Answer:

Row:	Seat:

Final Exam F23 V1
CSCI 13500: Software Analysis and Design 1
Hunter College, City University of New York

December 14, 2023, 9:00 - 11:00 AM, North Building 118

Exam Rules

- Show all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of a provided cheat sheet.
- When taking the exam, you may bring pens and pencils.
- Scratch paper is provided. For your convenience, you may take the scratch paper and cheat sheet off. But make sure not to put solutions to the scratch paper.
- 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:
EmpID:
Email:
Signature:

1 (30 points) Answer the following questions.

(1) Given `string greetings[] = {"Hello", "Hi", "How are you"}`, what is greetings[0][4]?

Answer: greetings[0][4] is `'o'`. Explanation: greetings[0] is the first element of array of strings, which is "Hello". Expression greetings[0][4] is the fifth letter of this string, which is letter 'o'.

(2) Given Employee class, declare that class Nurse as subclass of Employee class with public inheritance.

Answer: `class Nurse : public Employee`

(3) Write code to generate a random int between 3 and 10, where both ends are included. No library is needed.

Answer: Answer: Use rand() % 8 + 3 to generate a random int in [3, 10].

Explanation:

- There are $10 - 3 + 1 = 8$ integers from 3 to 10.
- `rand() % 8` generates a random integer from 0 to 7.
- rand() % 8 + 3 generates a random integer from 3 to 10.

(4) Given `string greeting = "Hello";` What is the value for greeting.substr(2,3)?

Answer: the answer is string `"llo"`.

Explanation: the first parameter is the starting index of the substring, number 2 is the index of the third letter.

The second prarameter in substr method is the length – aka, number of letters – in the substring.

So, greeting.substr(2, 3) means to extract a substring from string `greeting` whose value is "Hello". Starting from the letter indexed at 2 (the third letter), get 3 more letters, the result is `"llo"`.

(5) Write a command to compile and link TestBoard.cpp and Board.cpp to generate a runnable file **prog**.

Answer: `g++ -o prog TestBoard.cpp Board.cpp`

(6) What is the value of 1 / 2 * 5.6 in C++?

Answer: 0

Explanation: division operator / and multiplication operator * have the same precedence, and both are running from left to right. So / runs first in $1/2 * 5.6$, integer division of 1 / 2 is zero, after it is multiplied with 5.6, the result is still zero.

(7) Write **header** of a function called average to return the average of an array of double numbers with size n.

Answer: Use one the following:

```
double average(double* arr, int n);
```

```
double average(double arr[], int n);
```

(8) Given `int arr[] = {4, 3, 2, 1};` What is the value of `*(arr + 1)`?

Answer: 3

Explanation: arr + 1 is the address of arr[1], so *(arr + 1) is the element residing in that address, that is, arr[1], the second element of array `arr`.

(9) Declare and initialize a two-dimensional strings array called **synonyms** with two rows, each row with three columns. The first row is "kind", "gentle", "nice", the second row is "big", "large", "huge".

Answer: Either one of the following will work. The key is the capacity for the second or higher dimension must be specified.

```
string synonyms[2][3] = { {"kind", "gentle", "nice"}, {"big", "large", "huge"} };
```

or

```
string synonyms[][3] = { {"kind", "gentle", "nice"}, {"big", "large", "huge"} };
```

(10) What is output for the following code?

```
vector<int> nums;
for (int i = 12; i >= 0; i--)
    nums.push_back(i);

for (int i = 0; i < nums.size(); i++)
    if (i % 5 == 0)
       cout << nums[i] << " ";

cout << endl;
```

Answer: 12 7 2

Explanation: first put 12, $\cdots$, 0 to vector `nums`. Then

index i	0	1	2	3	4	5	6	7	8	9	10	11	12
nums[i]	12	11	10	9	8	7	6	5	4	3	2	1	0

Print out the elements whose index is a division of 5. So the print out is

12 7 2

Warning: the answer is not ~~10 5 0~~ since we are checking the index, not nums[i] % 5.

(11) What is the output of the following code?

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

int main() {
    int result = 0;
    for (int num = 3; num < 11; num += 4)
        result += num;

    cout << result << endl;
    return 0;
}
```

Answer: $3 + 7 = 10$

Explanation: num starts from 3, as long as it is smaller than 11, add it to variable result. After each round, increased num by 4.

The numbers are elegible to be added are 3 and 7. So result is 10.

You can also think in tabular format.

int result = 0;

num	$num < 11$?	result += num	num += 4
3	yes	result is increased by 3, changes from 0 to 3	num is increased by 4, and num changes to 7
7	yes	result is increased by 7, changes from 3 to 10	num is increased by 4, and num changes to 11
11	no, stop		

(12) What is output for the following code?

