Answer:

Final Exam F25 Final V2

CSCI 13500: Software Analysis and Design 1

Hunter College, City University of New York

December 18, 2025, 9:00 AM - 11:00 AM, N 118

1 (30 points) Answer the following questions.

(1) Given `std::string colors[] = {"green", "light blue", "navy blue"}`, what is colors[2].substr(1)?

Answer: colors[2].substr(1) is "avy blue". Explanation: colors[2] is the third element of array of strings, which is "navy blue". Expression colors[2].substr(1) is the substring from the second letter – index 1 – of this string to the end, which is substring "avy blue".

(2) Given a declaration `std::vector<int> v(3, 10); v.pop_back();`, what is the value of `v.size()`?

Answer: 2

explanation: `vector<int> v(3, 10);` creates a vector of integers with 3 elements, each element has value 10. After pop the last elements from it, v has one fewer element. Hence, there are 2 elements in vector v.

(3) How to generate a random integer in [120, 150]?

Answer: `rand() % 31 + 120`

`rand() % 31` generates a random int in [0, 30].

`rand() % 31 + 120` generates a random int in [120, 150].

(4) Given `std::string str = std::to_string(265) + "-02";`, where `to_string` converts an integer to a string. What is the value for `str`?

Answer: `"265-02"`

(5) What is the value of `3 + (6 + 1) % 5` in C++?

Answer: 5

Explanation: expression in parentheses runs first. Run 3 + 7 % 5.

7 % 5 is like to divide 7 pens among 5 persons, each person get 1 pen, 2 pens left.

Add 3 to the result of 7 % 5, so the sum is 5.

(6) Write **header** of a function called pop, given an array *arr* of **int** type with *size* many elements, remove the first element from the array. Return type of the function is `void`.

Answer: `void pop(int* arr, int& size);` or

`void pop(int arr[], int& size);`

(7) Declare class Coord3D as follows.

```
class Coord3D {
public:
  double x;
  double y;
  double z;
};
```

Declare a Coord3D object `point` and initialize its x, y, z to be 7, 8, 9, respectively.

Answer:

```
Coord3D point = {7, 8, 9};
```

or

```
Coord3D point{7, 8, 9};
```

or

```
Coord3D point;
point.x = 7;
point.y = 8;
point.z = 9;
```

(8) Given `int grades[] = {100, 75, 78, 96};` What is the value of `*(grades+2) - 2`?

Answer: 76

(9) Given the following code segment, finish the TODO part.

```
int main() {
  std::cout << "Enter size of an array: ";
  int size;
  std::cin >> size;
  while (size <= 0) {
    std::cout << "Re-enter, the size is not positive: ";
    std::cin >> size;
  }

  //TODO: write a statement to declare and initialize a pointer arr to
  //a dynamically allocated array with size elements of string type.
```

```
  //WRITE YOUR ANSWER IN THE FOLLOWING BOX.


  //omit other code ...
  return 0;
}
```

Answer: `std::string* arr = new std::string[size];` or `string* arr = new string[size];`

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

```
int main() {
  std::string s1 = "work";
  std::string s2 = "success";
  //In foo, if s1 > s2, exchange the values of s1 and s2.
  //The return type is void.
  foo(&s1, &s2);

  std::cout << s1 << " " << s2 << " " << std::endl;
  //expected output: success work
  return 0;
}
```

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

Answer: `void foo(std::string* p1, std::string* p2);` or `void foo(string* p1, string* p2);`

A complete code is shown as follows.

```
#include <iostream>
#include <string>

void foo(std::string* p1, std::string* p2);

int main() {
  std::string s1 = "work";
  std::string s2 = "success";
  //In foo, if s1 > s2, exchange the values of s1 and s2.
  //The return type is void.
  foo(&s1, &s2);

  std::cout << s1 << " " << s2 << " " << std::endl;
  //expected output: success work
  return 0;
}
```

```
17
18 void foo(std::string* p1, std::string* p2) {
19   if (*p1 > *p2) {
20     std::swap(*p1, *p2);
21   }
22 }
```

(11) What is output calling `foo` with an array with elements -5, 0, 2, -6, 7 and its corresponding size?

```
1 int foo(int* arr, int size) {
2   int i = 0;
3   while (i < size && arr[i] <= 0) {
4     i++;
5   }
6
7   return i;
8 }
```

Answer: 2

Explanation: the above loop is to return the index of the number of **first** positive element in the array; if every element of the array is non-positive, then return the number of elements of the array.

