

Course: ENSF 614 - Fall 2023

Lab #: Lab 2

Instructor: Moussavi

Student Name: Yajur Vashisht, Balkaran Gill

Submission Date: September 27th, 2023

Part A

Part A

Point 1

AR main	Stack					Bytes
	sum	0				8
	x	??	??	??	??	32
	y	2.3	1.2	2.0	4.0	32
	no args					

Point 2

The diagram illustrates the memory layout for two processes, `try_to_copy` and `main`, showing their respective stack frames and registers.

try_to_copy Process:

- AR:** `try_to_copy`
- Stack Bytes:**
 - no local variables:** A section of the stack with no local variables.
 - dest:** A field in the stack frame.
 - source:** A field in the stack frame.

main Process:

- AR:** `main`
- Stack:**
 - sum:** 0
 - x:** ??
 - y:** 2.3
 - no args:** A section of the stack with no arguments.

Data Flow: A red arrow points from the `y` register in the `main` process to the `dest` field in the `try_to_copy` process, indicating data transfer.

Point 3

AR
try_to_change

Stack Bytes		
no local variables		
*dest	•	8

AR
main

Stack					Bytes
sum	0				8
x	??	??	??	49.0	32
y	2.3	1.2	2.0	4.0	32
no args					

Point 4

AR add_them	Stack Bytes			AR main	Stack					Bytes	
	no local variables				sum	-3.95				8	
	arg		8		x	??	??	??	49.0	32	
					y	2.3	-8.25	2.0	4.0	32	
					no args						

Part B

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

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;

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

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

    length = (int) 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.";

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

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

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

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

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

cout << "\n\nUsing my_strcmp: ";

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

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

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

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

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

```

```

    return 0;
}

int my_strlen(const char *s) {

    int i = 0;

    while (s[i] != '\0') {
        i++;
    }

    return i;
}

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';
}

#include <iostream>

int my_strcmp(const char* str1, const char* str2) {
    int sum1 = 0;
    int sum2 = 0;

    while (*str1) {
        sum1 += static_cast<int>(*str1);
        str1++;
    }
}

```

```

while (*str2) {
    sum2 += static_cast<int>(*str2);
    str2++;
}

return sum1 - sum2;
}

```

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_strncmp:

```

"ABCD" is less than "ABCDE" ... my_strncmp returns: -69
"ABCD" is less than "ABND" ... my_strncmp returns: -11
"ABCD" is equal than "ABCD" ... my_strncmp returns: 0
"ABCD" is less than "ABCd" ... my_strncmp returns: -32
"Orange" is greater than "Apple" ... my_strncmp returns: 106
(base) Yajurs-Macbook:Lab 2 yajurvashisht$ █

```

Part C

Part C

AR
what

*max	•
*min	•
*guard	•
*p	•
*q	•
*a	•
n	6

x

y

w	
x[0]	99
x[1]	0
x[2]	0
x[3]	-99
x[4]	0
x[5]	0
y[0]	1
y[1]	0
y[2]	100
y[3]	2
y[4]	0
y[5]	3
y[6]	??
no args	

AR
main

Part E

```
#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;
}

void cplx_subtract(cplx z1, cplx z2, cplx *difference) {
    difference->real = z1.real - z2.real;
    difference->imag = z1.imag - z2.imag;
}

void cplx_multiply(const cplx *pz1, const cplx *pz2, cplx *product) {
    product->real = (pz1->real * pz2->real) - (pz1->imag * pz2->imag);
    product->imag = (pz1->real * pz2->imag) + (pz1->imag * pz2->real);
}
```