MIDTERM EXAM 2

EMPLID ☐☐☐☐☐☐☐☐☐

CSCI 135   NAME: FIRST LAST ☐

1. Write code that creates `float` pointer variables to show each of the possibilities below. Include other variable definitions, when appropriate:

a) an uninitialized pointer

b) a null pointer

c) a pointer to a single `float` variable with function scope

d) a pointer to an array of `float` values with function scope

e) a pointer to a single `float` object in dynamic memory

f) deallocate the memory in (e) and fix the dangling pointer

g) a pointer to a dynamic array of `float` values

h) deallocate the memory in (g) and fix the dangling pointer

2. Write a function that checks if two `float` arrays are of the same length and have the same values.
```
bool equals(float* a, int a_size, float* b, int b_size) {
```

```
}
```

3. What does the following code print?

```
int x = 15;
int y = 25;
int* s = &x;
int* t = s;
t = &y;
y = *t + 5;
x = *t - *s;
cout << x << " " << y << endl;
```

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:

x	y	s	t

4. Write a function that finds the first occurrence of a value in a two-dimensional array. Return an **int** array of length **2** with the indices of the row and column. For the returned array to persist beyond the scope of your function you must use dynamic memory. Do the clean up in **main()**.

```
const int COLUMNS = 5;
int* find_value(int values[][COLUMNS], int target, int rows) {



}
int main() {
    int array[3][COLUMNS] = {{2, 1, 4, 9, 0}, {1, 0, 2, 7, 4}, {6, 7, 3, 8, 1}};
    int* results = find_value(array, 8, 3); // look for: 8
    cout << "8 found at: " << results[0] << " " << results[1];


}
```

5. Write a code fragments that will use dynamic memory to initialize an "upside down" triangular array of characters with side 5, fill each element with character **'x'**, and print it out, so that it looks like this:

```
//Allocate rows in dynamic memory and set all to X
```

XXXXX

XXXX

XXX

XX

X

(This should remind you of the Galton Board example)

```
// Print out all rows of X's
```

```
// Deallocate the rows
```

6. Design a simple class **Person** that contains (or "has") the **name** of a person and two pointers: to the person's father and mother. In the **main()** function define objects for yourself and your parents, correctly establishing the pointer links. Use **nullptr** for your parents' parents.

```cpp
class Person {                          int main() {
public:




};


                                        }
```

Variable and Constant Definitions

```
 Type    Name      Initial value
  /       /          /
int cans_per_pack = 6;

const double CAN_VOLUME = 0.335;
```

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)	Sine, cosine, tangent of x (x in radians)	
tan(x)		

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;
}


do
{
   cout << "Enter a positive integer: ";
   cin >> input;
}
while (input <= 0);
```
Loop body executed at least once

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]
}
```

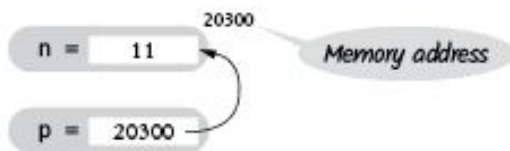## Enumerations, Switch Statement

```cpp
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;
}
```
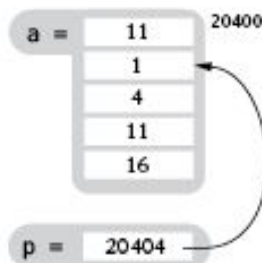
Pointers

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

n = | 11 | 20300
p = | 20300 | *Memory address*

```cpp
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
```

a = | 11 | 20400
    | 1 |
    | 4 |
    | 11 |
    | 16 |
p = | 20404 |

Input and Output

```cpp
#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

An array, vector, or other container (C++ 11)

```cpp
for (int v : values)
{
    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

```cpp
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

Derived class   *Base class*

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

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