In this example, the first two elements are non-positive but not the third, which is indexed at 2. Note that the index of the first element is 0.

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

```
1 std::vector<int> nums = {3, 1, -1, 2, -3};
2
3 int result = 0;
4 for (int i = 0; i < nums.size(); i++) {
5   if (nums[i] > 0) {
6     result += nums[i];
7   }
8 }
9
10 std::cout << result << std::endl;
```

Answer: 6

Print out the sum of all positive integers in the vector.

(13) Given arr with values 17, 3, 5, -6 with size 4, what will be the value of arr after calling foo on arr and size?

```
1 void foo(int arr[], int size) {
2   int i = 0;
3   int j = size-1;
```

```
  while (i < j) {
    std::swap(arr[i], arr[j]);
    i++;
    j--;
  }
}
```

Answer: -6 5 3 17

A complete code is as follows.

```
#include <iostream>
#include <string>

void foo(int arr[], int size);

int main() {
  int arr[] = {17, 3, 5, -6};
  int size = sizeof(arr) / sizeof(arr[0]);
  foo(arr, size);

  for (int i = 0; i < size; i++) {
    std::cout << arr[i] << " ";
  }
  std::cout << std::endl;

  return 0;
}

void foo(int arr[], int size) {
  int i = 0;
  int j = size-1;
  while (i < j) {
    std::swap(arr[i], arr[j]);
    i++;
    j--;
  }
}
```

(14) What the output of the following code? For simplicity, we omit libraries.

```
int main() {
  int size = 3;
  for (int row = 0; row < size; row++) {
    for (int col = 0; col < size; col++) {
      if (row >= col) {
        std::cout << "*";
```

```
      }
      else {
        std::cout << "-";
      }
    }
    std::cout << std::endl;
  }

  return 0;
}
```

Answer:

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

A complete code is as follows.

```
//Purpose: print bottom left shape
//*--
//**-
//***
#include <iostream>
#include <string>

int main() {
  int size = 3;
  for (int row = 0; row < size; row++) {
    for (int col = 0; col < size; col++) {
      if (row >= col) {
        std::cout << "*";
      }
      else {
        std::cout << "-";
      }
    }
    std::cout << std::endl;
  }

  return 0;
}
```

(15) What is the output of the following code? Assume that libraries are set up.

```
std::string foo(std::vector<std::string> v, char ch);
```

```
2
3  int main() {
4    std::vector<std::string> v = {"apple", "", "blue", "corn", "berry", "ant"};
5    std::cout << foo(v, 'b') << std::endl;
6    return 0;
7  }
8
9  std::string foo(std::vector<std::string> v, char ch) {
10   std::string result;
11   for (int i = 0; i < v.size(); i++) {
12     if (v[i] != "" && v[i][0] == ch) {
13       result += v[i] + ",";
14     }
15   }
16
17   return result;
18 }
```

Answer: blue,berry,

Explanation: concatenate strings starting with letter 'b'.

2 (15 points) Answer the following questions.

(1) Define a function, `getInitial`, for a string in the format of last name last, followed by comma symbol(,), followed by first name, return a string representing the initial.

For example, given a string with value `"Washington,George"`, the returned string is `"GW"`.

Hint: extract the last name and the first name. Instantiate an empty string. Concatenate the empty string with the first letter of the first name, then the first letter of the last name. You might use the following methods from string class.

`size_t find (char c, size_t pos = 0) const;`

Searches the string for the first occurrence of the sequence specified by its arguments.

`string substr (size_t pos = 0, size_t len = npos) const;`

Returns a newly constructed string object with its value initialized to a copy of a substring of this object.

Answer:

```
1  //Purpose: for a name in the format of the last name first,
2  //followed by a comma symbol (,), followed by the first name.
3  //Return the initial of a name by concatenating the first letter of
4  //the first name and the first letter of the last name.
5  #include <iostream>
6  #include <string>
```

```
7
8  std::string getInitial(std::string name);
9
10 int main() {
11   std::cout << getInitial("Washington,George") << std::endl; //print "GW" without
       quotes
12   return 0;
13 }
14
15 std::string getInitial(std::string name) {
16   size_t commaIdx = name.find(',');
17   std::string lastName = name.substr(0, commaIdx);
18   std::string firstName = name.substr(commaIdx+1);
19
20   std::string initial;
21   initial += firstName[0];
22   initial += lastName[0];
23
24   return initial;
25 }
```

(2) Write a function `pointerToLastOdd` that returns a **pointer** to the **last** occurrence (if there are more than one occurrence) of the odd integer in an array of int type with *size* many elements.

