

Answer:

|      |       |
|------|-------|
| Row: | SEAT: |
|      |       |

FINAL EXAM F23 V3  
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", "nice to meet you"}`, what is `greetings[1].size()`?

**Answer:** `greetings[1].size()` is **2**. Explanation: `greetings[1]` is the second element of array of strings, which is "Hi". Expression `greetings[1].size()` is the number of letters of string `greetings[1]`, which is 2.

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

**Answer:** `class Teacher : public Employee`

- (3) Write code to generate a random int between 70 and 100, where both ends are included. No library is needed.

**Answer:** Answer: Use `rand() % 31 + 70` to generate a random int in `[70, 100]`.

Explanation:

- There are  $100 - 70 + 1 = 31$  integers from 70 to 100.
- `rand() % 31` generates a random integer from 0 to 30.
- `rand() % 31 + 70` generates a random integer from 70 to 100.

- (4) Given `string greeting = "wow"`; What is the value for `greeting.substr(0,2)`?

**Answer:** the answer is string **"wo"**.

Explanation: the first parameter is the starting index of the substring, number 0 is the index of the first letter.

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

So, `greeting.substr(0, 2)` means to extract a substring from string `greeting` whose value is "wow". Starting from the letter indexed at 0 (the first letter), get 2 more letters, the result is **"wo"**.

- (5) Write a command to compile and link `TestField.cpp` and `Field.cpp` to generate a runnable file **prog**.

**Answer:** `g++ -o prog TestField.cpp Field.cpp`

- (6) What is the value of `1 / 3 * 6` in C++?

**Answer:** **0**

Explanation: division operator `/` and multiplication operator `*` have the same precedence, and both are running from left to right. So operator `/` runs first in `1/3*6`, integer division of `1 / 3` is zero, after it is multiplied with 6, the result is still zero.

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

**Answer:** Use one the following:

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

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

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

**Answer:** **6**

Explanation: dereference operator `*` has higher precedence than `+`. So we run `*arr` first.

- `arr` is the address of `arr[0]`, so `*arr` is the element residing in that address, that is, `arr[0]`, the first element of array `arr`.
- `*arr + 2` adds 2 to `arr[0]`. So the result is 6.

- (9) Declare and initialize a two-dimensional strings array called **synonyms** with three rows, each row with two columns. The first row is “hurry”, “quick”, the second row is “smile”, “giggle”, the third row is “small”, “tiny”.

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

```
string synonyms[3][2] = { {"hurry", "quick"}, {"smile", "giggle"}, {"small", "tiny"} };
```

or

```
string synonyms[][2] = { {"hurry", "quick"}, {"smile", "giggle"}, {"small", "tiny"} };
```

- (10) What is output for the following code?

```
1 vector<int> nums;
2 for (int i = 12; i >= 0; i--)
3     nums.push_back(i);
4
5 for (int i = 0; i < nums.size(); i++)
6     if (i % 3 == 0)
7         cout << nums[i] << " ";
8
9 cout << endl;
```

**Answer:** 12 9 6 3 0

Explanation: first put 12, ..., 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 3. So the print out is

12 9 6 3 0

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

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int result = 0;
6     for (int num = 3; num < 11; num += 5)
7         result += num;
8
9     cout << result << endl;
10    return 0;
11 }
```

**Answer:**  $3 + 8 = 11$

Explanation: num starts from 3, as long as it is smaller than 11, add it to variable result. After each round, increased num by 5.

The numbers are eligible to be added are 3 and 8. So result is 11.

You can also think in tabular format.

int result = 0;

| num | $num < 11?$ | result += num                                  | num += 5                                     |
|-----|-------------|--|--|
| 3   | yes         | result is increased by 3, changes from 0 to 3  | num is increased by 5, and num changes to 8  |
| 8   | yes         | result is increased by 8, changes from 3 to 11 | num is increased by 5, and num changes to 13 |
| 13  | no, stop    |  |  |

(12) What is output for the following code?

```
1  int a = 2;
2  int* p = &a;
3  *p += 7;
4  cout << a << endl;
```

**Answer:** 9

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 += 7;` is the same as `a += 7;` so `a` changes from the initial value 2 to be 9.

(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 *= 7;
13     else a += 2;
14 }
```

**Answer:** 7

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 multiplied by 7.

