Answer:

Final Exam S24 Final V3

CSCI 13500: Software Analysis and Design 1

Hunter College, City University of New York

May 22, 2024, 11:30 AM - 1:30 PM, North Building Auditorium

1 (30 points) Answer the following questions.

(1) Given `string animals[] = {"hare", "tortoise", "elephant"}`, what is animals[1].substr(3, 5)?

Answer: animals[1].substr(3, 5) is "toise". Explanation: animals[1] is the second element of array of strings, which is "tortoise". Expression animals[1].substr(3, 5) is the substring from the fourth letter of this string with 5 characters, which is substring "toise".

(2) Given Dog class, declare that class Husky as a subclass of Dog class with public inheritance. Note that Husky is a breed of Dog.

Answer: `class Husky : public Dog`

(3) Write statement to generate a random integer in [-5, 6].

Answer: Answer: `rand() % 12 -5` generate a random int in [-5, 6].

(4) Suppose data member `patterns` of a Hare object is {2, -2, 3, 6}. Method `move` of Hare is as follows.

```
1 void Hare::move() {
2     int index = rand() % patterns.size();
3     int stepsToMove = patterns[index];
4     position += stepsToMove;
5 }
```

Suppose rand() generates a random integer 7, and the value of data member `position` of an object is 5. After calling `move` method, what is the value of `position`?

Answer: 11

(5) Write a unix command to compile `Road.cpp` which has no main function to generate `Road.o`.

Answer: `g++ -c Road.cpp`

(6) What is the value of `3 + 5 / (2 % 3)` in C++?

Answer: 5

(7) Write **header** of a function called longestLen, given an array of string with *size* many elements, return the length of the longest string in the given array.

Answer: `int longestLen(string* strArr, int size);` or

`int longestLen(string strArr[], int size);`

(8) Given `int grades[] = {86, 77, 96, 81, 25};` What is the value of `*(grades + 3)`?

Answer: 81

(9) Declare a string-type pointer p. Apply a dynamically allocated memory to consecutively hold 10 strings, and put its initial address to p.

Answer:

```
string* p = new string[10];
```

or

```
string* p;
p = new string[10];
```

(10) Suppose we have main function defined as follows.

```
int main() {
   double weight = 2.7;
   int m = foo("hello", &weight);
   return 0;
}
```

What is the **header** of function foo?

Answer: int foo(string s, double* p); or

int foo(string, double*);

The first parameter is a string.

&weight is the address of a double variable `weight`. So the second formal parameter of foo should be double*.

(11) What is output for the following code?

```
    int a = -3;
    int* p = &a;
    a += 6;
    cout << *p << endl;
```

Answer: 3

Explanation: after `int* p = &a`, which saves a's address to pointer p, then $*p$ represents the variable whose address in p,

where p is the address of variable a. Note that no two variables can reside in the same address, so *p is an alias of variable a.

a += 6; is the same as $a = a + 6$; so a changes from the initital value -3 to 3. Then *p is 3.

(12) What the output of the following code for `foo(3)`?

```
void foo(int size) {
    for (int numAsts = size; numAsts > 0; numAsts -= 2) {
        for (int i = 0; i < (size - numAsts)/2; i++)
            cout << " ";

        for (int i = 0; i < numAsts; i++)
            cout << "*";

        cout << endl;
    }
}
```

Answer:

```
***
 *
```

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

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

int foo(string input, char ch, char ch2);

int main() {
    cout << foo("acaca", 'a', 'b') << endl;
    return 0;
}
```

```
int foo(string input, char ch, char ch2) {
    int num = 0;
    int num2 = 0;
    for (int i = 0; i < input.size(); i++) {
        if (input[i] == ch)
           num++;
        else if (input[i] == ch2)
               num2++;
    }

    return num - num2;
}
```

Answer: The code returns the number of appearances of 'a' minus the number of appearances of 'b'. The answer is 3.

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

```
vector<int> nums;

for (int i = 1; i < 6; i++)
    nums.push_back(i);

int product = 1;
for (int i = 0; i < nums.size(); i++)
    if (nums[i] % 2 != 0)
       product *= nums[i];

cout << product << endl;
```

Answer: 15

(15) What is the output of the following code? Assume that all necessary libaries are included and namespace is properly used.

