MIDTERM EXAM 2

EMPLID

CSCI 135

NAME: FIRST LAST

1. Write code that creates and sets **int** pointer variables **a, b, c, d, e,** and **f** – to show each of the possibilities below. Include other variable definitions, when appropriate:

a) a pointer to a single automatic integer variable

```
int x = 5;
int * p = & x;
```

b) a pointer to an automatic array of integers

```
int x[3] = {3, 4, 5};
int * p = x; //or int * p = x[0];
```

c) a null pointer

```
int * p = nullptr;
```

d) a pointer to garbage

```
int * p;
```

e) a pointer to a single integer object on heap

```
int * p = new int; //extra for: delete p
```

f) a pointer to a dynamic array of integers

```
int * p = new int[5]; //extra for delete[] p;
```

2. What does the following code print?

```
double a = 1000;
double b = 2000;
double* p = &a;
double* q = p;
b = *q;
p = &b;
a = *p + *q;
cout << a << " " << b << endl;
```

```
2000 1000
```

First, use this table to show how values of variables change as instructions execute. Use the **address-of** operator to show values of pointer variables:

a	b	p	q
1000	2000	&a	&a
2000	1000	&b	

3. Write a function that checks whether all elements in a two-dimensional array have the same value.

```
const int COLUMNS = 3;
bool all_values_identical(int values[][COLUMNS], int rows) {
   for (int i = 0; i < rows; i++) {
      for (int j = 0; j < COLUMNS-1 ; j++) {
         if (values[i][j] != values[i][j+1]) {
            return false;
         }
      }
   }
   return true;









}
```

4. Write a code snippet that will use an array of pointers and dynamic memory to initialize a triangular array of integers with side 4, assign zero to all elements, and print them out -- like this:

```
const int SIZE = 4;
int* counts[SIZE];
// Allocate arrays in dynamic memory
for (int i = 0; i < SIZE; i++) {
   counts[i] = new int[i + 1];
   for (int j = 0; j < i + 1; j++) {
      counts[i][j] = 0;
   }
}
// Print all counts
for (int i = 0; i < SIZE; i++) {
   for (int j = 0; j < i + 1; j++) {
      cout << counts[i][j];
   }
   cout << endl;
}
// Deallocate the rows
for (int i = 0; i < SIZE; i++) {
   delete[] counts[i];




}
```

```
0
00
000
0000
```

5. Design a simple class **`Section`** that contains (or "has") its **`course_name`** and its **`section_number`** , and a simple class **`Student`** that contains (or "has") a **`name`** and a pointer to a **`Section`** object. In the **`main()`** function define three **`Student`** and two **`Section`** objects, and correctly establish the pointer links so that two students be in the first section, and one student in the second section.

```
class Section {
public:
    string course_name;
    int section_number;
};

class Student {
public:
    string name;
    Section * section;
};
```

```
int main() {
    Section section_1;
    sec1.course_name = "MATH100";
    sec1.section_number = 1001;

    Section section_2;
    sec2.course_name = "MATH100";
    sec2.section_number = 1002;

    Student carol;
    carol.name = "Carol";
    carol.section = &section_1;

    Student bob;
    bob.name = "Bob";
    bob.section = &section_1;

    Student alice;
    alice.name = "Alice";
    alice.section = &section_2;
    return 0;
}
```

6. Define an enumerated type **`PhaseOfWater`**, which can hold three possible values: **`SOLID`**, **`LIQUID`**, and **`GASEOUS`**. Write a **`main()`** function that will use a **`switch`** statement, which will hinge on a variable of this type to print: **`"Ice"`**, **`"Water"`**, or **`"Steam"`**.

```
enum PhaseOfWater { SOLID, LIQUID, GASEOUS };

int main() {
    PhaseOfWater phase_1 = GASEOUS;
    switch (phase_1) {
        case SOLID:
            cout << "Ice";
            break;
        case LIQUID:
            cout << "Water";
            break;
        case GASEOUS:
            cout << "Steam";
            break;
    }
}
```