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;
    }
}
```

Variable and Constant Definitions

```
int cans_per_pack = 6;

const double CAN_VOLUME = 0.335;
```

Type, *Name*, *Initial value*

Mathematical Operations

```
#include <cmath>
```

`pow(x, y)`	Raising to a power x^y
`sqrt(x)`	Square root $\sqrt{x}$
`log10(x)`	Decimal log $\log_{10}(x)$
`abs(x)`	Absolute value $\lvert x \rvert$
`sin(x)`, `cos(x)`, `tan(x)`	Sine, cosine, tangent of x (x in radians)

Selected Operators and Their Precedence

(See Appendix B for the complete list.)

`[]`	Array element access
`++ -- !`	Increment, decrement, Boolean *not*
`* / %`	Multiplication, division, remainder
`+ -`	Addition, subtraction
`< <= > >=`	Comparisons
`== !=`	Equal, not equal
`&&`	Boolean *and*
`||`	Boolean *or*
`=`	Assignment

Loop Statements

Condition

```
while (balance < TARGET)
{
   year++;
   balance = balance * (1 + rate / 100);
}
```

Executed while condition is true

Initialization, *Condition*, *Update*

```
for (int i = 0; i < 10; i++)
{
   cout << i << endl;
}
```

Loop body executed at least once

```
do
{
   cout << "Enter a positive integer: ";
   cin >> input;
}
while (input <= 0);
```

Conditional Statement

Condition

```
if (floor >= 13)
{
   actual_floor = floor - 1;
}
else if (floor >= 0)
{
   actual_floor = floor;
}
else
{
   cout << "Floor negative" << endl;
}
```

Executed when condition is true

Second condition (optional)

Executed when all conditions are false (optional)

String Operations

```
#include <string>
string s = "Hello";
int n = s.length(); // 5
string t = s.substr(1, 3); // "ell"
string c = s.substr(2, 1); // "l"
char ch = s[2]; // 'l'
for (int i = 0; i < s.length(); i++)
{
   string c = s.substr(i, 1);
   or char ch = s[i];
   Process c or ch
}
```

Function Definitions

Return type, *Parameter type and name*

```
double cube_volume(double side_length)
{
   double vol = side_length * side_length * side_length;
   return vol;
}
```

Exits function and returns result.

Reference parameter

```
void deposit(double& balance, double amount)
{
   balance = balance + amount;
}
```

Modifies supplied argument

Arrays

Element type, *Length*

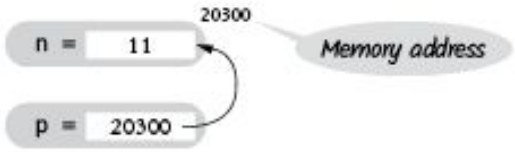
```
int numbers[5];
int squares[] = { 0, 1, 4, 9, 16 };
int magic_square[4][4] =
{
   { 16, 3, 2, 13 },
   { 5, 10, 11, 8 },
   { 9, 6, 7, 12 },
   { 4, 15, 14, 1 }
};

for (int i = 0; i < size; i++)
{
   Process numbers[i]
}
```

Enumerations, Switch Statement

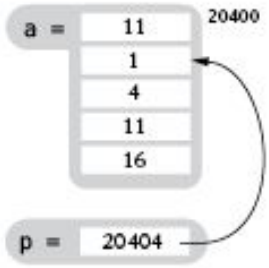
```
enum Color { RED, GREEN, BLUE };
Color my_color = RED;

switch (my_color) {
   case RED :
      cout << "red";   break;
   case GREEN:
      cout << "green"; break;
   case BLUE :
      cout << "blue";  break;
}
```

Pointers

```
int n = 10;
int* p = &n; // p set to address of n
*p = 11; // n is now 11
```




```
int a[5] = { 0, 1, 4, 9, 16 };
p = a; // p points to start of a
*p = 11; // a[0] is now 11
p++; // p points to a[1]
p[2] = 11; // a[3] is now 11
```



Input and Output

```
#include <iostream>
cin >> x; // x can be int, double, string
cout << x;

while (cin >> x) { Process x }
if (cin.fail()) // Previous input failed


#include <fstream>
string filename = ...;
ifstream in(filename);
ofstream out("output.txt");

string line; getline(in, line);
char ch; in.get(ch);
```

Range-based for Loop

```
for (int v : values)   // An array, vector, or other container (C++ 11)
{
   cout << v << endl;
}
```

Output Manipulators

```
#include <iomanip>
```

`endl`	Output new line
`fixed`	Fixed format for floating-point
`setprecision(n)`	Number of digits after decimal point for fixed format
`setw(n)`	Field width for the next item
`left`	Left alignment (use for strings)
`right`	Right alignment (default)
`setfill(ch)`	Fill character (default: space)

Class Definition

```
class BankAccount
{
public:
   BankAccount(double amount);          // Constructor declaration
   void deposit(double amount);         // Member function declaration
   double get_balance() const;          // Accessor member function
   ...
private:                                // Data member
   double balance;
};

void BankAccount::deposit(double amount)   // Member function definition
{
   balance = balance + amount;
}
```

Inheritance

```
class CheckingAccount : public BankAccount   // Derived class, Base class
{
public:
   void deposit(double amount);         // Member function overrides base class
private:
   int transactions;                    // Added data member in derived class
};

void CheckingAccount::deposit(double amount)
{
   BankAccount::deposit(amount);        // Calls base class member function
   transactions++;
}
```