        s1++;
        s2++;
    }
    if (*s1 == '\0' && *s2 == '\0') {
        return 0;
    }
    if (*s1 == '\0') {
        return -(*s2);
    }
    return *s1;
}
```

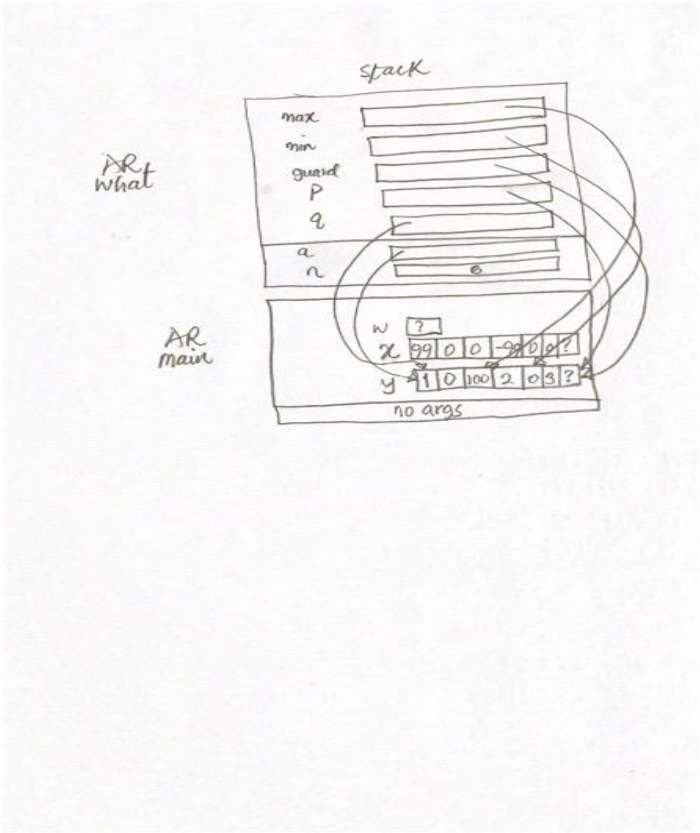
Output

```
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 - Custom function:
"ABCD" is less than "ABCDE" ... my_strcmp returns: -69
"ABCD" is less than "ABND" ... my_strcmp returns: -11
"ABCD" is equal than "ABCD" ... my_strcmp returns: 0
"ABCD" is less than "ABCd" ... my_strcmp returns: -32
"Orange" is greater than "Apple" ... my_strcmp returns: 14
Program ended with exit code: 0

Lab2exe_C




```
/*
 *  File Name: lab2exe_C.cpp
 *  Assignment: ENSF 614 Lab 2, exercise C
 *  Created by Mahmood Moussavi
 *  Completed by: Emmanuel Alafonye
 *  Submission Date: Sept 27, 2023.
 */

#include <iostream>
using namespace std;
```

```
int what(const int *a, int n);

// This function was not written for easy readability!
// It's a drill exercise about pointer arithmetic!
int what(const int *a, int n)
{
  const int *max = a, *min = a + n - 1, *guard = a + n;
  const int *p, *q;
  for (p = a + 1; p != guard; p++) {
    if (*p > *max)
      max = p;
  }
  for (q = a + n - 1; q != a; q--) {
    if (q[-1] < *min)
      min = q - 1;
  }

  // point one (after the 2nd loop has finished)

  return min - max;
}

int main(void)
{
  int w;
  int x[] = {99, 0, 0, -99, 0, 0};
  int y[] = {1, 0, 100, 2, 0, 3};
  w = what(x, sizeof(x) / sizeof(int));
  cout << "1st result: " << w << ".\n";
  w = what(y, sizeof(y) / sizeof(int));
  cout << "2nd result: " << w << ".\n";
  return 0;
}
```

