Row	Seat

Final Exam CSCI 135 Version 1: Programming Design and Analysis

Hunter College, City University of New York

Final Exam Date and Time:19 May 2022, 11:30 – 1:30 PM

Exam Rules
- Show all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes.
- When taking the exam, you may have with you pens and pencils, and the cheat sheet provided.
- You may not use a computer, calculator, tablet, phone, earbuds, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating on examinations, obtaining unfair advantage, and falsification of records and official documents) as serious offenses against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and will pursue cases of academic dishonesty according to the Hunter College Academic Integrity Procedures.

I understand that all cases of academic dishonesty will be reported to the Dean of Students and will result in sanctions.
Name: KEY
Emp ID:
Email:
Signature:
Initial:

Initial:

1. Short answer questions (3-point each).

   (1) Suppose class Undergraduate is derived from class Student, which class is a superclass?

   Student is a superclass of Undergraduate.

   (2) Declare an array of strings, call it **names**. Initialize with “Ann”, “Bob”, “Charles”.

```
string names[] = {"Ann", "Bob", "Charles"};
Another solution: string names[3] = {"Ann", "Bob", "Charles"};
Yet another solution:
        string *names = new string[3];
        names[0] = "Ann";
        names[1] = "Bob";
        names[2] = "Charles";
        //Remember to release dynamic memory of names
        //when no longer need it.
        //delete[] names;
        //names = nullptr;
```

   Warning: string *names = {"Ann", "Bob", "Charles"}; is **NOT** correct.

   (3) Write code to print 1, 2, 4, 8, 16, …, 1024 = 2^{10}, where the next item is twice of the previous one.

Just need to print out the sequence of integers, format is not important, you can have spaces to separate numbers, or comma to separate them, or a number in a line.

```
//num starts from 1, print it, then double its value.
for (int num = 1; num <= 1024; num *= 2)
    cout << num << " ";

Another solution: use pow function from cmath library
for (int i = 0; i <= 10; i++)
    cout << pow(2, i) << endl;
```

Initial:

(4) Given function bool isPrime(int n), which return true if n is a prime integer, false otherwise. Write code to find out **how many** prime integers are in [1, 100].

```
int numPrimes = 0;
for (int num = 1; num <= 100; num++)
    //isPrime(num) returns a boolean,
    //if (isPrime(num)) is the same as
    //if (isPrime(num) == true)
    if (isPrime(num))
        numPrimes++;
```

(5) Given int arr[] = {1, -2, 97}; and int *p = arr; What is the value of *(p+2)?

Answer: 97
Explanation:

- After statement int *p = arr; int pointer p points to arr. That is, p stores the address of the first element of array arr. You may think p as an alias of arr.
- Expression (p + 2) is the address of the third element of arr.
- Expression *(p + 2) means the element who lives in the address stored in (p + 2). So *(p + 2) is the third element of arr, that is, arr[2].

(6) Given a **struct** called Dog, which includes the following data members: breed as a string and weight as a double. Suppose diesel is declared as a variable of Dog. **Write code** to set the weight of diesel to be 25.6.

```
diesel.weight = 25.6;
```

(7) What is output for the following code?

```
vector<int> nums;

for (int i = 0; i < 10; i++)
    nums.push_back(i);

for (int i = 0; i < nums.size(); i++)
    if (nums[i] % 2 == 0)
      cout << nums[i] << endl;
```

By the first for loop, vector nums becomes [0, 1, 2, 3, 4, 5, 6, 7, 8, 9].
By the second for loop, print out even numbers in vector nums, one in a line.
0
2
4
6
8

Initial:

(8) Read the following code. What is the output?

```
int arr[] = {5, 2, 3, 1};
int size = sizeof(arr) / sizeof(arr[0]);

for (int i = 0; i < size-1; i++)
   if (arr[i] > arr[i+1])
     swap(arr[i], arr[i+1]); //function to exchange two given parameters

for (int i = 0; i < size; i++)
   cout << arr[i] << endl;
```

