

CPSC 1011 Lab 10 - Pointers

Jasmine Noelle Montalvo

TOTAL POINTS

73 / 100

QUESTION 1

Question 2a 12 pts

1.1 Part 1 4 / 4

✓ **+ 4 pts** Answer is ``10``.

+ **0 pts** Answer is not ``10``.

1.2 Part 2 0 / 4

+ **4 pts** Answer is ``0x7fff8c1ff994``.

✓ **+ 0 pts** Answer is not ``0x7fff8c1ff994``.

1.3 Part 3 0 / 4

+ **4 pts** Answer is ``0x7fff8c1ff988``.

✓ **+ 0 pts** Answer is not ``0x7fff8c1ff988``.

QUESTION 2

2 Question 2b 2 / 4

+ **4 pts** Answer is `*exactly* `integer1= 11`` (with varying white space).

✓ **+ 2 pts** Answer includes ``11``, but is not `*exactly* `integer1= 11`` (with varying white space).

+ **0 pts** Answer does not include ``11``.

💬 Need to specify which variable is 11.

QUESTION 3

3 Question 2c 2 / 4

+ **4 pts** Answer is ``yes`` (or something that otherwise confirms that the output will remain

the same), and may include printed output ``integer1= 11`` (with varying white space) or the answer ``11``.

✓ **+ 2 pts** Answer is not ``yes`` (or something that otherwise confirms that the output will remain the same), but does include printed output ``integer1= 11`` (with varying white space) or the answer ``11``.

+ **0 pts** Answer is not ``yes`` (or something that otherwise confirms that the output will remain the same), and does not include the printed output ``integer1= 11`` (with varying white space) or the answer ``11``.

QUESTION 4

4 Question 2d 2 / 4

+ **4 pts** Answer is ``no`` (or something that otherwise states that the output will change), and may include printed program output.

✓ **+ 2 pts** Answer is not ``no`` (or something that otherwise states that the output will change), but does include printed program output.

+ **0 pts** Answer is not ``no`` (or something that otherwise states that the output will change), and does not include printed program output.

QUESTION 5

5 Question 2e 6 / 6

✓ **+ 6 pts** Answer is similar to the following:

types.

"part (d) is different because it increments the address of what `p1` is pointing to, and then de-references that other memory location"

+ 0 pts Answer is not similar to the following:

"part (d) is different because it increments the address of what `p1` is pointing to, and then de-references that other memory location"

QUESTION 6

6 Question 3 8 / 12

+ 12 pts Answer includes (and/or is similar to) all of the following:

- * `p2` represents a pointer to a character
- * program will compile with warnings
- * incompatible pointer types

✓ **+ 8 pts** Answer includes (and/or is similar to)

**only* two of three of the following:*

- * `p2` represents a pointer to a character
- * program will compile with warnings
- * incompatible pointer types

+ 4 pts Answer includes (and/or is similar to)

**only* one of three of the following:*

- * `p2` represents a pointer to a character
- * program will compile with warnings
- * incompatible pointer types

+ 0 pts Answer does not include or is not at all similar to the following:

- * `p2` represents a pointer to a character
- * program will compile with warnings
- * incompatible pointer types

- 💬 The p1 and p2 are integer and character pointer types, not just integer and character

QUESTION 7

7 Question 4a 4 / 4

✓ **+ 4 pts** Answer is:

`0`

`1`

`2`

`3`

`4`

`5`

`6`

`7`

`8`

`9`

+ 2 pts Answer is: `0-9` with varying white space.

+ 0 pts Answer is not:

`0`

`1`

`2`

`3`

`4`

`5`

`6`

`7`

`8`

`9`

or any collection of `0-9` with varying white space.

QUESTION 8

8 Question 4b 3 / 6

+ 6 pts Answer includes (and/or is similar to) both of the following:

* ``array_of_integers`` evaluates to the location in memory of the beginning of the array, i.e. the first element of the array

* if you de-reference it, you get the first value in the array, which is ``0``

✓ **+ 3 pts** Answer includes (and/or is similar to)

**only* one of the following:*

* ``array_of_integers`` evaluates to the location in memory of the beginning of the array, i.e. the first element of the array

* if you de-reference it, you get the first value in the array, which is ``0``

+ 0 pts Answer does not include or is not similar to the following:

* ``array_of_integers`` evaluates to the location in memory of the beginning of the array, i.e. the first element of the array

* if you de-reference it, you get the first value in the array, which is ``0``

QUESTION 9

9 Question 4c 6 / 6

✓ **+ 6 pts** Answer includes (and/or is similar to) both of the following:

* no

* initializing same array elements via pointer

+ 3 pts Answer includes (and/or is similar to)

