

CS2263

Lab 3

Date: February 8th, 2024

Student:

Name: Will Ross

Number: #3734692

Email: will.ross@unb.ca

Due Date: February 12th, 2024

Contents

- [Lab 3](#)
 - [Name](#)
 - [Studnet ID](#)
 - [Course](#)
 - [Date](#)
 - [Contents](#)
 - [Pre Lab](#)
 - [Answer](#)
 - [Exercise 1](#)
 - [Source code](#)
 - [Output Screenshot](#)
 - [Are the pointer variables incremented between successive print operations?](#)
 - [Use the memory addresses printed by your program to calculate the increments used for each pointer variable.](#)
 - [tptr \(int pointer\)](#)
 - [cptr \(char pointer\)](#)
 - [dptr \(double pointer\)](#)
 - [Are the increments for different pointers the same? Explain why.](#)
 - [Exercise 2](#)
 - [Source code](#)
 - [Output Screenshot](#)
 - [Exercise 3](#)
 - [Source code](#)
 - [Output Screenshot](#)
 - [Exercise 4](#)
 - [Compiler command](#)
 - [Source code](#)
 - [Output Screenshot](#)
 - [Diagram of memory locations](#)
 - [Are the results \(i.e. numerical values\) printed from your program different from the results shown in the textbook? Explain why](#)
 - [Memory addresses](#)
 - [Numeric Changes](#)
 - [TextBook Results](#)
 - [My Results](#)

Lab 3

Pre Lab

Briefly (in couple of sentences) explain what is the meaning of the following reference to array `a[]`:

`a[-1]`

Answer

In C, negative indexing of arrays doesn't exist like python or java. C will attempt to access a memory location before the start of the array `a[]` which will lead to a undefined behavior. This causes c to crash and cause a segmentation fault or unexcepted error. All due to accessing memory that the array has not been allocated for.

Exercise 1

Source code

```
// arithmetic1.c
#include <stdio.h>
#include <stdlib.h>
int main (int argc ,char * * argv)
{
    int    arr1[] = {7, 2, 5, 3, 1, 6, -8, 16, 4};
    char   arr2[] = {'m', 'q', 'k', 'z', '%', '>'};
    double arr3[] = {3.14, -2.718, 6.626, 0.529};

    int len1 = sizeof(arr1) / sizeof(int);
    int len2 = sizeof(arr2) / sizeof(char);
    int len3 = sizeof(arr3) / sizeof(double);

    printf("lengths = %d, %d, %d\n", len1, len2, len3);

    int    * iptr = arr1;
    char   * cptr = arr2;
    double * dptr = arr3;

    printf("addresses = %p, %p, %p\n", (void*) iptr, (void*) cptr, (void*) dptr);
    printf("values = %d, %c, %f\n", * iptr, * cptr, * dptr);

    iptr ++;
    cptr ++;
    dptr ++;

    printf("addresses = %p, %p, %p\n", (void*) iptr, (void*) cptr, (void*) dptr);
    printf("values = %d, %c, %f\n", * iptr, * cptr, * dptr);

    iptr ++;
    cptr ++;
    dptr ++;

    printf("addresses = %p, %p, %p\n", (void*) iptr, (void*) cptr, (void*) dptr);
    printf("values = %d, %c, %f\n", * iptr, * cptr, * dptr);

    iptr ++;
    cptr ++;
    dptr ++;

    printf("addresses = %p, %p, %p\n", (void*) iptr, (void*) cptr, (void*) dptr);
    printf("values = %d, %c, %f\n", * iptr, * cptr, * dptr);

    return EXIT_SUCCESS;
}
```