If size is 0 or there is no odd integer, return nullptr.

For example, given an array with elements 5, 4, 3, 2, 1, the return of the function is a pointer to element 1. Given an array with elements 2, 6, return of the function is nullptr.

Answer:

```
1  //Purpose: return the pointer to the last odd number in the array
2  #include <iostream>
3  #include <string>
4
5  int* pointerToLastOdd(int arr[], int size);
6
7  int main() {
8    int arr[] = {5, 4, 3, 2, 1};
9    int size = sizeof(arr) / sizeof(arr[0]);
10
11   int* ptr = pointerToLastOdd(arr, size);
12   std::cout << ptr << std::endl; //0x16b7f6e50,
13   //the actual address changes from different systems and different running time
14   std::cout << ptr - arr << std::endl; //print 4,
15   //ptr - arr is the index of the element residing at ptr
16
17   int arr2[] = {0, 2, 4, 6}; //no odd number
```

```
  int size2 = sizeof(arr2) / sizeof(arr2[0]);

  int* ptr2 = pointerToLastOdd(arr2, size2);
  std::cout << ptr2 << std::endl; //nullptr is printed as 0x0

  return 0;
}

int* pointerToLastOdd(int arr[], int size) {
  for (int index = size-1; index >= 0; index--) {
    if (arr[index] % 2 != 0) {
       return arr + index;
    }
  }

  return nullptr;
}
```

3 (10 points) Programming exercise on class

Define class `Length` to represent length in feet and inches. It is reasonable to define it to have two integer fields:

`foot` for the number of feet, and

`inch` for the number of inches. Note that a foot has 12 inches, so we need to make sure that inch is in [0, 11].

Define non-member function `minusInches`, given Length object len and integer parameter numInches, the function should create and return a Length object that is the result of subtracting `len` from `numInches`. Example:

Suppose `len` is {2, 7} and `numInches` is 10. Then the return of `minusInches` function on the above parameters is {1, 9}.

Reason: 2 feet 7 inches is 2 * 12 + 7 = 31 inches. Subtract 10 from 31 is 21 inches, which equals 1 foot and 9 inches, where 1 is obtained by 21 divided by 12 and 9 is the remainder of 21 divided by 12.

For simplicity, assume that the total number of inches from `len` is no smaller than `numInches`.

Answer:

```
#include <iostream>
#include <string>

class Length {
public:
  int foot;
  int inch;
};

Length minusInches(Length len, int inchesToAdd);
```

```
11
12 int main() {
13   Length len{2, 7};
14   Length result = minusInches(len, 10);
15
16   std::cout << result.foot << " " << result.inch << std::endl; //print 1 9
17   return 0;
18 }
19
20 Length minusInches(Length len, int numInches) {
21   int totalInches = len.foot * 12 + len.inch - numInches;
22
23   return {totalInches / 12, totalInches % 12};
24 }
```

4 (10 points) Write codes of vector

Define a function called `longStrings`, for a vector `v` of strings and an int `n`, search the `v`, find out the strings whose lengths are longer than `n`, put them to a vector of strings, in the same order. Return that vector.

Given a vector of strings with elements `"hello", "", "hey", "hi"` and 2, the return is vector with elements `"hello", "hey"`.

Answer:

```
1  std::vector<std::string> longString(std::vector<std::string> v, int n) {
2    std::vector<std::string> result;
3    for (int i = 0; i < v.size(); i++) {
4      if (v[i].length() > n) {
5        result.push_back(v[i]);
6      }
7    }
8
9    return result;
10 }
```

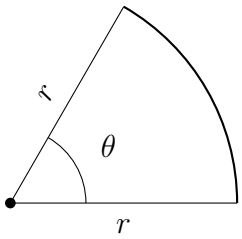
A complete code is shown as follows.

```
1  //Define a function called longStrings, for a vector v of strings and an int n, search
       the v, find out the strings whose lengths are longer than n, put them to a vector of
        strings, in the same order. Return that vector.
2
3  //Given a vector of strings with elements "hello", "", "hey", "hi" and integer 2, the
       return is vector with elements "hello", "hey".
4
5  #include <iostream>
6  #include <string>
```

```cpp
#include <vector>

std::vector<std::string> longString(std::vector<std::string> v, int n);

int main() {
  std::vector<std::string> vec = {"hello", "", "hey", "hi"};
  std::vector<std::string> result = longString(vec, 2);

  for (int i = 0; i < result.size(); i++) {
    std::cout << "\"" << result[i] << "\"" << " ";
    //print "hello" "hey"
  }

  return 0;
}

std::vector<std::string> longString(std::vector<std::string> v, int n) {
  std::vector<std::string> result;
  for (int i = 0; i < v.size(); i++) {
    if (v[i].length() > n) {
      result.push_back(v[i]);
    }
  }

  return result;
}
```

