**CS 121 Week 10 Worksheet - OOP & Pointers**

**Miscellaneous notes on Pointers:**

* Variables in a program are stored in unique locations in memory that have addresses.
  + - Use the address operator "&" to get the address of a variable (grabbed from PPT):
      * e.g. int num = -23; cout << &num; //prints address of num
* An **aliased** variable is a variable that is linked to another. If one variable is an alias of another and **either one** is changed, both are affected (and possess the same value). Note that they must be of the same type.
  + - Example syntax: int will, &bill = will; //bill is an alias of will
    - It might help to think of the previous example as "I want to set the address of bill equal to will, so I may edit the same memory location with two different identifiers"
* A **pointer** is a variable which stores the memory address of another variable (of the same type). They provide an alternative way to access memory locations.
  + - Similar to an alias variable, a pointer references to another variable (of the same type)
    - You can use pointers to simulate call-by-reference of a function
    - Initialize a pointer by either setting it equal to NULL/nullptr/0, the address of another variable (of the same type), or by allocating new memory to it (dynamic memory allocation)
    - A "NULL" pointer is a pointer that doesn't reference to any memory location (use this to "unlink" if you're pointing to a variable that isn't dynamically-allocated)
      * E.g.: int \* ptr = NULL; //ptr = nullptr; and ptr = 0; work too
* Pointers can be used to make **dynamic memory** (noted by the **new** keyword).
* **Static arrays** are really just **constant pointers** (i.e. they cannot/don't change their memory address unlike typical pointers). If you print out a static array by identifier only, you will get the **address of the first subscript** (same goes for pointers pointing to static arrays).
* When we said that arrays are being passed by reference, that wasn't entirely true. Since arrays are special types of pointers, you're really passing an array **by pointer**.
  + - Note that calling an array subscript is the same thing as dereferencing the element at that given position. (Check CH10, P16 on the PPT for this.)
      * e.g.: int ar[4];

ar[0] = 3; //equivalent is \*(ar + 0) = 3; or \*ar = 3;

ar[3] = 4; //equivalent is \*(ar + 3) = 4;

* + - * When calling \*(ar + 3), we mean (address\_of\_ar[0] + 3 \* size\_of\_int)
        + Meaning, if *ar* was at address 100 (meaning *&ar[0]* is 100) then:

Address of *ar[3]* is: (100 + 3 \*4) = 112

* + - * + As a more general form:

\*(pointer\_identifier + subscript) = (base\_address + subscript \* size\_of\_datatype)

**Classes Practice:**

Create a class that represents a classroom of students and a teacher. Think of **at least** four types of actions (methods) it does, what its member variables it may have, and you only need to create the blueprints of the class (bonus points for a full implementation). Make sure to show examples of creating an instance of the class and calling a few methods from the class.

**HINT:** There should be at least one nested class within the class blueprints, and whatever nested objects you have in the class should have blueprints too.

------------------------------------------------------------------------------------------------------------------------------------------

**Pointer Practice:**

1. Given:

int vals[]={4,7,11};

int \*valptr = vals;

Determine the outputs of the following and provide reasoning on why those are the outputs:

cout << \*(valptr+1);

cout << \*(valptr+2);

cout << (\*valptr)+1;

cout << \*valptr+2;

2. What are four different ways to access an element's position in an array, given that a pointer is pointing to the array? Name each array access method.

1. What is the output of each console output statement? Write answers as comments in the code.

double john = 44, jacob = 33;

double &jingle = john;

john = 41;

cout << john << ":" << jacob << ":" << jingle << endl;

jacob = jingle;

jingle = 12;

cout << john << ":" << jacob << ":" << jingle << endl;

1. Determine the output of the following code and fill the stack diagram on the right. Make sure to write arrows for where pointers point to. Addresses are given in the stack diagram. If values are changed to variables, show the change by crossing out the old one and replacing it with the newer value.

int x = 5;

int y;

int \* ptr = nullptr;

cout << ptr << endl;

ptr = &x;

cout << ptr << endl;

cout << \*ptr << endl;

\*ptr = 10;

cout << x << " " << \*ptr << endl;

1. Same instructions as #2.

double x = 12.0;

double y = -3.2;

double \*ptr, \*ptr2;

ptr = NULL;

ptr2 = 0;

ptr = &x;

ptr2 = ptr;

cout << \*ptr << " " << \*ptr2 << endl;

\*ptr2 = 2.718;

cout << x << " " << \*ptr2

<< " " << \*ptr << endl;

cout << &x << " " << ptr2

<< " " << ptr << endl;

1. Same instructions as #2 and #3. **HINT:** You will need multiple cells for the array (I've given the first position, *list[0]*).

int list[5];

list[4] = 3;

\*(list + 2) = 1;

int\* ptr;

ptr = list;

ptr[3] = 44;

\*(ptr + 1) = 12;

\*ptr = 2;

for(int i = 0; i < 5; ++i)

{

cout << \*(ptr + i) << " ";

}

1. Explain the differences between the two lines below. Assume *ptr* and *ptr2* are pointers.

cout << (ptr == ptr2) << endl;

cout << (\*ptr == \*ptr2) << endl;

1. Convert the following function to pass by-pointer (instead of reference). Make sure to edit *main* to reflect your changes.

void GetEmployeeInfo(string& name, double& salary);

int main()

{

string employeeName;

double pay;

GetEmployeeInfo( employeeName, pay);

cout << "Employee name: " << employeeName << endl;

cout << "Salary: " << pay << endl;

///...etc code

return 0;

}

void GetEmployeeInfo(string& name, double& salary)

{

cin >> name >> salary; //assume user entered "Bob 45003.33"

}

**HINT:** Your function's header will look like the following:

void GetEmployeeInfo(string\* name, double\* salary);