(a) In the first for loop, if the left element is larger than its right neighbor, swap them.
- When i is 0, arr[i] is arr[0] and arr[i+1] is arr[1]. Since arr[0] > arr[1], swap arr[0] with arr[1], array arr changes to [**2**, **5**, 3, 1].
- When i is 1, arr[i] is arr[1] and arr[i+1] is arr[2]. Since arr[1] is 5 and arr[2] is 3, swap arr[1] with arr[2], arr changes to [2, **3**, **5**, 1].
- When i is 2, arr[i] is arr[2] and arr[i+1] is arr[3]. Since arr[2] is 5 and arr[3] is 1, condition arr[2] > arr[3] holds, swap arr[2] with arr[3], arr is [2, 3, **1**, **5**].

(b) In the second loop, print the elements of arr, one element in a line. We get
2
3
1
5

Define out-of-order as follows,
(a) To sort in ascending order, then arr[i] should be smaller or equal to arr[i+1] in a sorted array. If arr[i] > arr[i+1], where 0 <= i < size -1 (why?), then arr[i] and arr[i+1] is out of order.
(b) To sort in descending order, then arr[i] should larger than or equal to arr[i+1] in a sorted array. If arr[i] < arr[i+1], then arr[i] and arr[i+1] are out-of-order.

The key idea of bubble sort is to swap the out-of-order adjacent pairs, sink the current biggest element (bubble) to the end.

(9) Declare and initialize a two-dimensional string array called arr with two rows. The first row is “big”, “medium”, “small”, the second row is “smile”, “giggle”, “laugh”.

```
string arr[][3] = { {"big", "medium", "small"}, {"smile", "giggle", "laugh"} }; or
string arr[2][3] = { {"big", "medium", "small"}, {"smile", "giggle", "laugh"} };
```

Note: for multi-dimensional array in C++, the sizes of the second and higher dimensions must be specified. The size of the first dimension is optional.

(10) Declare the **header** of a function called **sort**, which takes two integers, if the first one is larger than the second one, swap them. Return type is **void**. No need to define the function, **just define the header of the function**.

void sort(int& m, int& n); or
void sort(int&, int&)

Note that & after int cannot be omitted, variables m and n can be different names. & after type in function header means this function is pass by reference.

2. Declare an int variable called size and initialize it to be 10. Create a **two**-dimensional dynamic allocated memory array, call it data, which has size rows, and row indexed at i has (i+1) columns, where i is the index of row and starts from 0.

```
int size = 10;
int **data = new int*[size];
for (int row = 0; row < size; row++)
    data[row] = new int[row+1];
```

Set each element of data to be a random int in [60, 100].

```
for (int row = 0; row < size; row++)
    for (int col = 0; col < row +1; col++)
        data[row][col] = rand() % (100 - 60 +1) + 60;
```

Release dynamically allocated memory of data and handle dangling pointer problem.

```
for (int row = 0; row < size; row++)
{
    delete[] data[row];
    data[row] = nullptr;
}
delete[] data;
data = nullptr;
```

Note: if size is declared as a const, the following implementation also works.

```
//By convention, name of a const uses all capital letters.
//Part 1
const int SIZE = 10;
int *data[SIZE];
for (int row = 0; row < SIZE; row++)
    data[row] = new int[row+1];
```

Initial:

```
//Part 2
for (int row = 0; row < SIZE; row++)
    for (int col = 0; col < row +1; col++)
        data[row][col] = rand() % (100 - 60 +1) + 60;

//Part 3
for (int row = 0; row < SIZE; row++)
{
    delete[] data[row];
    data[row] = nullptr;
}
```

3. Define a **class** called Date, which includes data members, year and month, both as ints.
    - Data member year is an astronomical year, where year 0 means 1 BC, and counts negative years from 2 BC backward (−1 backward), so 100 BC is −99 (per wiki). So, year can be negative.
    - Data member month is an integer between 1 and 12.