**only* one of the following:*

* no

* initializing same array elements via pointer

+ 0 pts Answer does not include or is not similar to the following:

* no

* initializing same array elements via pointer

QUESTION 10

10 Question 4d 6 / 6

✓ **+ 6 pts** Answer includes (and/or is similar to) both of the following:

* no

* printing same array elements via pointer

+ 3 pts Answer includes (and/or is similar to)

**only* one of the following:*

* no

* printing same array elements via pointer

+ 0 pts Answer does not include or is not similar to the following:

* no

* printing same array elements via pointer

QUESTION 11

Question 4e 12 pts

11.1 Part 1 6 / 6

✓ **+ 6 pts** Answer includes (and/or is similar to) all of the following:

* no, output does not change

* de-references the pointer, assigns a value, then increments the pointer

* fills up the array with the correct values in the correct locations

+ 4 pts Answer includes (and/or is similar to)

**only* two of three of the following:*

* no, output does not change

* de-references the pointer, assigns a value, then increments the pointer

* fills up the array with the correct values in the

correct locations

+ 2 pts Answer includes (and/or is similar to)

only one of three of the following:

* no, output does not change

* de-references the pointer, assigns a value, then increments the pointer

* fills up the array with the correct values in the correct locations

+ 0 pts Answer does not include or is not at all similar to the following:

* no, output does not change

* de-references the pointer, assigns a value, then increments the pointer

* fills up the array with the correct values in the correct locations

11.2 Part 2 0 / 6

+ 6 pts Answer includes (and/or is similar to)

both of the following:

* no

* the address of where ``ptr_of_array`` is pointing to after the initialization is now one past the end of the array

+ 3 pts Answer includes (and/or is similar to)

only one of the following:

* no

* the address of where ``ptr_of_array`` is pointing to after the initialization is now one past the end of the array

✓ + 0 pts Answer does not include or is not similar to the following:

* no

* the address of where ``ptr_of_array`` is pointing to after the initialization is now one past the end of the

array

Cannot find answer to this question.

QUESTION 12

12 Question 5a 12 / 12

✓ + 12 pts Answer is:

``a is 1, b is 2``

``i is 2, *pi is 3``

``a is 1, b is 3``

+ 6 pts Answer is:

``a is 1, b is 2``

``i is 2, *pi is 3``

``a is 1, b is 3``

with non-exact but similar white space or capitalization/punctuation.

+ 0 pts Answer is not:

``a is 1, b is 2``

``i is 2, *pi is 3``

``a is 1, b is 3``

or any similar output with varying white space or capitalization/punctuation.

QUESTION 13

13 Question 5b 12 / 12

✓ + 12 pts Answer explains or depicts that ``i`` is incremented locally within the function ``increment``, then the pointer ``pi`` is incremented locally within the function ``increment``, which affects the actual value of ``b`` since it was passed as the argument for ``pi``.

****Note:**** Answers for this question may be highly variable. A similar answer is acceptable as long as it explains why each value is incremented at the right time.

+ 0 pts Answer does not explain or depict that `i` is incremented locally within the function `increment`, then the pointer `pi` is incremented locally within the function `increment`, which affects the actual value of `b` since it was passed as the argument for `pi`.

****Note:**** Answers for this question may be highly variable. A similar answer is acceptable as long as it explains why each value is incremented at the right time.

QUESTION 14

14 Absence Penalty 0 / 0

✓ - **0 pts** Student was present in lab when required.

- **0 pts** Student was not present in lab, but provided a reasonable excuse. Responsible TA(s) will provide more information in comments.

- **50 pts** Student was not present in lab, and did not provide a reasonable excuse ahead of time. Responsible TA(s) will provide more information in comments.

/*Jasmine Montalvo

CPSC 1011 sect.001 Spring 2023

lab 10: pointers

lab description: In this lab we are instructed to answer questions from an answer sheet that is given. We are meant to learn more about pointers and pointer operators. We will also be discussing the relationship between arrays and pointers. Finally, practicing passing pointers as function parameters.

lab notes: This file holds the answers to the questions given on the answer sheet.

