

FINAL EXAM

EMPLID

CSCI 135

NAME: FIRST LAST

1 (16%)

MOCK EXAM

MOCK EXAM

MOCK EXAM

MOCK EXAM

MOCK EXAM

(3%) i What is a pointer?

A variable that contains the memory address of another variable.

(2%) ii What does the **&** operator do?

Returns the address of its operand.

(2%) iii What does the ***** operator do?

Depending on the context, it indicates that the declared variable is of a pointer type, or dereferences a pointer variable returning the value, to which it points.

(3%) iv What does the **->** operator do?

Dereferences a pointer to an object of a class type and accesses its member.

(3%) v What is the public interface of a class?

The public interface of a class are its public properties -- functions and variables accessible to users of the class.

(3%) vi Can a function return a pointer to a **dynamic** array? Why?

Yes, because memory for the dynamic array is allocated on the memory heap and persists even after the function returns.

2 (11%)

(5%) i What is the output of the following code snippet? Show your work and put answer in the box:

```
int num = 0;
int* ptr = &num;
*ptr = 80;
num = 90;
cout << *ptr << endl;
```

num

0

80

90

Answer: 90

(6%) ii What is the output of the following code snippet? Show your work and put answer in the box:

```
int arr[10] = { 18, 29, 8, 4, 15, 6, 74, 28, 95, 11 };
int* ptr = arr;
ptr = ptr + 5;
cout << *ptr << endl;
```

*ptr

18

6

Answer: 6

3 (9%) Write a recursive function that determines the number of digits in a positive number n .

Hint: If n is < 10 , it has one digit; otherwise, it has one more digit than $(n/10)$ has.

```
int determine_number_of_digits(int n)
{
    if (n < 10) //base case
    {
        return 1;
    }
    else //recursive case
    {
        return determine_number_of_digits(n/10) + 1;
    }
}
```

4 (18%)

(6%) i Write a function that replaces the value to which `p` points with `x`, if `x` is greater, and always returns the old value to which `p` was pointing.

```
double replace_if_greater(double* p, double x)
{
    double old = *p;
    if (x > old)
    {
        *p = x;
    }
    return x;
}
```

(6%) ii Write a function that reverses in place a non-empty `string s`.

```
void reverse_in_place(string& s)
{
    char tmp;
    for (int i = 0; i < s.length()/2; i++)
    {
        tmp = s[i];
        s[i] = s[s.length() - i - 1];
        s[s.length() - i - 1] = tmp;
    }
}
```

(6%) iii Write a function that removes the last string from a non-empty vector of strings, adds the new string to the vector, and returns the removed string.

```
string replace(vector<string>& my_vector, string new_string)
{
    string old = my_vector.pop_back();
    my_vector.push_back(new_string);
    return old;
}
```

5 (16%)

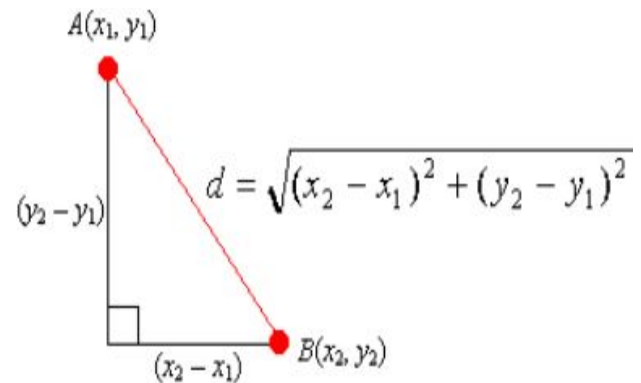
(4%) i Define a simple data-only `class Point` (with only public data members and no member functions — the book would call it a `struct`).

A point has a `double x` and a `double y` coordinates.

```
class Point
{
public:
    double x;
    double y;
};
```

(6%) ii Write a function that computes the distance from `Point a` to `Point b`.

```
#include <cmath>
double compute_distance(Point a, Point b)
{
    double dx = b.x - a.x;
    double dy = b.y - a.y;
    return sqrt(pow(dx, 2) + pow(dy, 2));
}
```



(6%) iii Write a `main()` function that reads the coordinates of the two points from the user input, calls your function, and displays the result.

```
int main()
{
    Point a;
    Point b;
    cout << "x and y of two points: ";
    cin >> a.x >> a.y >> b.x >> b.y;
    cout << compute_distance(a, b);
}
```

6 (18%)

(9%) i Implement a class `Rectangle`. Provide a constructor to construct a rectangle with a given width and length, member functions `get_perimeter()` and `get_area()` that compute the perimeter and area, and a member function void `resize(double factor)` that resizes the rectangle by multiplying the width and length by the given factor.

```
class Rectangle
{
public:
    Rectangle(double l, double w);
    double get_perimeter();
    double get_area();
    void resize(double factor);
private:
    double length;
    double width;
};

Rectangle::Rectangle(double l, double w)
{
    length = l;
    width = w;
}

double Rectangle::get_perimeter()
{
    return 2 * (length + width);
}

double Rectangle::get_area()
{
    return length * width;
}

void Rectangle::resize(
    double factor)
{
    width = length * factor;
    length = width * factor;
}
```

