# Breakout / Lab 03

Memory Maps & Pointers, Oh My!

CSCI 1730 – Fall 2015

## Problem / Exercise

Pointers are an integral part of C++. While they may be the cause of many headaches, they allow for a great deal of control and flexibility. Because they are low-level constructs, they require a solid understanding of the memory model behind it all. Not every language has pointers, but what you learn about memory management will apply to most every language.
The goals for this lab are:

- Understand and use pointers and pointer operations.
- Discover the relationship between arrays, strings, and pointers.
- Explore passing pointers as function parameters and the `const` modifier with pointers.

This lab is structured differently from previous labs. The for the group and individual brainstorms, simply outline/describe your current understanding of pointers. For the remainder of the exercise, you'll be asked to observe some code and comment on what it's output is. You may work on this with people in your group, however, please submit an individual hard copy at the beginning your next breakout session (include names of your group members).

**NOTE:** While group collaboration is allowed for this lab, please understand that you are still expected to be able to personally reproduce any of your answers as well provide further insight when asked to do so by the instructor or a TA.

## 1 Group Brainstorm

**You are NOT allowed to use the computers during this time.**

Breakup into groups based on your seating and brainstorm about how to solve the problem or exercise. Make sure everyone understands the problem, and sketch out potential ways to move towards a solution. Perhaps something that was discussed during lecture might be useful?

## 2 Submit Individual Brainstorm

**You may use a computer from this point forward.**

Login to eLC and submit a version of your group's brainstorm, written in your own words. You may add additional information if you want. You need to write enough in order to convince the grader that you understand the problem or exercise and that you have a plan for moving forward towards a solution. Please include the last names of the other people in your group in your submission. The brainstorm submission should be available on eLC in your assignment dropbox. We prefer that you submit your individual brainstorms before the end of your breakout period, however, you generally have until 3PM on the day of your breakout (as indicated on eLC) to submit them.

**NOTE:** Submissions that do not include an individual brainstorm will not be graded.

## 3 Submission Instructions

Please submit a hard copy this lab at the beginning of your next breakout session.

## 4 Warmup Activity

1. Compile and run the following C++ program:

```
#include <iostream>
#include <cstdlib>

using std::cout;
using std::endl;

int main() {

    int integer1, * p1, ** p2;
    integer1 = 10;  // line 11
    p1 = &integer1; // line 12
    p2 = &p1;       // line 13

    cout << "integer1 = " << integer1 <<endl; // line 15
    cout << "p1 = " << p1 << endl;            // line 16
    cout << "p2 = " << p2 << endl;            // line 17

    return EXIT_SUCCESS;

} // main
```

2. Suppose after the declaration of variables `integer1`, `p1`, and `p2`, we have the following memory map:

| **Symbol Name** | **Type** | **Memory Address** | **Value** |
|---:|---|:---:|---|
| integer1 | int | 80 | |
| p1 | int * | 84 | |
| p2 | int ** | 92 | |

   (a) Fill in the "Value" column in the in the memory map to reflect the changes that are caused by lines 11, 12, and 13.

   (b) If we substitute lines 15–17 with the the following two statements, then what will be the output of the program?

```
(*p1)++;
cout << "integer1 = " << *p1 << endl;
```

   (c) Will the output of the program be the same if we substitute the above two lines with the following two statements?

```
integer1++;
cout << "integer1 = " << *p1 << endl;
```

   (d) Will the output of the program be the same if we substitute the above two lines with the following two statements?

```
*p2++;
cout << "integer1 = " << integer1 << endl;
```

   (e) Explain why the outputs of parts (c) and (d) are the same or different from each other.

3. Consider the following program and answer the questions below:

```
#include <iostream>
#include <cstdlib>

int main() {
    int * p1;
    int ** p2;
    p2 = p1;
    return EXIT_SUCCESS;
} // main
```

   (a) What does the variable `p2` represent?
   (b) Will this program compile? Why or why note? (Try it!)