Output Screenshot

```
[q3d5k@gc112m38 Lab3]$ cd "/home1/ugrads/q3d5k/Cs2263/Labs/Lab3/" && gcc arithmetic1.c -o arithmetic1 && "/home1/ugrads/q3d5k/Cs2263/Labs/Lab3/"arithmetic1
lengths = 9, 6, 4
addresses = 0x7ffa9d5b770, 0x7ffa9d5b76a, 0x7ffa9d5b740
values = 7, m, 3.140000
addresses = 0x7ffa9d5b774, 0x7ffa9d5b76b, 0x7ffa9d5b748
values = 2, q, -2.718000
addresses = 0x7ffa9d5b778, 0x7ffa9d5b76c, 0x7ffa9d5b750
values = 5, k, 6.626000
addresses = 0x7ffa9d5b77c, 0x7ffa9d5b76d, 0x7ffa9d5b758
values = 3, z, 0.529000
[q3d5k@gc112m38 Lab3]$
```

Are the pointer variables incremented between successive print operations?

Yes all the pointer variables are being incremented between the successive print operations

Use the memory addresses printed by your program to calculate the increments used for each pointer variable.

tptr (int pointer)

The increment is 4 bytes (0x7ffa9d5b770 - 0x7ffa9d5b774 - 0x7ffa9d5b778 - 0x7ffa9d5b77c)
which is the size of the int in the program being run

cptr (char pointer)

The increment is 1 byte (0x7ffa9d5b76a - 0x7ffa9d5b76b - 0x7ffa9d5b76c - 0x7ffa9d5b76d)
which is the size of the char in the program being run

dptr (double pointer)

The increment is 8 bytes (0x7ffa9d5b740 - 0x7ffa9d5b748 - 0x7ffa9d5b750 - 0x7ffa9d5b758)
which is the size of the double in the program being run

Are the increments for different pointers the same? Explain why.

No depending on the data type the pointers increments will be different, due to the different byte size of the data types

```
int = 4 bytes
char = 1 byte
double = 8 bytes
```

In c when you increment a pointer it moves to the next element of its data type meaning that the three pointers will never increment the same way for this program.

Exercise 2

Source code

```
// arithmetic1.c
#include <stdio.h>
#include <stdlib.h>

void printArray(int arr[], int len){
    int *aptr = arr;
    printf("Index\tValue\tAddress\t\t\t\tValue\n");
    for(int i = 0; i < len; i++){
        printf("%d\t%d\t%p\t\t\t\t\t", i, arr[i], arr[i], *aptr++);

    }
}

int main (int argc ,char * * argv)
{
    //for exercise 2
    int arr[] = {10, 11, 12, 13, 14, 15, 16};
    int len = sizeof(arr) / sizeof(int);
    printArray(arr, len);

    return EXIT_SUCCESS;
}
```

Output Screenshot

```
PS C:\Users\willr\Documents\GitHub\Cs2263\Labs\Lab3> cd "c:\Users\willr\Documents\GitHub\Cs2263\Labs\Lab3\" ; if ($?) { gcc arrPrint.c -o arrPrint } ; if ($?) { .\arrPrint }
Index  Value  Address                Value
0      10     0000000e64dffe80      10
1      11     0000000e64dffe84      11
2      12     0000000e64dffe88      12
3      13     0000000e64dffe8c      13
4      14     0000000e64dffe90      14
5      15     0000000e64dffe94      15
6      16     0000000e64dffe98      16
PS C:\Users\willr\Documents\GitHub\Cs2263\Labs\Lab3> 
```

Activate Windows
Go to Settings to activate Windows.

