

MIDTERM EXAM 2

EMPLID

CSCI 135

NAME: FIRST LAST

1. Write a function: `bool equals(char* a, int a_size, char* b, int b_size)` that checks whether two `char` arrays are of equal length and have the same characters in the same order.

```
bool equals(char* a, int a_size, char* b, int b_size)
{
    if (a_size != b_size) {
        return false;
    }
    for (int i = 0; i < a_size; i++) {
        if (a[i] != b[i]) {
            return false;
        }
    }
    return true;
}
```

2. Write a function: `void bar_chart(int* values, int size)` that displays a bar chart of values in the array `values[]`, using asterisks and dashes, as below. Assume that all values in `values[]` are positive and no larger than 30.

```
void bar_chart(int* values, int size) {

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < values[i]; j++) {
            cout << "*";
        }
        for (int j = values[i]; j < 30; j++) {
            cout << "-";
        }
        cout << endl;
    }
}

int main() {
    const int SIZE = 6;
    int values[SIZE] = {9, 4, 15, 21, 14, 7};
    bar_chart(values, SIZE);
    return 0;
}
```

```
*****-----
****-----
*****-----
*****-----
*****-----
*****-----
```

3. 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. Make sure that this returned array persists beyond the scope of your function without using global or static variables -- use dynamic memory.

```
const int COLUMNS = 4;
int* find_value(int values[][COLUMNS], int target, int rows) {

    bool found = false;
    int * results = new int[2]{-1, -1}; // initialize to not-found
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < COLUMNS && !found; j++) {
            if (values[i][j] == target) {
                results[0] = i;
                results[1] = j;
                found = true;
            }
        }
    }
    return results;
}

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

4. Write a code snippet that will 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:

```
const int SIZE = 5;
char* counts[SIZE];
// Allocate arrays in dynamic memory
for (int i = 0; i < SIZE; i++) {
    counts[i] = new char[SIZE - (i + 1)];
    for (int j = 0; j <= SIZE - (i + 1); j++) {
        counts[i][j] = 'X';
    }
}
// Print all counts
for (int i = 0; i < SIZE; i++) {
    for (int j = 0; j <= SIZE - (i + 1); j++) {
        cout << counts[i][j];
    }
    cout << endl;
}
// Deallocate the rows
for (int i = 0; i < SIZE; i++) {
    delete[] counts[i];
}
```

```
XXXXXX
XXXXX
XXX
XX
X
```

(This should remind you of the Galton Board example)

5. 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.

```
class Person {
public:
    string name;
    Person * father;
    Person * mother;
};

int main() {
    Person mom;
    mom.name = "Carol";
    mom.father = nullptr;
    mom.mother = nullptr;
    Person dad;
    dad.name = "Bob";
    dad.father = nullptr;
    dad.mother = nullptr;
    Person me;
    me.name = "Alice";
    me.father = & dad;
    me.mother = & mom;
    return 0;
}
```

6. Define an enum `TimeOfDay`, which can hold four possible values: `MORNING`, `AFTERNOON`, `EVENING`, and `NIGHT`. Write a `main()` function that will use a `switch` statement, which will hinge on a variable of this type to print the appropriate greeting: "Good morning", "Good afternoon", etc.

```
enum TimeOfDay { MORNING, AFTERNOON, EVENING, NIGHT };

int main() {
    TimeOfDay now = AFTERNOON;
    switch (now) {
        case MORNING:
            cout << "Good morning.";
            break;
        case AFTERNOON:
            cout << "Good afternoon.";
            break;
        case EVENING:
            cout << "Good evening.";
            break;
        case NIGHT:
            cout << "Good night.";
            break;
    }
}
```


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 $ x $
sin(x)	Sine, cosine, tangent of x (x in radians)
cos(x)	
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;
}
```

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;
}
```

Executed when condition is true

Second condition (optional)

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

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]
}
```

Vectors

```
#include <vector>
vector<int> values = { 0, 1, 4, 9, 16 };

vector<string> names;

names.push_back("Ann");
names.push_back("Cindy"); // names.size() is now 2

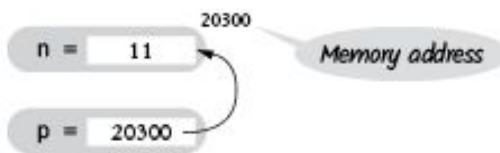
names.pop_back(); // Removes last element

names[0] = "Beth"; // Use [] for element access
```

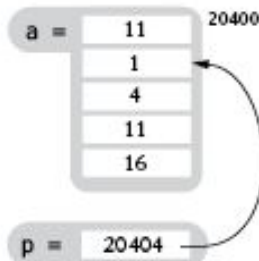
Element type
Initial values (C++ 11)
Initially empty
Add elements to the end

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)
{
    cout << v << endl;
}
```

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

Output Manipulators

```
#include <iomanip>
```

endl	Output new line
fixed	Fixed format for floating-point
setprecision(<i>n</i>)	Number of digits after decimal point for fixed format
setw(<i>n</i>)	Field width for the next item
left	Left alignment (use for strings)
right	Right alignment (default)
setfill(<i>ch</i>)	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:
    double balance; // Data member
};

void BankAccount::deposit(double amount)
{
    balance = balance + amount;
}
```

Member function definition

Inheritance

```
class CheckingAccount : public BankAccount
{
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++;
}
```

Derived class
Base class