## 5 Arrays and Pointers Activity

Now let's look at the relationship between pointer notation and array notation (Hint: They both accomplish the exact the same thing!).

1. Compile and run the following program, then answer the corresponding questions:

```
#include <iostream>
#include <cstdlib>

using std::cout;
using std::endl;

int main() {

    int iarray [10], * p1, * p2;
    p1 = iarray;
    p2 = &iarray[0];

    for (int i = 0; i < 10; ++i) iarray[i] = i;
    for (int i = 0; i < 10; ++i) cout << *(p2 + i) << endl;;

    return EXIT_SUCCESS;

} // main
```

   (a) What is the type of the `iarray` variable (Hint: It's some kind of pointer.)?
   (b) What is the value of `iarray`?
   (c) If you dereference `iarray`, what do you get?
   (d) What is the output of the program?
   (e) Will the output change if we replace the first for-loop with the following? Why or why not?

   ```
   for (int i = 0; i < 10; ++i) *(p1 + i) = i;
   ```

   (f) Will the output change if we replace the second for-loop with the following? Why or why not?

   ```
   for (int i = 0; i < 10; ++i) cout << *(p1 + i) << endl;
   ```

   (g) Will the output change if we replace the first for-loop with the following? Why or why not?

   ```
   for (int i = 0; i < 10; ++i) *(p1++) = i;
   ```

2. Consider the following code:

```
#include <iostream>
#include <cstdlib>

using std::cout;
using std::endl;

int main() {
    int x = 10;
    int * px = &x;
    int a [10] = {100, 100, 100, 100, 100, 100, 100, 100, 100, 100};
    int b [5] = {1000, 1000, 1000, 1000, 1000};
    int * p1 = a;
    int * p2 = b;
    int (* parray)[10] = &a; // line 17
    int * ap1 [3];
    ap1[0] = &x;
    ap1[1] = p1;
    ap1[2] = b;
    int (* ap2 [2]) [10];
    ap2[0] = &a;              // line 25
    ap2[1] = parray;
    return EXIT_SUCCESS;
} // main
```

(a) Complete the memory map given below.

| Symbol | Address | Value | Meaning of Symbol's Type |
|---:|---|---|---|
| x | addr-x | 10 | an `int` |
| px | addr-px | addr-x | a pointer to an `int` |
| a | addr-a | addr-a[0] | an array of 10 `int`s |
| a[0] | addr-a[0] | 100 | an `int` |
| a[1] | | | |
| a[2] | | | |
| a[3] | | | |
| a[4] | | | |
| a[5] | | | |
| a[6] | | | |
| a[7] | | | |
| a[8] | | | |
| b | addr-b | addr-b[0] | an array of 5 `int`s |
| b[0] | addr-b[0] | | an `int` |
| b[1] | | | |
| a[2] | | | |
| a[3] | | | |
| a[4] | | | |
| p1 | | | a pointer to an `int` |
| p2 | | | |
| parray | | | |
| ap1 | | | an array of 3 pointers to `int`s |
| ap1[0] | | | |
| ap1[1] | | | |
| ap1[2] | | | |
| ap2 | | | an array of 2 pointers to arrays of 10 `int`s |
| ap2[0] | | | |
| ap2[1] | | | |

(b) Will the program successfully compile if we substitute line 17 with the following statement? Why or why not?

```
int (* parray) [10] = &b;
```

(c) Will the program successfully compile if we substitute line 17 with the following statement? Why or why not?

```
int (* parray) [] = &b;
```

(d) Will the program successfully compile if we substitute line 25 with the following statement? Why or why not?

```
ap2[0] = &(*parray);
```

## 6 Functions, Pointers, and Tricky Declarations Activity

In this part, you will get to play with pointers as function parameters and practice how to simulate pass-by-reference by passing pointers by value. Also, you will become familiar with complicated mixed representations of pointers, arrays, and functions.