5 (15 points) Define class.

1. Define an Arc as a portion of a circle.



   (a) radius of the circle **r**

   (b) the **angle** between the two edges, represented as θ in the above figure, in **degrees**.

2. **Assume that Arc.hpp is provided** where data members **r** and **angle** are declared as **double** types. Angle is represented in **degrees**. **Your job** is to define **Arc.cpp** with the following requirement.

3. Define the default constructor, set data members **r** to be 1 and **angle** to be 45.

Answer:

```
Arc::Arc() {
  r = 1;
  angle = 45;
}
```

4. Define a non-default constructor, which takes formal parameters r and angle, both are **double** types.

   (a) If r is positive and angle is strictly larger than 0 and angle is strictly smaller than 360, set data member **r** by given parameter r and set data member **angle** by given parameter angle.

   (b) otherwise, set data members **r** to be 1 and **angle** to be 45.

Answer:

```
Arc::Arc(double r, double angle) {
  if (r > 0 && angle > 0 && angle < 360) {
    this->r = r;
    this->angle = angle;
  }
  else {
    this->r = 1;
    this->angle = 45;
  }
}
```

5. Define method **getPerimeter**, return the value of $\frac{angle}{360} * 2\pi r + 2r$. Note that r and *angle* are data members. Note that π is represented by `M_PI` in C++.

Answer:

```
double Arc::getPerimeter() const {
  return angle / 360 * 2 * M_PI * r + 2 * r;
}
```

6. Define method **setAngle**, if given parameter angle is positive and is smaller than 360, reset data member **angle** by given parameter angle.

Answer:

```
void Arc::setAngle(double angle) {
  if (angle > 0 && angle < 360) {
    this->angle = angle;
  }
}
```

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

7. Create an Arc object named **shape** from its non-default constructor with the radius of the arc as 3 and the angle as 45.

Answer:

```
Arc shape(3, 45);
```

8. Reset the angle to be 90.

Answer:

```
shape.setAngle(90);
```

9. Find out and print the perimeter of shape.

Answer:

```
std::cout << "perimeter: " << shape.getPerimeter() << std::endl;
```

Answer: A complete code is as follows.

code of Arc.hpp

```
#ifndef Arc_H
#define Arc_H
class Arc {
public:
  Arc();
  Arc(double r, double angle); //degrees
  double getArea() const;
  double getPerimeter() const;
  void setRadius(double r);
  void setAngle(double angle);

private:
  double r; //radius of the arc
  double angle; //angle of the arc, represented in degrees
};
#endif
```

Code of Arc.cpp

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

Arc::Arc() {
  r = 1;
  angle = 45;
}
```

```
8
9  Arc::Arc(double r, double angle) {
10   if (r > 0 && angle > 0 && angle < 360) {
11     this->r = r;
12     this->angle = angle;
13   }
14   else {
15     this->r = 1;
16     this->angle = angle;
17   }
18 }
19
20 void Arc::setRadius(double r) {
21   if (r > 0) {
22     this->r = r;
23   }
24 }
25
26 void Arc::setAngle(double angle) {
27   if (angle > 0 && angle < 360) {
28     this->angle = angle;
29   }
30 }
31
32 double Arc::getArea() const {
33   return angle / 360 * M_PI * r * r;
34 }
35
36 double Arc::getPerimeter() const {
37   return angle / 360 * 2 * M_PI * r + 2 * r;
38 }
```

code of TestArc.cpp

```
1  #include <iostream>
2  #include <string>
3  #include "Arc.hpp"
4
5  //Run code using
6  //g++ TestArc.cpp Arc.cpp
7  //sample output:
8  //area: 7.06858
9  //perimeter: 10.7124
10 int main() {
11   Arc shape(3, 45);
12   shape.setAngle(90);
13   std::cout << "area: " << shape.getArea() << std::endl;
```

```
  std::cout << "perimeter: " << shape.getPerimeter() << std::endl;

  return 0;
}
```

6 (10 points) function on vectors

Define a function called `mixNums`, given a vector of **double** numbers `vec`, do the following:

Check whether all the elements in `vec` has at least a postive number **and** at least a negative number.

For example, if `vec` has values 1.6, -2.7, 0, the returned is true since there is one positive number 1.6 and one negative number -2.7.

If `vec` has values 1, 2.3, 3.3, the returned is false since there is no negative numbers.

Answer: function `mixNums` is defined as follows.