(9%) ii Write individual one line commands to accomplish the following (1.5 points each):

1 Create a local rectangle object of size 2 wide by 4 long;

```
Rectangle r = (2, 4);
```

2 Increase its size by 50%;

```
r.resize(1.5);
```

3 Print out its perimeter and its area;

```
cout << r.get_perimeter << " " << r.get_area << endl;
```

4 Create another rectangle object, but this time in the dynamic memory;

```
Rectangle * r2 = new Rectangle(2, 4);
```

5 Deallocate this object's memory;

```
delete r2;
```

6 Take care of the dangling pointer.

```
r2 = nullptr;
```

7 (12%)

In the game of Tic-Tac-Toe, at any moment each of the nine cells may be either empty or occupied by a cross or a circle. When one of the two players makes her next move, she places her sign (a cross or a circle) on the board. A player wins, when there are three of her signs on the board in a row in a straight line -- the way it is with the line of circles in the picture.

Write the function for next move, which places the player's `Symbol s` on the board at `Position p`, based on the declarations and prototype below. The function must return the outcome of the move: won, did not win, or illegal, if the cell is outside of the board or was already occupied.

```
enum Symbol { CROSS, CIRCLE, EMPTY };
enum Outcome { WON, NOT_WON, ILLEGAL };
class Position {
public:
    int row;
    int col;
};
```



Outcome next_move (Symbol board [3][3], Symbol s, Position p);

```
Outcome next_move (Symbol board[3][3],
Symbol s, Position p)
```

```
{
    if (board[row][col] != EMPTY)
    {
        return ILLEGAL;
    }

    board[row][col] = s;

    if (board[0][0] == board[0][1] &&
        board[0][1] == board[0][2])
    {
        return WON;
    }
    if (board[1][0] == board[1][1] &&
        board[1][1] == board[1][2])
    {
        return WON;
    }
    if (board[2][0] == board[2][1] &&
        board[2][1] == board[2][2])
    {
        return WON;
    }
    if (board[0][0] == board[1][0] &&
        board[1][0] == board[2][0])
    {
        return WON;
    }
}
```

```
    if (board[0][1] == board[1][1] &&
        board[1][1] == board[2][1])
    {
        return WON;
    }
    if (board[0][2] == board[1][2] &&
        board[1][2] == board[2][2])
    {
        return WON;
    }
    if (board[0][0] == board[1][1] &&
        board[1][1] == board[2][2])
    {
        return WON;
    }
    if (board[0][0] == board[1][1] &&
        board[1][1] == board[2][2])
    {
        return WON;
    }
    if (board[2][0] == board[1][1] &&
        board[1][1] == board[0][2])
    {
        return WON;
    }
    else
    {
        return NOT_WON;
    }
}
```

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

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

Executed while condition is true

```
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

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

Condition

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

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

Return type

Parameter type and name

Exits function and returns result.

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

Reference parameter

Modifies supplied argument

Arrays

```
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]
}
```

Element type

Length

Vectors

```
#include<vector> Element type Initial values (C++ 11)
vector<int> values = { 0, 1, 4, 9, 16 };

vector<string> names; Initially empty

names.push_back("Ann"); Add elements to the end
names.push_back("Cindy"); // names.size() is now 2

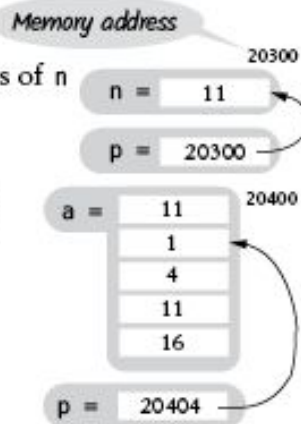
names.pop_back(); // Removes last element

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

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);
```

```
void increment_print() {
    static int s_value = 0; //static duration
    s_value++;
    cout << s_value << '\n';
} //s_value is not destroyed, but goes out of scope

int main() {
    increment_print(); //1
    increment_print(); //2
}
```

```
class Item {
private:
    int m_id;
    static int s_id_counter;
public:
    Item() {
        m_id = s_id_counter++;
    }
    int get_id() const {
        return m_id;
    }
};

int Item::s_id_counter = 1;

int main() { //
    Item first;
    Item second;
    cout << first.get_id(); //1
    cout << second.get_id(); //2
}
```

Static Variables

Static Data Members

Range-based for Loop

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

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;
}
```

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

```
Derived class Base class
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++;
}
```