Exercise 3

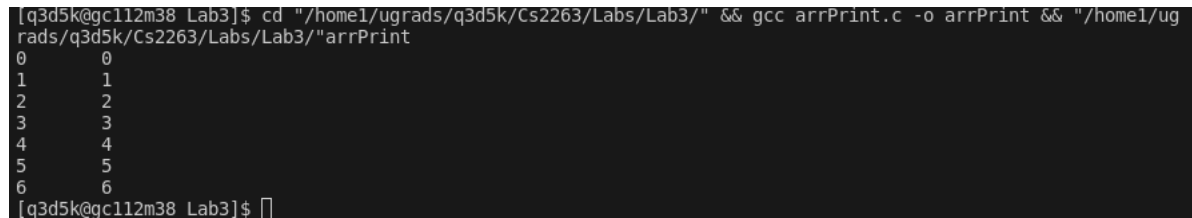
Source code

```
#include <stdio.h>
#include <stdlib.h>

int arrindex(int a[], int * p){
    return p - a;
}

int main (int argc ,char * * argv)
{
    //for exercise 3
    int arr[] = {10, 11, 12, 13, 14, 15, 16};
    for (int i = 0; i < sizeof(arr)/sizeof(arr[0]); i++){
        printf ("%d\t%d\n", i, arrindex( arr, & arr[i]));
    }
    return EXIT_SUCCESS;
}
```

Output Screenshot



```
[q3d5k@qc112m38 Lab3]$ cd "/home1/ugrads/q3d5k/Cs2263/Labs/Lab3/" && gcc arrPrint.c -o arrPrint && "/home1/ugrads/q3d5k/Cs2263/Labs/Lab3/"arrPrint
0      0
1      1
2      2
3      3
4      4
5      5
6      6
[q3d5k@qc112m38 Lab3]$
```

Exercise 4

Compiler command

```
[q3d5k@qc112m38 Lab3]$ cd "/home1/ugrads/q3d5k/Cs2263/Labs/Lab3/" && gcc wrongindex.c -o wrongindex && "/home1/ugrads/q3d5k/Cs2263/Labs/Lab3/"wrong
```

Source code

```
/*
 * wrongindex.c
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char * * argv)
{
    int x = -2;
    int arr[] = {0, 1, 2, 3, 4};
    int y = 15;

    //memory address of x and y
    //printf("& x = %p, & y = %p\n", (void*)& x, (void*)& y);

    printf("& of x = %p,\n& of y = %p\n", & x, & y);

    //one invaild
    printf("& of arr[%d] %d\t%p\n", -1,arr[-1], &arr[-1]);

    //all valid
    for(int i = 0; i < sizeof(arr)/sizeof(arr[0]) + 1; i++){

        printf("& of arr[%d]\t%d\t%p\n", i,arr[i], &arr[i]);

    }

    printf("x = %d, y = %d\n", x, y);

    arr[-1] = 7;
    arr[5] = -23;

    printf("x = %d, y = %d\n", x, y);

    arr[6] = 108;

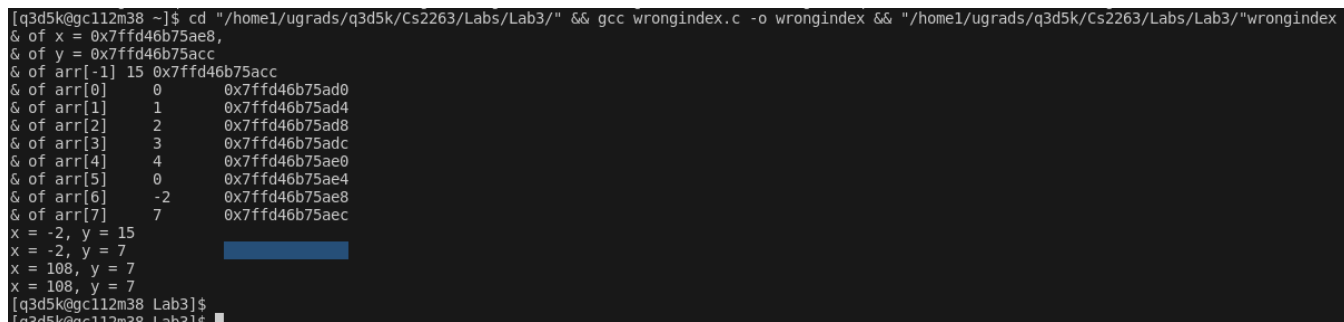
    printf("x = %d, y = %d\n", x, y);

    arr[7] = -353;

    printf("x = %d, y = %d\n", x, y);

    return EXIT_SUCCESS;
}
```