    Define a default constructor, set year to be 1900 and month to be 1.

```
Date::Date()
{
    year = 1900;
    month = 1;
}
```

    Define method nextMonth, which add one month to current date. You need to consider the case when current month is December or not.

```
void Date::nextMonth()
{
    if (month == 12)
    {
       month = 1;
       year++;
    }
    else month++;
}
```

    In main function, create a Date object using default constructor, and call its nextMonth method.

```
Date dt;
dt.nextMonth();
    //Warning: cannot write statement
    //cout << dt.nextMonth() << endl;
    //since the return type of nextMonth is void.
```

Initial:

4. Define a function, for a given array of strings, its size, and a target string, return the index of the **last** occurrence of that target if found, otherwise, return -1.

```
int lastIndex(string* arr, int size, string target)
{
    for (int i = size-1; i >= 0; i--)
        if (target == arr[i])
           return i;

    return -1;
}
```

5. Define a function, for an array of integers and its size, return a vector consisting of only positive integers in this array.

```
vector<int> positive(int* arr, int size)
{
    vector<int> result;
    for (int i = 0; i < size; i++)
        if (arr[i] > 0)
           result.push_back(arr[i]);

    return result;
}
```

Initial:

6. Define class square pyramid.
   (1) Data member are side and height, both may contain decimal numbers.
   (2) Define non-default constructor which takes two formal parameters side and height, if this given parameter side is positive, use it to initialize data member side, otherwise, initialize data member side to be 1. If given parameter height is positive, use it to initialize data member height, otherwise, set data member height to be 1.
   (3) Define a method to reset data member side. If the given parameter is positive, then use it to reset data member side, otherwise, do not change the side of the current object.
   (4) Define a method to get data member side.
   (5) Define a method to get the volume of a square pyramid. The formula is $1/3(side)^2 height$.

Name of a class cannot contain spaces. You can name the class as SquarePyramid or SqPyramid or whatever name as long as it is meaningful.

Note that if a formal parameter has the same name as a data member, to distinct data member from the formal parameter, add this-> before data member, where this is a keyword, which is a pointer to the current object. If a data member does not share the same name as a data member, there is no need to add this-> in front of data member.

For example, the following implementations are both ok.

```
//If formal parameter of a constructor/method share the same name as date members,
//must put this-> before data member to distinct a data member from same-name parameter.
SquarePyramid::SquarePyramid(double side, double height)
{
     if (side > 0)
        this->side = side;
        //without this->, code becomes side = side;
        //which does nothing.
     else this->side = 1;
          //without this->, set formal parameter side to 1,
          //constructor needs to set data members,
          //not formal parameters.

     if (height > 0)
        this->height = height;
     else this->height = 1;
}

//The following implementation is also fine.
SquarePyramid::SquarePyramid(double side2, double height2)
{
     if (side2 > 0)
```

```
        side = side2;
     else side = 1;

     if (height2 > 0)
        height = height2;
     else height = 1;
}
```

Approach 1

Separate compilation of header file (suffix hpp) and source code (suffix cpp). When user uses this class, include the header file is enough. This is a prefer approach to define a class.

//------codes of SquarePyramid.hpp----

```
#ifndef Square_Pyramid_H
#define Square_Pyramid_H
class SquarePyramid
{
public:
    SquarePyramid(double side, double height);
    void setSide(double side);
    double getSide() const;
    double volume() const;
private:
    double side;
    double height;
};
#endif
```