```
bool mixNums(std::vector<double> vec) {
  int numPos = 0; //number of positive elements
  int numNeg = 0; //number of negative elements
  for (int i = 0; i < vec.size(); i++) {
    if (vec[i] > 0) {
      numPos++;
    }
    else if (vec[i] < 0) {
      numNeg++;
    }
  }

  return numPos > 0 && numNeg > 0;
}
```

A complete code is as follows.

```
#include <iostream>
#include <string>
#include <vector>

bool mixNums(std::vector<double> vec);

int main() {
  std::vector<double> vec = {1.6, -2.7, 0};
  std::cout << std::boolalpha << mixNums(vec) << std::endl; //true

  std::vector<double> vec2 = {1, 2.3, 3.3};
  std::cout << std::boolalpha << mixNums(vec2) << std::endl; //false

  return 0;
}
```

```
16
17 bool mixNums(std::vector<double> vec) {
18   int numPos = 0; //number of positive elements
19   int numNeg = 0; //number of negative elements
20   for (int i = 0; i < vec.size(); i++) {
21     if (vec[i] > 0) {
22       numPos++;
23     }
24     else if (vec[i] < 0) {
25       numNeg++;
26     }
27   }
28
29   return numPos > 0 && numNeg > 0;
30 }
```

7 (10 points) Recursive Function

Define a recursive function `minLen`, given an array of **string** with size many elements, return the minimum length of all strings.

For example, for array with elements `"hello", "hi", "how"`, the return is 2, since the shortest string "hi" has 2 characters.

Hint: you should set the return type of `minLen` as `size_t`, which is similar to `int` but represents only non-negative ingegers.

You may use `std::min` function from `algorithm` library, which takes two parameters of the same types and return the minimum. Or, you may use if-else statement to replace `std::min`.

Warning: If you do not use recursion, you will not get any point.
No repetition statement, global or static variables are allowed in this function.
Use array, not vector.

Answer: Code of function is as follows.

```
1  size_t minLen(std::string arr[], int size) {
2    if (size == 1) {
3      return arr[0].length();
4    }
5
6    //minLen(arr+1, size-1) is the shortest string in
7    //the subarray of arr without arr[0]
8    return std::min(arr[0].length(), minLen(arr+1, size-1));
9    //Warning: std::min cannot compare an int and a size_t,
10   //the types need to be the same.
11   //length method of a string returns size_t type.
12
13   //the above return statement can be rewritten as
14 //  size_t remMinLen = minLen(arr+1, size-1);
```

```
//  if (arr[0].length() < remMinLen) {
//    return arr[0].length();
//  }
//  else {
//    return remMinLen;
//  }
}
```

A complete code is as follows.

```
//Purpose: define a recursive function to return the minimum length of all strings in a
    given array.
#include <iostream>
#include <string>
#include <algorithm> //std::min

size_t minLen(std::string arr[], int size);

int main() {
  std::string arr[] = {"hello", "hi", "hey"};
  int size = sizeof(arr) / sizeof(arr[0]);

  std::cout << minLen(arr, size) << std::endl; //2,
  //the length of the shortest string "hi"
  return 0;
}

size_t minLen(std::string arr[], int size) {
  if (size == 1) {
    return arr[0].length();
  }

  //minLen(arr+1, size-1) is the shortest string in
  //the subarray of arr without arr[0]
  return std::min(arr[0].length(), minLen(arr+1, size-1));
  //Warning: std::min cannot compare an int and a size_t,
  //the types need to be the same.
  //length method of a string returns size_t type.

  //the above return statement can be rewritten as
//  size_t remMinLen = minLen(arr+1, size-1);
//  if (arr[0].length() < remMinLen) {
//    return arr[0].length();
//  }
//  else {
//    return remMinLen;
//  }
```

```
37 }
```