Output

```
1st result: 3.
2nd result: 2.
Program ended with exit code: 0
```

Lab2exe_E

```
/*
 *  File Name: lab2exe_E.cpp
 *  Assignment: ENSF 614 Lab 2, exercise E
 *  Created by Mahmood Moussavi
 *  Completed by: Emmanuel Alafonye
 *  Submission Date: Sept 27, 2023.
 */
```

```
#include <iostream>
#include <stdlib.h>
using namespace std;
#include "lab2exe_E.h"

double read_double_only(void);
/*
 *  Read a double, then skip to the end of a line of input.
 *
 *  REQUIRES
 *    User has been prompted to enter a double.
 *  PROMISES
 *    If user enters bad input, exit is called with an arg of 1.
 *    Otherwise:
 *      Characters following the int are discarded up to
 *      end-of-line or end-of-file, whichever is first.
 *      Return value is the double that was read.
 */

int main(void)
{
  cplx w, z;          /* entered by user */
  cplx sum, diff, prod  ;          /* sum of w and z */
  cout << "This programs needs values for complex numbers w and z.\n";

  cout << "  Please enter the real part of w     : ";
  w.real = read_double_only(); // Accepts the real value of w.real

  cout << "  Please enter the imaginary part of w: ";
  w.imag = read_double_only(); // Accepts the real value of w.image

  cout << "  Please enter the real part of z     : ";
  z.real = read_double_only();
  cout << "  Please enter the imaginary part of z: ";
  z.imag = read_double_only();

  cout << "\nw is (" << w.real << ") + j(" << w.imag << ")\n";
  cout << "z is (" << z.real << ") + j(" << z.imag << ")\n";

  // w = 1.5 + j 0.75, and z = -2.5 - j 0.5

  sum = cplx_add(w, z);
  // Added Diff and product function.
  diff = cplx_subtract(w, z);
  prod = cplx_multiply(w, z);

  cout << "\nsum is (" << sum.real << ") + j(" << sum.imag << "}\n";
  cout << "difference is (" << diff.real << ") + j(" << diff.imag << ")\n";
```

```
  cout << "product is (" << prod.real << ") + j(" << prod.imag << ")\n";

  return 0;
}

double read_double_only(void)
{
  double value_read;
  // int char_code;

  if (!(cin >> value_read)) {
    cout << "Error trying to read in a double.  Program terminated.\n";
    exit(1);
  }

  return value_read;
}



/*
 *  File Name: lab2exe_E.cpp
 *  Assignment: ENSF 614 Lab 2, exercise E
 *  Created by Mahmood Moussavi
 *  Completed by: Emmanuel Alafonye
 *  Submission Date: Sept 27, 2023.
 */

#include "lab2exe_E.h"

cplx cplx_add(cplx z1, cplx z2){
  cplx result;

  result.real = z1.real + z2.real;
  result.imag = z1.imag + z2.imag;
  return result;
}

cplx cplx_subtract(cplx z1, cplx z2){ // Subtract two complex numbers
    cplx result;
    result.real = z1.real - z2.real;
    result.imag = z1.imag - z2.imag;
    return result;
}

cplx cplx_multiply(cplx z1, cplx z2){ // Multiply two complex numbers
    cplx result;
    result.real = (z1.real * z2.real) - (z1.imag * z2.imag);
    result.imag = (z1.real * z2.imag) + (z1.imag * z2.real);
    return result;
```

```
}

/*
 *  File Name: lab2exe_E.cpp
 *  Assignment: ENSF 614 Lab 2, exercise E
 *  Created by Mahmood Moussavi
 *  Completed by: Emmanuel Alafonye
 *  Submission Date: Sept 27, 2023.
 */

#ifndef CPLX_H
#define CPLX_H

struct cplx {
  double real;
  double imag;
};

/*  NOTES:
 *    The following set of function prototypes make for a good
 *    exercise in programming with structs but constitute a BAD module
 *    interface design.  A good interface would use the same pattern
 *    for all four function prototypes.
 *
 *    cplx_add probably has the most convenient interface, because it
 *    lets you write things like
 *
 *       w = cplx_add( z1, cplx_add(z2, z3) );
 *
 *    On the other hand, cplx_multiply probably has the most efficient
 *    interface, because it eliminates any copying of structs.
 */

// cplx cplx_add(cplx z1, cplx z2);
cplx cplx_add(cplx z1, cplx z2)
{
  cplx result;

  result.real = z1.real + z2.real;
  result.imag = z1.imag + z2.imag;
  return result;
}
/* PROMISES: Return value is complex sum of z1 and z2. */

void cplx_subtract(cplx z1, cplx z2,  cplx *difference);
/*
 * REQUIRES
 *   difference points to a variable.
 * PROMISES
```

```
 *   *difference contains complex difference obtained
 *   by subtracting z2 from z1.
 */
// w = 1.5 + j 0.75, and z = -2.5 - j 0.5
void cplx_multiply(const cplx *pz1,
           const cplx *pz2,
           cplx *product);
/*
 * REQUIRES
 *   pz1, pz2 and product point to variables.
 *   pz1 != product && pz2 != product.
 * PROMISES
 *   *product contains complex product of *pz1 and *pz2.
 */
cplx cplx_subtract(cplx z1, cplx z2){ // Subtract two complex numbers
   cplx result;
   result.real = z1.real - z2.real;
   result.imag = z1.imag - z2.imag;
   return result;
}

cplx cplx_multiply(cplx z1, cplx z2){ // Multiply two complex numbers
   cplx result;
   result.real = (z1.real * z2.real) - (z1.imag * z2.imag);
   result.imag = (z1.real * z2.imag) + (z1.imag * z2.real);
   return result;
}
#endif /* ifndef CPLX_H */
```

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: -0.5

w is (1.5) + j(0.75)
z is (-2.5) + j(-0.5)

sum is (-1) + j(0.25}
difference is (4) + j(1.25)
product is (-3.375) + j(-2.625)
```