Output Screenshot



```
[q3d5k@gc112m38 ~]$ cd "/home1/ugrads/q3d5k/Cs2263/Labs/Lab3/" && gcc wrongindex.c -o wrongindex && "/home1/ugrads/q3d5k/Cs2263/Labs/Lab3/"wrongindex
& of x = 0x7ffd46b75ae8,
& of y = 0x7ffd46b75acc
& of arr[-1] 15 0x7ffd46b75acc
& of arr[0] 0 0x7ffd46b75ad0
& of arr[1] 1 0x7ffd46b75ad4
& of arr[2] 2 0x7ffd46b75ad8
& of arr[3] 3 0x7ffd46b75adc
& of arr[4] 4 0x7ffd46b75ae0
& of arr[5] 0 0x7ffd46b75ae4
& of arr[6] -2 0x7ffd46b75ae8
& of arr[7] 7 0x7ffd46b75aec
x = -2, y = 15
x = -2, y = 7
x = 108, y = 7
x = 108, y = 7
[q3d5k@gc112m38 Lab3]$
```

Diagram of memory locations

Frame	Symbol	Address	Value
Main	x	0x7ffd46b75ae8	-2
Main	x	0x7ffd46b75ae8	108
Main	y	0x7ffd46b75ae8	15
Main	y	0x7ffd46b75ae8	7
Main	arr[-1]	0x7ffd46b75acc	7
Main	arr[0]	0x7ffd46b75ad0	0
Main	arr[1]	0x7ffd46b75ad4	1
Main	arr[2]	0x7ffd46b75ad8	2
Main	arr[3]	0x7ffd46b75adc	3
Main	arr[4]	0x7ffd46b75ae0	4
Main	arr[5]	0x7ffd46b75ae4	0
Main	arr[6]	0x7ffd46b75ae8	-2
Main	arr[7]	0x7ffd46b75aec	7

Are the results (i.e. numerical values) printed from your program different from the results shown in the textbook? Explain why

Memory addresses

My memory address will be completely different due to c being a hardware low level language so our memory addresses/pointers will most likely always be different.

Numeric Changes

In the text they do the out of bounds call `arr[-1]` there `x` changes to 108 while my `y` changes to 7 when I call `arr[-1]` this means that it does not overlap with `y` in my program due to my personal memory layout.

TextBook Results

```
& x = 0x7fffcabf4e68, & y = 0x7fffcabf4e6c
& arr[0] = 0x7fffcabf4e50, & arr[4] = 0x7fffcabf4e60
x = -2, y = 15
x = -2, y = 15
x = 108, y = 15
x = 108, y = -353
```

As we can see, `x` has changed because of this assignment:

```
arr[6] = 108;
```

Similarly, `y` is changed because of this assignment:

```
arr[7] = -353;
```

My Results

```
[q3d5k@gc112m38 ~]$ cd "/home1/ugrads/q3d5k/Cs2263/Labs/Lab3/" && gcc wrongindex.c -o wrongindex && "/home1/ugrads/q3d5k/Cs2263/Labs/Lab3/"wrongi
& of x = 0x7ffd46b75ae8,
& of y = 0x7ffd46b75acc
& of arr[-1] 15 0x7ffd46b75acc
& of arr[0]    0      0x7ffd46b75ad0
& of arr[1]    1      0x7ffd46b75ad4
& of arr[2]    2      0x7ffd46b75ad8
& of arr[3]    3      0x7ffd46b75adc
& of arr[4]    4      0x7ffd46b75ae0
& of arr[5]    0      0x7ffd46b75ae4
& of arr[6]   -2      0x7ffd46b75ae8
& of arr[7]    7      0x7ffd46b75aec
x = -2, y = 15
x = -2, y = 7
x = 108, y = 7
x = 108, y = 7
[q3d5k@gc112m38 Lab3]$
```