```
1    int a = 2;
2    int* p = &a;
3    *p += 3;
4    cout << a << endl;
```

Answer: 5

Explanation: after `int* p = &a`, which saves a's address to pointer p, then $*p$ represents the guy who lives in the address of variable a. Note that no two variables can reside in the same address, so *p is an alias of variable a.

*p += 3; is the same as $a+ = 3$; so a changes from the initital value 2 to become 5.

(13) What is the output for the following code?

```
1  void foo(int& a);
2
3  int main() {
4      int num = 1;
5      foo(num);
6      cout << num << endl;
7      return 0;
8  }
9
10 void foo(int& a) {
11     if ( a % 2 != 0 )
12        a++;
13     else a += 2;
14 }
```

Answer: 2

Symbol & after int function header means this parameter is passed by reference, ie, the original copy of actual parameter `num` is passed to formal parameter `a`. So, `a` is actually `num`.

Inside function body of foo, `a` – the original copy of `num` – is 1 and cannot be divided by 2, so `a` is increased by 1.

Since `a` is also an original copy of `num`, the change applies to `num`.

When function foo finishes and return to its caller, num keeps its its value 2.

(14) What the output when input is 81?

```
cout << "Enter a number: ";
double num;
cin >> num;
switch ((int)num / 10) {
    case 10:
    case 9:  cout << "excellent" << endl;
             break;
    case 8: cout << "good" << endl;
             break;
    case 7: cout << "ok" << endl;
             break;
    case 6: cout << "work hard" << endl;
             break;
    default: cout << "do not give up" << endl;
}
```

Answer: good

Explanation: the result of integer division of 81 / 10 is 8, it is like to divide 81 pens among 10 kids, each kid gets 8 pens.

Then find out the label matching 8 and run the statements until statement break; or the end of switch statement is reached.

(15) What is the panel like when press down in game 1024? The empty cell is 0.

1		1
	1	1
1		1

Answer: After merging, the result looks like.

		1
2	1	2

Some students might select a random cell to place 1, that is ok.

Some students might not draw a framed table, and write the numbers in rows and columns and write 0 for empty cell, that is fine as well.

0 0 0
0 0 1
2 1 2

2 (10 points) Answer the following questions.

(1) Define a function, for an given array of integers with its size, return number of elements that is not zero.

For example, call the function with array with values 1, 0, 2, 0, 1, the size of array is 5, then the return is 3.

Answer:

```
int getNumNonZeros(int arr[], int size) {
    int numNumZeros = 0;
    for (int i = 0; i < size; i++)
        if (arr[i] != 0)
           numNumZeros++;

    return numNumZeros;
}
```

A complete code to define and test the above function is as follows.

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

int getNumNonZeros(int arr[], int size);
//int getNumNonZeros(int* arr, int size); //is also fine

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

    cout << getNumNonZeros(arr, size) << endl; //print 3
    return 0;
}

int getNumNonZeros(int arr[], int size) {
    int numNumZeros = 0;
    for (int i = 0; i < size; i++)
        if (arr[i] != 0)
           numNumZeros++;

    return numNumZeros;
}
```

(2) Define function `void sortByLen(string* a, string* b)`, if the length of $*a$ is larger than the length of $*b$, swap $*a$ with $*b$, otherwise, do nothing. Note that dereference operator * has lower precedence than dot operator.

Answer:

```
void sortByLen(string* a, string* b) {
    if ((*a).size() > (*b).size()) //ok
    //if (a->size() > b->size()) //also ok
       swap(*a, *b);
}
```

A complete code is as follows.