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 7.

(14) What the output when input is 90.1?

```
1 cout << "Enter a number: ";
2 double num;
3 cin >> num;
4 switch ((int)num / 10) {
5     case 10:
6     case 9: cout << "excellent" << endl;
7         break;
8     case 8: cout << "good" << endl;
9         break;
10    case 7: cout << "ok" << endl;
11        break;
12    case 6: cout << "work hard" << endl;
13        break;
14    default: cout << "do not give up" << endl;
15 }
```

**Answer:** excellent

Explanation: (int)num changes a number to an int by truncating the decimal numbers. When num is 90.1, (int)num is 90. The result of integer division of 90 / 10 is 9, it is like to divide 90 pens among 10 kids, each kid gets 9 pens.

Then find out the label matching 9 and run the statements until statement break; or the end of switch statement is reached.

(15) What is the panel like when press left in game 1024? The empty cell is 0. No need to put 1 in an empty random cell.

|   |   |   |
|---|---|---|
| 1 | 1 | 1 |
|   | 1 | 1 |
| 1 |   | 1 |

**Answer:** After merging, the result looks like.

|   |   |  |
|---|---|--|
| 2 | 1 |  |
| 2 |   |  |
| 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.

|   |   |   |
|---|---|---|
| 2 | 1 | 0 |
| 2 | 0 | 0 |
| 2 | 0 | 0 |

## 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 negative.  
For example, call the function with array with values -1, 0, -2, 0, 6, the size of array is 5, then the return is 2.

Answer:

```
1 int getNumNegatives(int arr[], int size) {
2     int numNegatives = 0;
3     for (int i = 0; i < size; i++)
4         if (arr[i] < 0)
5             numNegatives++;
6
7     return numNegatives;
8 }
```

A complete code is as follows.

```
1 #include <iostream>
2 using namespace std;
3
4 int getNumNegatives(int arr[], int size);
5 //int getNumNegatives(int* arr, int size); //is also fine
6
7 int main() {
8     int arr[] = {-1, 0, -2, 0, 6};
9     int size = sizeof(arr) / sizeof(arr[0]);
10
11     cout << getNumNegatives(arr, size) << endl; //print 2
12     return 0;
13 }
14
15 int getNumNegatives(int arr[], int size) {
16     int numNegatives = 0;
17     for (int i = 0; i < size; i++)
18         if (arr[i] < 0)
19             numNegatives++;
20
21     return numNegatives;
22 }
```

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

Answer:

```
1 void sortByLenRev(string* a, string* b) {
2     if ((*a).size() < (*b).size()) //ok
3     //if (a->size() < b->size()) //use a->size() instead of (*a).size(), also ok
4         swap(*a, *b);
5 }
```

```
1 #include <iostream>
2 using namespace std;
3
4 void sortByLenRev(string* a, string* b);
5
6 int main() {
7     string s1 = "hi";
8     string s2 = "hello";
9
10    sortByLenRev(&s1, &s2);
11
12    cout << "s1 = " << s1 << ", s2 = " << s2 << endl;
13    //print s1 = hello, s2 = hi
14    return 0;
15 }
16
17 void sortByLenRev(string* a, string* b) {
18     if ((*a).size() < (*b).size()) //ok
19     //if (a->size() < b->size()) //use a->size() instead of (*a).size(), also ok
20         swap(*a, *b);
21 }
```

### 3 (20 points) Programming exercises

- (1) Define a function, for a given string, if it contains at least a digit **and** a special symbol in %, ?, or !, return true, otherwise, return false.

For example, for string "abc", the return is false. For string "?!", the return is false. For "2!", the return is true. For "?2", the return is true.

Hint: you might use the following function.

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

You can count the number of occurrences of digits and number of occurrences of special symbols.