*/

//WARM UP:

// 1

// compile and run the given program:

/* #include <stdio.h>

int main(int argc, char*argv[]) {
 int integer1, *p1, **p2;

integer1 = 10;

p1 = &integer1;

p2 = &p1;

*p1++;

printf("integer1= %d\n", *p1);

printf("p2= %p\n", p1);

printf("p2= %p\n", p2);

return 0;

} */

// 2

// a

/* box 1: 0x7fff8c1ff994 10

 * box 2: 0x7fff8c1ff988 0x7fff0db5d324

 * box 3: 0x7fff8c1ff980 0x7fff0db5d328

*/

// b

/* increment= 11

 * p2= 0x7ffc187689f8

```
*/
```

```
// c
```

```
/*integer1= 11
```

```
* p2= 0x7ffdf59e031
```

```
*/
```

```
// d
```

```
/*integer1= 2136481272
```

```
*p2= 0x7fff7f581df8
```

```
*/
```

```
// e
```

```
/*The b and c increment value is the same because there is nothing in  
* the code that changes the value of the increment while part d does  
* change the value of the increment value. The p2 values that are  
* displayed change everytime because even if the data changes, the  
* address values is the one being printed and the address changes all  
* the time.
```

```
*/
```

```
// 3
```

```
// consider the following program and answer the questions below
```

```
/* #include <stdio.h>
```

```
int main(int argc, char *argv[]) {
```

```
    int *p1;
```

```
    char *p2;
```

```
    p2 = p1;
```

```
    return 0;
```

```
} */
```

```
// Q1
```

```
/* p2 could represent the second point on a line that is a char type. */
```

```
// Q2
```

```
/* The program will not compile without a warning. */
```

```
// Q3
/* The programs doesn't compile because p2 cannot be set to p1. They
 * cannot be set as equal because they are different value types one
 * being a character and the other being an integer.
 */
```

// ARRAY AND POINTERS:

// 4

// compile and run the following program & answer the corresponding questions.

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

```
int main(void) {
    int array_of_integers[10], *ptr_of_array, *ptr_of_first_element;
    int i;

    ptr_of_array = array_of_integers;
    ptr_of_first_element = &array_of_integers[0];

    for (i = 0; i < 10; i++)
        array_of_integers[i] = i;

    for (i = 0; i < 10; i++)
        printf("%d\n", *(ptr_of_first_element + i));
        printf("%d\n", *array_of_integers);
    return 0;
}
```

// a

/* The output of the original function is: 0

```

 *           1
 *           2
 *           3
 *           4
 *           5
 *           6
 *           7
 *           8
 *           9
 */
```

//////// // b


```
/* The array_of_integers is equal to [0,1,2,3,4,5,6,7,8,9].
```

```
* If you dereference it you get: 0
```

```
*      1
*      2
*      3
*      4
*      5
*      6
*      7
*      8
*      9
*      0
*/
```

```
// c
```

```
/* The output of the function is: 0
```

```
*      1
*      2
*      3
*      4
*      5
*      6
*      7
*      8
*      9
```

```
* This output doesn't change because the pointer that is dereferenced
* is equal to the array_of_integers.
```

```
*/
```

```
// d
```

```
/* The output of the function is: 0
```

```
*      1
*      2
*      3
*      4
*      5
*      6
*      7
*      8
*      9
```

```
* The output of the function is still the same because the program
```

```

* continues to run through each element of the array when it prints
* each of the values.
*/

```

```

// e

```

```

/* The output of the function is: 0

```

```

*           1
*           2
*           3
*           4
*           5
*           6
*           7
*           8
*           9

```

```

* The output of the function doesn't change because the pointer of the
* array still increase by one when it runs through to program and
* prints the values.
*/

```

```

//FUNCTION AND POINTERS

```

```

// 5

```

```

/* #include <stdio.h>

```

```

void increment(int i, int *pi);

```

```

int main(void) {

```

```

    int a = 1, b = 2;

```

```

    printf("a is %d, b is %d \n", a, b);

```

```

    increment(a, &b);

```

```

    printf("a is %d, b is %d \n", a, b);

```

```

    return 0;

```

```

}

```

```

void increment(int i, int *pi) {

```

```

    i = i + 1;

```

```

    *pi = *pi + 1;

```

```

    printf("i is %d, *pi is %d \n", i, *pi);

```

```
} */
```

```
// a
```

```
/* The output of the function is: a is 1, b is 2
```

```
 *                               i is 2, *pi is 3
```

```
 *                               a is 1, b is 3
```

```
 */
```

```
// b
```

```
/* The first line that is printed shows the values that are assigned in
```

```
 * line 8 of the code above. The second line that is printed is printed
```

```
 * after the i and pi have been raised by an increment of 1. The code
```

```
 * that increments the values by 1 in line 19 and printed from line 20.
```

```
 * The third line printed shows the values of a and b after the
```

```
 * increment function. Since i isn't dereferenced and reassigned, the
```

```
 * value is still equal to 1 in the main function, while pi was
```

```
 * dereferenced and changes the value in b in the main function making
```

```
 * the value of b equal to 3.
```

```
 */
```