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

void sortByLen(string* a, string* b);

int main() {
    string s1 = "hello";
    string s2 = "hi";

    sortByLen(&s1, &s2);

    cout << "s1 = " << s1 << ", s2 = " << s2 << endl;
    //print s1 = hi, s2 = hello
    return 0;
}

void sortByLen(string* a, string* b) {
    if ((*a).size() > (*b).size()) //ok
    //if (a->size() > b->size()) //also ok
       swap(*a, *b);
}
```

3 (20 points) Programming exercises

(1) Define a function, for a given string, if it contains at least a letter **and** a digit, return true, otherwise, return false.

For example, for string "abc", the return is false. For string "12", the return is false. For "a2", the return is true. For "2ab", the return is true.

Hint: you might use isalpha to check whether a charcter is a letter (alphabetic) or not. Use isdigit to check whether a character is in ['0', '9'].

int isalpha (int c); Check if character is alphabetic

int isdigit (int c); Check if character is decimal digit

You can count the number of occurrences of letters and number of occurrences of digits.

Answer:

```
bool hasLetterDigit(string s) {
    int numLetters = 0;
    int numDigits = 0;
    for (int i = 0; i < s.size(); i++)
        if (isalpha(s[i]))
           numLetters++;
        else if (isdigit(s[i]))
               numDigits++;

    return numLetters > 0 && numDigits > 0;
}
```

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

//test whether a string contains at least an occurrence of digit
//and an occurrence of letter.
bool hasLetterDigit(string s);

int main() {
    cout << boolalpha << hasLetterDigit("abc") << endl; //false
    cout << boolalpha << hasLetterDigit("12") << endl; //false
    cout << boolalpha << hasLetterDigit("1a") << endl; //true
    cout << boolalpha << hasLetterDigit("a12") << endl; //true
    return 0;
}

bool hasLetterDigit(string s) {
    int numLetters = 0;
    int numDigits = 0;
    for (int i = 0; i < s.size(); i++)
        if (isalpha(s[i]))
           numLetters++;
        else if (isdigit(s[i]))
```

```
            numDigits++;

    return numLetters > 0 && numDigits > 0;
}
```

(2) Question on dynamically allocated memory

(a) Define **panel** to be int** type.

Answer: int** panel;

(b) Allocate memory of panel to be a two-dimensional array with 3 rows, each row has 2 columns.

Answer:

```
int numRows = 3;
int numCols = 2;
panel = new int*[numRows];
for (int row = 0; row < numRows; row++)
    panel[row] = new int[numCols];
}
```

(c) Initialize the element of panel indexed at (row)th row and (col)th column to be row * col, where row and col are indices and $0 <= row < 3$ and $0 <= col < 2$.

Answer:

```
for (int row = 0; row < numRows; row++) //numRows can be written as 3
    for (int col = 0; col < numCols; col++) //numCols can be written as 2
        panel[row][col] = row * col;
```

(d) Release the dynamically allocated memory and avoid dangling pointer problem.

Answer:

```
for (int row = 0; row < numRows; row++) { //numRows can be written as 3
    delete[] panel[row];
    panel[row] = nullptr;
}

delete[] panel;
panel = nullptr;
```

A complete code is as follows.

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

int main() {
    int** panel;

    int numRows = 3;
    int numCols = 2;
    panel = new int*[numRows];
    for (int row = 0; row < numRows; row++)
        panel[row] = new int[numCols];

    //set panel[row][col] by row * col
```

```
    for (int row = 0; row < numRows; row++) //numRows can be written as 3
        for (int col = 0; col < numCols; col++) //numCols can be written as 2
            panel[row][col] = row * col;

    //Release the dynamically allocated memory and avoid dangling pointer problem.
    for (int row = 0; row < numRows; row++) { //numRows can be written as 3
        delete[] panel[row];
        panel[row] = nullptr;
    }

    delete[] panel;
    panel = nullptr;

    return 0;
}
```

4 (10 points) Write codes of vector

Define a function, for a given vector of strings, return a vector of all strings with even length.

For example, call the above function on a vector of strings with values “ab”, “ccd”, “abcd”, the return is a vector of strings with values “ab” and “abcd”.

Answer:

```
vector<string> evenLen(vector<string> vec) {
    vector<string> result;

    for (int i = 0; i < vec.size(); i++)
        if (vec[i].size() % 2 == 0) //vec[i].size() can be replaced by vec[i].length()
          result.push_back(vec[i]);

    return result;
}
```

A complete code is shown as follows.

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

vector<string> evenLen(vector<string> vec);

int main() {
    vector<string> vec = {"ab", "ccd", "abcd"};

    vector<string> result = evenLen(vec);

    for (int i = 0; i < result.size(); i++)
        cout << result[i] << endl;
        //print
        //ab
        //abcd

    return 0;
}

vector<string> evenLen(vector<string> vec) {
    vector<string> result;

    for (int i = 0; i < vec.size(); i++)
        if (vec[i].size() % 2 == 0) //vec[i].size() can be replaced by vec[i].length()
          result.push_back(vec[i]);

    return result;
}
```

5 (10 points) Define a class.