```
void foo(vector<int>& v, int index, int value);

int main() {
    vector<int> v = {2, 3, 1};
    foo(v, 1, -1);

    for (int i = 0; i < v.size(); i++)
        cout << v[i] << " ";
    cout << endl;
    return 0;
```

```
}

void foo(vector<int>& v, int index, int value) {
     if (index >= 0 && index < v.size())
        v[index] = value;
}
```

Answer: 2 -1 1

2 (15 points) Answer the following questions.

1. Define function `percentage`, for an given array of characters with its size, return the percentage of characters that are neither 'A' nor 'B' in this array.

   For example, call the function with array with values 'A', 'C', 'B'. The size of array is 3, and there is one occurrence of character that are neither 'A' nor 'B'. The return is 33.3333.

   Warning: C++ is case-sensitive programming language.

   Answer:

```
//Note: you can also use int to replace size_t,
//but size_t is non-negative integer, so it is more appropriate.
double percentage(char arr[], size_t size) {
    int num = 0;
    for (int i = 0; i < size; i++)
        if (!(arr[i] == 'A' || arr[i] == 'B'))
        //if (arr[i] != 'A' && arr[i] != 'B') //also ok, by De Morgan's Law
           num++;

    return 100.0 * num / size;
}
```

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

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

//Define function percentage, for an given array of characters with
//its size, return the percentage of characters that are
//either 'A' or 'B' in this array.

double percentage(char arr[], size_t size);

int main() {
    char arr[] = {'A', 'C', 'B'};
    int size = sizeof(arr) / sizeof(arr[0]);
    cout << percentage(arr, size) << endl;

    return 0;
}

double percentage(char arr[], size_t size) {
    int num = 0;
    for (int i = 0; i < size; i++)
        if (!(arr[i] == 'A' || arr[i] == 'B'))
```

```
23          //if (arr[i] != 'A' && arr[i] != 'B') //also ok, by De Morgan's Law
24              num++;
25
26      return 100.0 * num / size;
27  }
```

2. **Provide** definition of class Date, which contains public integer members

`year`
`month`

The value in `month` should NEVER be > 12 or < 1, where 1 represents January and 12 represents December.

(a) **Define** a NON-member function `subtract_month` which, given Date object `curr` and number of integer representing `num_months`, return Date object representing the date after moving backward `num_months` from `curr`. For simplicity, **assume** that `num_months` is non-negative.

(b) Examples:

i. Suppose `curr` has `year` 2021 and `month` 1, which represents January 2021.

ii. Suppose `num_months` is 23.

iii. Call `subtract_month` function on the above `curr` and `num_months`, return `Date` object with data members `year` 2019 and `month` 2, which represents February 2019.

(c) Hints: one year has 12 months. Note that 23 months equals 1 year and 11 month. What if subtracting the month of current date, say 1, from month 11, is less than 1?

Answer:

```
Date subtract_month(Date curr, int num_months) {
    int newYear = curr.year - num_months / 12;
    int newMonth = curr.month - num_months % 12;
    if (newMonth < 1) {
       newMonth += 12;
       newYear--;
    }

    Date newDate = {newYear, newMonth};
    return newDate;
}
```

A complete code is as follows.

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

class Date {
public:
    int year;
    int month;
};

Date add_month(Date curr, int num_months);
```

```
12
13 int main() {
14     Date curr = {2021, 1};
15     Date before = subtract_month(curr, 23);
16     cout << before.year << " " << before.month << endl;
17     return 0;
18 }
19
20 Date subtract_month(Date curr, int num_months) {
21     int newYear = curr.year - num_months / 12;
22     int newMonth = curr.month - num_months % 12;
23     if (newMonth < 1) {
24        newMonth += 12;
25        newYear--;
26     }
27
28     Date newDate = {newYear, newMonth};
29     return newDate;
30 }
```

3 (10 points) Programming exercise on pointer

1. A two-dimensional coordinate point p is represented by x-coordinate x and y-coordinate y, both of double type.

```
class Coord2D {
public:
    double x;
    double y;
};
```

A Coord2D object represents a point in a 2-dimensional plane, where data members `x` and y are the x- and y-coordinate of that point, respectively. Do not mix point in geometry with pointer in C++.

The x-coordinate of the midpoint between two points is defined as $\frac{\texttt{x of the first point + x of the second point}}{2}$. The y-coordinate is $\frac{\texttt{y of the first point + y of the second point}}{2}$.

For example, given Coord2D object a whose x is 1 and y is 2, Coord2D object b whose x is 6 and y is 3, the midpoint of a and b is a point whose x-coordinate is $\frac{1+6}{2} = 3.5$ and whose y-coordinate is $\frac{2+3}{2} = 2.5$.

Define function `midpoint`, given two **pointers** to Coord2D objects, return a Coord2D object representing the midpoint of the two pointed Coord2D objects.

Answer:

```
Coord2D midpoint(Coord2D* p1, Coord2D* p2) {
    Coord2D point;
    point.x = (p1->x + p2->x) / 2;
    point.y = (p1->y + p2->y) / 2;

    return point;
}
```

2. Write the following statements in main function. No need to include libraries or other parts of main function.

   - Define a as a Coord2D object with x-coordinate 1 and y-coordinate 2.
   - Define b as a Coord2D object with x-coordinate 6 and y-coordinate 3.
   - Find out and print the x- and y-coordinate of the midpoint of a and b.

Answer:

```
Coord2D a = {1, 2};
Coord2D b = {6, 3};

Coord2D point = midpoint(&a, &b);
cout << "x: " << point.x << ", y: " << point.y << endl; //x: 3.5 y: 2.5
```

Optional: a complete code is shown as follows.

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

class Coord2D {
public:
    double x;
    double y;
};

Coord2D midpoint(Coord2D* p1, Coord2D* p2);

int main() {
    Coord2D a = {1, 2};
    Coord2D b = {6, 3};

    Coord2D point = midpoint(&a, &b);
    cout << "x: " << point.x << ", y: " << point.y << endl; //x: 3.5 y: 2.5

    return 0;
}

Coord2D midpoint(Coord2D* p1, Coord2D* p2) {
    Coord2D point;
    point.x = (p1->x + p2->x) / 2;
    point.y = (p1->y + p2->y) / 2;

    return point;
}
```

4 (10 points) Write codes of vector

Define a function called `choose`, for a vector `v` of integers and `left` and `right` as integers, return a vector with all the elements from `v` that are **NOT** in the range of [left, right], in the same order.

For example, given a vector of integers with elements 12, 3, 6, 7, 5 and left 3 and right 6, the return is a vector with elements 12, 7.

Answer:

```
vector<int> choose(vector<int> v, int left, int right) {
    vector<int> result;
    for (int i = 0; i < v.size(); i++) {
        if (v[i] < left || v[i] > right)
          result.push_back(v[i]);
    }

    return result;
}
```

A complete code is shown as follows.

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

vector<int> choose(vector<int> v, int left, int right);
int main() {
    vector<int> v = {12, 3, 6, 5, 7};
    vector<int> result = choose(v, 3, 6);

    for (int i = 0; i < result.size(); i++)
        cout << result[i] << " ";
    cout << endl;

    return 0;
}

vector<int> choose(vector<int> v, int left, int right) {
    vector<int> result;
    for (int i = 0; i < v.size(); i++) {
        if (v[i] < left || v[i] > right)
          result.push_back(v[i]);
    }

    return result;
}
```

5 (15 points) Define class for hexagon shape.

1. Each regular hexagon has 6 same-length sides.

   Assume that Hexagon.hpp is provided where data member **side** is defined as double type. **Your job** is to define the following constructors and methods in Hexagon.cpp. Suppose libraries are included properly in Hexagon.cpp.

2. Define the default constructor, initialize data member **side** to be 1.

   Answer:

```
Hexagon::Hexagon() {
    side = 1;
}
```

3. Define a non-default constructor, which takes formal parameters side, a double type.

   (a) If given parameter side is positive, use it to initialize data member **side**, otherwise, initialize data member **side** by 1.

   Answer:

```
Hexagon::Hexagon(double side) {
    if (side > 0)
       this->side = side;
    else this->side = 1;
}
```

4. Define method **setSide**, if given parameter side is positive, use it to set data member **side**.

   Answer:

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

5. Define method **getPerimeter**, which returns 6 times **side**, the sum of all sides.

   Answer:

```
double Hexagon::getPerimeter() const {
    return 6 * side;
}
```

6. Define method **getArea**, which returns the area, calculated by $\frac{3\sqrt{3}}{2} side^2$, square root can be calculate by `sqrt` function from `cmath` library.

   double sqrt (double x);

Answer:

```
double Hexagon::getArea() const {
    return 3 * sqrt(3) / 2 * side * side;
}
```

Define **HexagonTest.cpp**, do the following:

1. Create a Hexagon object named **hexa** from its default constructor.

Answer:

```
    Hexagon hexa;
```

2. Print out the area of **hexa**.

Answer:

```
    cout << hexa.getArea() << endl;
```

3. Reset the side of **hexa** to be 2.

Answer:

```
    hexa.setSide(2);
```

Answer: Code of Hexagon.hpp is as follows.

```
#ifndef Hexagon_H
#define Hexagon_H

class Hexagon {
public:
    Hexagon();
    Hexagon(double side);
    void setSide(double side);
    double getPerimeter() const;
    double getArea() const;

private:
```

```
    double side;
};
#endif
```

Code of Hexagon.cpp is as follows.