1. What is the output of the following code (assume proper includes, etc)? Why? Which variable in the code is ”passed-by-reference?”

```
void divBy2(int & n) {
    n = n / 2;
} // divBy2

void divBy2(int * np) {
    *np = *np / 2;
} // divBy2

int main() {
    int x = 44;

    divBy2(x);
    cout << x << endl;

    divBy2(&x); // line 16
    cout << x << endl;

    return EXIT_SUCCESS;
} // main
```

2. In the code presented in the previous question, why is `&x` used for the function parameter on line 16?

3. Complete the following table by providing plain English meanings to the corresponding valid C++ declarations.

| **Declaration** | **Meaning** |
|---|---|
| `int x;` | x is an int |
| `int * x;` | x is a pointer to an int |
| `char ** x;` | |
| `int * x [5];` | |
| `int (* x) [5];` | x is a pointer to an array of 5 ints |
| `int (* x [5]) [5];` | |
| `int * (* x [5]) [5];` | x is an array of 5 pointers to arrays of 5 pointers to ints |
| `int x();` | |
| `int x(int);` | |
| `int * x();` | |
| `int * x(int *);` | |
| `int (* x)();` | |
| `int ** (* x)(int **);` | |
| `int (* x [5])();` | |
| `int * ((* x) [5])();` | |

## 7 Const Pointers Activity

1. Consider the following code (assume proper includes, etc):

```
int main() {

    int x = 44;
    int y = 22;

    // part (a)
    int * p1 = &x;
    *p1 = 11;

    // part (b)
    const int * p2 = &x;
    *p2 = 11;
    p2 = &y;

    // part (c)
    const int * const p3 = &x;
    *p3 = 11;
    p3 = &y;

    return EXIT_SUCCESS;

} // main
```

   (a) Which statements of the code marked for part (a) are valid, and which statements are invalid? If a statement is invalid, please explain why.
   (b) Which statements of the code marked for part (b) are valid, and which statements are invalid? If a statement is invalid, please explain why.
   (c) Which statements of the code marked for part (c) are valid, and which statements are invalid? If a statement is invalid, please explain why.

2. Consider the following code (assume proper includes, etc):

```
int main() {

    int a [2] = {1, 2};
    int b [2] = {3, 4};

    // part (a)
    int * p1 = a;
    const int * p2 = a;
    const int * const p3 = a;

    // part (b)
    p1++;
    p2++;
    p3++;

    // part (c)
    a = b;
    a++;
    (*a)++;

    return EXIT_SUCCESS;
} // main
```

(a) Which statements of the code marked for part (a) are valid, and which statements are invalid? If a statement is invalid, please explain why.

(b) Which statements of the code marked for part (b) are valid, and which statements are invalid? If a statement is invalid, please explain why.

(c) Which statements of the code marked for part (c) are valid, and which statements are invalid? If a statement is invalid, please explain why.

3. Consider the following code (assume proper includes, etc):

```
int a [2] = {1, 2};

const int foo() {
    return 2;
} // foo

const int * bar() {
    return a;
} // bar

int * const baz() {
    return a;
} // baz

int main() {

    // part (a)
    int x = foo();
    x++;
    foo()++;

    // part (b)
    const int * p1 = bar();
    p1++
    (*p1)++;
    bar()++;
    (*bar())++;

    // part (c)
    const int * p2 = baz();
    p2++
    (*p2)++;
    baz()++;
    (*baz())++;

    return EXIT_SUCCESS;

} // main
```

(a) Which statements of the code marked for part (a) are valid, and which statements are invalid? If a statement is invalid, please explain why.

(b) Which statements of the code marked for part (b) are valid, and which statements are invalid? If a statement is invalid, please explain why.

(c) Which statements of the code marked for part (c) are valid, and which statements are invalid? If a statement is invalid, please explain why.