Here is Course.hpp of class **Course**.

```
#include <string>
class Course {
public:
    ...//omitted
private:
    std::string name; //represent course name
    int credit; //represent number of credit hour
};
```

Your job: define Course.cpp with the following requirement.

1. Include necessary library and header file.
2. Define a default constructor, which sets data member **name** to be "CS 135" and set data member **credit** to be 4.
3. Define a non-default constructor, which takes formal parameters name, a string, and credit, an int. Set data member **name** by given parameter name. If given parameter credit is positive, use it to set data member **credit**, otherwise, set data member **credit** to be 3.
4. Define method **getCredit** to return the value of data member **credit**.

Answer:

```
#include "Course.hpp"
#include <iostream>
#include <string> //need in some C++ versions
using namespace std;

Course::Course() {
    name = "CS 135";
    credit = 4;
}

Course::Course(string name, int credit) {
    this->name = name;
    if (credit > 0)
       this->credit = credit;
    else this->credit = 3;
}

int Course::getCredit() const {
    return credit;
}
```

A complete code is as follows.
code of Course.hpp

```
#ifndef Course_H
#define Course_H
#include <string> //need in some C++ version
class Course {
public:
    Course();
    Course(std::string name, int credit);
    std::string getName() const;
    int getCredit() const;
    void setName(std::string name);
    void setCredit(int credit);
private:
    std::string name; //it is discouraged to use namespace in hpp,
        //it might affect all the source code including this header file
    int credit;
};
#endif
```

code of Course.cpp

```
#include "Course.hpp"
#include <iostream>
#include <string> //need in some C++ versions
using namespace std;

Course::Course() {
    name = "CS 135";
    credit = 4;
}

Course::Course(string name, int credit) {
    this->name = name;
    if (credit > 0)
       this->credit = credit;
    else this->credit = 3;
}

string Course::getName() const {
    return name;
}

int Course::getCredit() const {
    return credit;
}

void Course::setName(string name) {
    this->name = name;
}

```

```
void Course::setCredit(int credit) {
    if (credit > 0)
       this->credit = credit;
}
```

content of TestCourse.cpp

```
#include <iostream>
#include "Course.hpp"

using namespace std;

int main() {
    Course cs;

    cout << "name: " << cs.getName() << endl;
    cout << "credit: " << cs.getCredit() << endl;

    cs.setName("CS 235");
    cs.setCredit(3);

    cout << "name: " << cs.getName() << endl;
    cout << "credit: " << cs.getCredit() << endl;
    return 0;
}
```

6 (10 point) Define a subclass

Here are part of Person.hpp of Person class.

```
class Person {
public:
    Person(string name, int age); //non-default constructor of Person class
    virtual string toString() const; //return a textual information of name and age.
    ...//omit other constructors and methods
private:
    string name;
    int age;
};
```

Declare Employee as a subclass of Person. Each employee is a person, with additional data member **salary**, which may contain decimal numbers. Suppose Employee.hpp is declared. In Employee.cpp, do the following:

Define non-default constructor of Employee, which takes parameters name (a string), age (an int), and salary (a double) to initialize the corresponding data members. This constructor can invoke the corresponding constructor of its super class, then initialize data member unique to the subclass. Data member salary should be positive. If parameter salary is not positive, set data member salary to be 1000.

Answer:

```
Employee::Employee(string name, int age, double salary) : Person(name, age) {
     if (salary <= 0)
        this->salary = 1000;
     else this->salary = salary;
}
```

Override toString method inherited from Person class to return a string representing the employee's information like name, age, and salary. You may use `string to_string (double val);` from std namespace to convert double number val to a string. Also, you can call toString method in the superclass.

Answer:

```
string Employee::toString() const {
    string str = Person::toString();
    str += "salary: " + to_string(salary) + "\n";
    //to_string(double) belongs to std namespace

    return str;
}
```