**Answer:**

```
1 //Define a function, for a given string,
2 //if it contains at least a digit and a special symbol in
3 //%, ?, or !, return true, otherwise, return false.
4
5 //For example, for string "abc", the return is false.
6 //For string "?!", the return is false.
7 //For "2!", the return is true.
8 //For "?2", the return is true.
9
10 #include <iostream>
11 using namespace std;
12
13 bool hasDigitSpecialSymbol(string s);
14
15 int main() {
16     string strs[] = {"abc", "?!", "2!", "?2"};
17     int size = sizeof(strs) / sizeof(strs[0]);
18
19     for (int i = 0; i < size; i++)
20         cout << boolalpha << hasDigitSpecialSymbol(strs[i]) << endl;
21     //print false false true true, one in a row
22
23     return 0;
24 }
25
26 bool hasDigitSpecialSymbol(string s) {
27     int numDigits = 0;
28     int numSpecialSymbols = 0;
29     for (int i = 0; i < s.size(); i++)
30         if (isdigit(s[i])) //can write as if (s[i] >= '0' && s[i] <= '9')
31             numDigits++;
32         else if (s[i] == '%' || s[i] == '?' || s[i] == '!')
33             numSpecialSymbols++;
34
35     return numDigits > 0 && numSpecialSymbols > 0;
36 }
```



(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 4 columns.

**Answer:**

```
1 int numRows = 3;
2 int numCols = 4;
3 panel = new int*[numRows];
4 for (int row = 0; row < numRows; row++)
5     panel[row] = new int[numCols];
6 }
```

- (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  $0 \leq \text{row} < 3$  and  $0 \leq \text{col} < 4$ .

**Answer:**

```
1 for (int row = 0; row < numRows; row++) //numRows can be written as 3
2     for (int col = 0; col < numCols; col++) //numCols can be written as 4
3         panel[row][col] = row - col;
```

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

**Answer:**

```
1 for (int row = 0; row < numRows; row++) { //numRows can be written as 3
2     delete[] panel[row];
3     panel[row] = nullptr;
4 }
5
6 delete[] panel;
7 panel = nullptr;
```

A complete code is as follows.

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int** panel;
6
7     int numRows = 3;
8     int numCols = 4;
9     panel = new int*[numRows];
10    for (int row = 0; row < numRows; row++)
11        panel[row] = new int[numCols];
12
13    //set panel[row][col] by row - col
```

```
14     for (int row = 0; row < numRows; row++) //numRows can be written as 3
15         for (int col = 0; col < numCols; col++) //numCols can be written as 4
16             panel[row][col] = row - col;
17
18     //Release the dynamically allocated memory and avoid dangling pointer problem.
19     for (int row = 0; row < numRows; row++) { //numRows can be written as 3
20         delete[] panel[row];
21         panel[row] = nullptr;
22     }
23
24     delete[] panel;
25     panel = nullptr;
26
27     return 0;
28 }
```

## 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 value "ab", "abcd".

Answer:

```
1 vector<string> evenLen(vector<string> vec) {
2     vector<string> result;
3
4     for (int i = 0; i < vec.size(); i++)
5         if (vec[i].size() % 2 == 0) //vec[i].size() can be replaced by vec[i].length()
6             result.push_back(vec[i]);
7
8     return result;
9 }
```

A complete code is shown as follows.

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 vector<string> evenLen(vector<string> vec);
6
7 int main() {
8     vector<string> vec = {"ab", "ccd", "abcd"};
9
10    vector<string> result = evenLen(vec);
11
12    for (int i = 0; i < result.size(); i++)
13        cout << result[i] << endl;
14    //print
15    //ab
16    //abcd
17
18    return 0;
19 }
20
21 vector<string> evenLen(vector<string> vec) {
22     vector<string> result;
23
24     for (int i = 0; i < vec.size(); i++)
25         if (vec[i].size() % 2 == 0) //vec[i].size() can be replaced by vec[i].length()
26             result.push_back(vec[i]);
27
28     return result;
29 }
```

## 5 (10 points) Define a class.

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

```
1 #include <string>
2 class Course {
3 public:
4     ...//omitted
5 private:
6     std::string name; //represent course name
7     int credit; //represent number of credit hour
8 };
```

**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 235” and set data member **credit** to be 3.
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 **setName** to change the value of data member **name**.