//=====codes of SquarePyramid.cpp======

```
#include "SquarePyramid.hpp"
//Define non-default constructor which takes two formal
//parameters side and height,
//if this given parameter side is positive,
//use it to initialize data member side, otherwise,
//initialize data member side to be 1.
//If given parameter height is positive,
//use it to initialize data member height,
//otherwise, set data member height to be 1.
SquarePyramid::SquarePyramid(double side, double height)
{
     if (side > 0)
        this->side = side;
     else this->side = 1;

     if (height > 0)
```

```
        this->height = height;
     else this->height = 1;
}

//Define a method to reset data member side.
//If the given parameter is positive,
//then use it to reset data member side, otherwise,
//do not change the side of the current object.
void SquarePyramid::setSide(double side)
{
    if (side > 0)
       this->side = side;
}

//Define a method to get data member side.
double SquarePyramid::getSide() const
{
    return side;
}

//Define a method to get the volume of a square pyramid.
//The formula is 1/3(side)^2 height.
double SquarePyramid::volume() const
{
    return 1.0 / 3 * side * side * height;
    //Note that 1.0 / 3 cannot write as 1 / 3
    //if it is the first item of the right-hand side expression.
    //However, side * side * height / 3 is fine,
    //since side and height are double types,
    //the numerator is double type.
}
```

Approach 2

Put all codes in one file, but separate declaration from implementation. Need to include the source code to use this class.

//------ content of SqPyramid.cpp ------

```
class SqPyramid
{
public:
    SqPyramid(double side, double height);
    void setSide(double side);
    double getSide() const;
    double volume() const;
```

Initial:

```
private:
    double side;
    double height;
};

SqPyramid::SqPyramid(double side, double height)
{
    if (side > 0)
       this->side = side;
    else this->side = 1;

    if (height > 0)
       this->height = height;
    else this->height = 1;
}

void SqPyramid::setSide(double side)
{
    if (side > 0)
       this->side = side;
}

double SqPyramid::getSide() const
{
    return side;
}

double SqPyramid::volume() const
{
    return 1.0 / 3 * side * side * height;
}
```

Approach 3

Combine declaration and implementation inside a class definition. Unless the class is very simple, we do not want to take this approach.

Square Pyramid is called Pentahedron.

```
class Pentahedron
{
public:
    Pentahedron(double side, double height)
    {
        if (side > 0)
```

Initial:

```
            this->side = side;
        else this->side = 1;

        if (height > 0)
            this->height = height;
        else this->height = 1;
    }

    void setSide(double side)
    {
        if (side > 0)
            this->side = side;
    }

    double getSide() const
    {
        return side;
    }

    double volume() const
    {
        return 1.0 / 3 * side * side * height;
    }

private:
    double side;
    double height;
};
```

Initial:

7. Define a **recursive** function that test whether a given string contains only letters ‘a’ or ‘b’. Also, an empty string by definition is not a string contains only letter ‘a’ or ‘b’.
   Hint: for base cases, you may need to consider a string has no letter or a string has only one letter.

Note that if you do not use recursion, you will not get any point. No repetition statement is allowed in this function. Here are the keys to solve this problem.

(1) If a string is empty, return false.
(2) If a string has only one letter, and that letter is either ‘a’ or ‘b’, return true.*
(3) Otherwise, either the string has only letter but that letter is not ‘a’ or ‘b’, or the string has two or letters. The string contains ‘a’ and ‘b’ if and only if the first letter is either ‘a’ or ‘b’ and the substring except the first letter contains ‘a’ or ‘b’ only.

*If a string has only one letter, and that letter is neither ‘a’ nor ‘b’, by (3), the return is false.

//Only need function only_ab, #include and main function are not needed.

```
#include <iostream>
#include <cassert>
using namespace std;

bool only_ab(string str);
int main()
{
    assert(only_ab("") == false);
    assert(only_ab("a") == true);
    assert(only_ab("b") == true);
    assert(only_ab("ba") == true);
    assert(only_ab("ab") == true);
    assert(only_ab("acb") == false);
    assert(only_ab("abc") == false);
    return 0;
}

bool only_ab(string str)
{
    int size = str.length();

    if (size == 0)
       return false;

    char ch = str[0];
    if (size == 1 && (ch == 'a' || ch == 'b'))
       return true;

    return (ch == 'a' || ch == 'b') &&
           only_ab(str.substr(1));
}
```