(optional) A complete code is as follows.
code of Person.hpp

```
#ifndef Person_H
#define Person_H
#include <string> //needed

//we normally do not add using namespace std;
//in a header file (ended with .hpp),
//since the source code that include
```

```
//the header file may not like to use that namespace.

class Person {
public:
    Person();
    Person(std::string name, int age);
    std::string getName() const;
    int getAge() const;
    void setAge(int age);
    void setName(std::string name);
    virtual std::string toString() const;

private:
    std::string name;
    int age;
};
#endif
```

code of Person.cpp

```
#include <iostream>
#include <string>
#include "Person.hpp"
using namespace std;

Person::Person() {
    name = "John Doe";
    age = 18;
}

Person::Person(string name, int age) {
    this->name = name;
    if (age >= 0 && age <= 130)
       this->age = age;
    else this->age = 18;
}

string Person::getName() const {
    return name;
}

int Person::getAge() const {
    return age;
}

void Person::setName(string name) {
    this->name = name;
}

void Person::setAge(int age) {
```

```
     if (age >= 0 && age <= 130)
        this->age = age;
}

string Person::toString() const {
    string str = "";
    str += "name: " + name + "\n";
    str += "age: " + to_string(age) + "\n";
    return str;
}
```

code of Employee.hpp

```
#ifndef Employee_H
#define Employee_H
#include <string>
#include "Person.hpp"

class Employee : public Person {
public:
    Employee();
    Employee(std::string name, int age, double salary);
    double getSalary() const;
    void setSalary(double salary);
    virtual std::string toString() const;
private:
    double salary;
};
#endif
```

code of Employee.cpp

```
#include "Employee.hpp"
#include <string>
using namespace std;

//invoke default construtor of Person
Employee::Employee() : Person() {
    salary = 1000;
}

Employee::Employee(string name, int age, double salary) : Person(name, age) {
     if (salary <= 0)
        this->salary = 1000;
     else this->salary = salary;
}

double Employee::getSalary() const {
    return salary;
}

```

```
void Employee::setSalary(double salary) {
    if (salary > 0)
       this->salary = salary;
}

string Employee::toString() const {
    string str = Person::toString();
    str += "salary: " + to_string(salary) + "\n"; //to_string(double) belongs to std namespace
    return str;
}
```

code of TestPersonEmployee.cpp

```
#include <iostream>
#include "Person.hpp"
#include "Employee.hpp"
using namespace std;
//sample output:
//name: Ann
//age: 27
//
//name: Bob
//age: 35
//salary: 20000.000000
//
//name: John Doe
//age: 18
//salary: 1000.000000
int main() {
    int size = 3;
    Person** personPtr = new Person*[size]; //use Person* to store address of Person/Employee
    //to avoid information slicing for Employee

    Person ann("Ann", 27);
    personPtr[0] = &ann; //personPtr[0] is Person* type

    Employee bob("Bob", 35, 20000);
       //call non-default constructor of Employee
    personPtr[1] = &bob;

    Employee johnDoe; //call default constructor of Employee
    personPtr[2] = &johnDoe;

    for (int i = 0; i < size; i++)
        cout << personPtr[i]->toString() << endl;

    delete[] personPtr;
    personPtr = nullptr;
    return 0;
}
```

7 (10 points) Define recursive function

Define a recursive function to check whether an array of strings is palindrome or not. An array of strings is palindrome if the elements read from left to right and from right to left are the same.

For example, array of strings with values “hi”, “how are you”, “hi” is palindrome, but array of strings wth values “hi”, “how are you” is not palindrome.

Hint: an array is a palindrome if and only the leftmost element equals the rightmost element and the subarray from the second element to the second-to-last element is palindrome. Think what are the initial address and size of that subarray?

Warning: If you do not use recursion, you will not get any point. No repetition statement is allowed in this function.

Answer:

```
bool isPalindrome(string* arr, int size) {
    if (size <= 1)
       return true;

    return arr[0] == arr[size-1] && isPalindrome(arr+1, size-2);
}
```

A complete code to define isPalindrome function and test it in main function.

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

bool isPalindrome(string* arr, int size);
int main() {
    string arr[] = {"hi", "hello", "hi"};
    int size = sizeof(arr) / sizeof(arr[0]);

    cout << boolalpha << isPalindrome(arr, size) << endl;

    string arr2[] = {"hi", "hello", "oh", "hi"};
    int size2 = sizeof(arr2) / sizeof(arr2[0]);

    cout << boolalpha << isPalindrome(arr2, size2) << endl;

    string arr3[] = {"hi"};
    int size3 = sizeof(arr3) / sizeof(arr3[0]);

    cout << boolalpha << isPalindrome(arr3, size3) << endl;
    return 0;
}

bool isPalindrome(string* arr, int size) {
    if (size <= 1)
       return true;

    return arr[0] == arr[size-1] && isPalindrome(arr+1, size-2);
}
```