```
#include "Hexagon.hpp"
#include <cmath>

Hexagon::Hexagon() {
    side = 1;
}

Hexagon::Hexagon(double side) {
    if (side > 0)
       this->side = side;
    else this->side = 1;
}

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

double Hexagon::getPerimeter() const {
    return 6 * side;
}

double Hexagon::getArea() const {
    return 3 * sqrt(3) / 2 * side * side;
}
```

Code of HexagonTest.cpp is as follows.

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

//Put Hexagon.hpp, Hexagon.cpp, and HexagonTest.cpp in the same folder
//run the following commands
//    g++ Hexagon.cpp HexagonTest.cpp
//    ./a.out

//If Hexagon.cpp and HexagonTest.cpp are the ONLY cpp files in the current folder,
//we can also replace
//    g++ Hexagon.cpp HexagonTest.cpp
//by the following command, where *.cpp means all the cpp files
```

```
//    g++ *.cpp

int main() {
    //Create a Hexagon object named hexa from its default constructor.
    Hexagon hexa;

    //Print out the area of hexa.
    cout << hexa.getArea() << endl;

    //Reset the side of hexa to be 2.
    hexa.setSide(2);
    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;
};
```

1. Declare Student as a subclass of Person. Each student is a person, with additional data member **courses**, a vector of strings, to describe courses.
2. **Your job:** override `toString` method in Student,
   (a) Invoke `toString` method from super class to get a string representing name and age.
   (b) Concatenate the above string with a string representing the courses information. For example, if the courses are "CS 127", "CS 135", the string to represent courses information can be "CS 127, CS 135, ". We add a ',' after each course since the name of a course contains spaces.
   (c) Return the concatenated string.

Answer:

```
std::string Student::toString() const { //override cannot be here
    std::string str = Person::toString();
    str += "courses:\n"; //\n means new line
    for (int i = 0; i < courses.size(); i++)
        str += courses[i] + ", ";

    return str;
}
```

3. Define method **getNumCourses** to return the number of courses of a student.

Answer:

```
int Student::getNumCourses() const {
   return courses.size();
}
```

Answer: (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 Student.hpp

Answer:

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

//(1) We normally do not use standard namespace in a header file,
//    ended by .hpp; otherwise, all the files included
//    the header file will have to use standard namespace as well.
//    This is like, English is the most popular language,
//    but we do not set it as default language in every setting.
//(2) Without using namespace std, we use std::vector instead of
//    vector, use std::string instead of string.
class Student : public Person {
public:
    Student(std::string name, int age);
    int getNumCourses() const;
    std::string toString() const override; //override can be omitted, and can only be
        used in c++11 or above
    void addCourse(std::string course);
```

```
private:
    std::vector<std::string> courses;
};
#endif
```

code of Student.cpp

```
#include "Student.hpp"

Student::Student(std::string name, int age) : Person(name, age) {
    //in the very beginning, no course is selected,
    //that is, data member courses is an empty vector.
    //Nothing needs to do in this constructor
}

std::string Student::toString() const { //override cannot be here
    std::string str = Person::toString();
    str += "courses:\n"; //\n means new line
    for (int i = 0; i < courses.size(); i++)
        str += courses[i] + ", ";

    return str;
}

//add a course
void Student::addCourse(std::string course) {
    if (course != "")
       courses.push_back(course);
}

int Student::getNumCourses() const {
   return courses.size();
}
```

code of StudentTest.cpp

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

//Put Person.hpp, Person.cpp, Student.hpp, Student.cpp, and StudentTest.cpp in the
    same folder
//run the following commands
//    g++ -std=c++11 *.cpp
//or
//    g++ -std=c++11 Person.cpp Student.cpp StudentTest.cpp
//If the above command runs without complaints, run
```

```
//    ./a.out

//sample output:
//name: Ann
//age: 18
//courses:
//CS 127, CS 135,

int main() {
    Student ann("Ann", 18);
    ann.addCourse("CS 127");
    ann.addCourse("CS 135");
    cout << ann.toString() << endl;
    return 0;
}
```

7 (10 points) Define recursive function

Define a recursive function, for an given array of integers, return the minimum integer. Note that the size of an array in C++ cannot be zero.

For example, suppose the array of integers has elements 2, 3, 1, the return is 1.

Hint: what if the array has only one element? When the array has more than one element, how to find out the minimum element in a subarray?

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

Answer:

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

int min(int* arr, int size);

int main() {
    int arr[] = {2, 3, 1};
    int size = sizeof(arr) / sizeof(arr[0]);
    cout << min(arr, size) << endl;
    return 0;
}

int min(int* arr, int size) {
    if (size == 1)
       return arr[0];

    //size must > 1, otherwise we would return in the above if-statement
    int value = min(arr+1, size-1);
    //min(arr+1, size-1) returns the minimum element in
    //the subarray from arr[1] to the last element of arr.
    //The initial address of this subarray is arr+1,
    //the number of elements in this subarray is size-1.

    if (value < arr[0])
       return value;
    else return arr[0]; //else can be omitted
}
```