**Answer:**

```
1 #include "Course.hpp"
2 #include <iostream>
3 #include <string> //need in some C++ versions
4 using namespace std;
5
6 Course::Course() {
7     name = "CS 235";
8     credit = 3;
9 }
10
11 Course::Course(string name, int credit) {
12     this->name = name;
13     if (credit > 0)
14         this->credit = credit;
15     else this->credit = 3;
16 }
17
18 void Course::setName(string name) {
19     this->name = name;
20 }
```

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

```

1 #ifndef Course_H
2 #define Course_H
3 #include <string> //need in some C++ version
4 class Course {
5 public:
6     Course();
7     Course(std::string name, int credit);
8     std::string getName() const;
9     int getCredit() const;
10    void setName(std::string name);
11    void setCredit(int credit);
12 private:
13     std::string name; //it is discouraged to use namespace in hpp,
14         //it might affect all the source code including this header file
15     int credit;
16 };
17 #endif

```

code of Course.cpp

```

1 #include "Course.hpp"
2 #include <iostream>
3 #include <string> //need in some C++ versions
4 using namespace std;
5
6 Course::Course() {
7     name = "CS 235";
8     credit = 3;
9 }
10
11 Course::Course(string name, int credit) {
12     this->name = name;
13     if (credit > 0)
14         this->credit = credit;
15     else this->credit = 3;
16 }
17
18 string Course::getName() const {
19     return name;
20 }
21
22 int Course::getCredit() const {
23     return credit;
24 }
25
26 void Course::setName(string name) {
27     this->name = name;
28 }
29

```

```
30 void Course::setCredit(int credit) {  
31     if (credit > 0)  
32         this->credit = credit;  
33 }
```

content of TestCourse.cpp

```
1 #include <iostream>  
2 #include "Course.hpp"  
3  
4 using namespace std;  
5  
6 int main() {  
7     Course cs;  
8  
9     cout << "name: " << cs.getName() << endl;  
10    cout << "credit: " << cs.getCredit() << endl;  
11  
12    cs.setName("CS 335");  
13    cs.setCredit(2);  
14  
15    cout << "name: " << cs.getName() << endl;  
16    cout << "credit: " << cs.getCredit() << endl;  
17    return 0;  
18 }
```

## 6 (10 point) Define a subclass

Here are part of Person.hpp of Person class.

```
1 class Person {
2 public:
3     Person(string name, int age); //non-default constructor of Person class
4     virtual string toString() const; //return a textual information of name and age.
5     ...//omit other constructors and methods
6 private:
7     string name;
8     int age;
9 };
```

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

**Define non-default constructor of Assistant**, which takes parameters name (a string), age (an int), and allowance (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 allowance should be positive. If parameter member allowance is not positive, set data member allowance to be 100.

**Override toString method inherited from Person class** to return a string representing the assistant's information like name, age, and allowance. 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:**

```
1 Assistant::Assistant(string name, int age, double allowance) : Person(name, age) {
2     if (allowance <= 0)
3         this->allowance = 100;
4     else this->allowance = allowance;
5 }
6
7 string Assistant::toString() const {
8     string str = Person::toString();
9     str += "allowance: " + to_string(allowance) + "\n";
10    //to_string(double) belongs to std namespace
11    return str;
12 }
```

The following is a complete code.  
code of Person.hpp

```
1 #ifndef Person_H
2 #define Person_H
3 #include <string> //needed
4
5 //we normally do not add using namespace std;
6 //in a header file (ended with .hpp),
7 //since the source code that include
8 //the header file may not like to use that namespace.
9
10 class Person {
```

```

11 public:
12     Person();
13     Person(std::string name, int age);
14     std::string getName() const;
15     int getAge() const;
16     void setAge(int age);
17     void setName(std::string name);
18     virtual std::string toString() const;
19
20 private:
21     std::string name;
22     int age;
23 };
24 #endif

```

code of Person.cpp

```

1  #include <iostream>
2  #include <string>
3  #include "Person.hpp"
4  using namespace std;
5
6  Person::Person() {
7      name = "John Doe";
8      age = 18;
9  }
10
11 Person::Person(string name, int age) {
12     this->name = name;
13     if (age >= 0 && age <= 130)
14         this->age = age;
15     else this->age = 18;
16 }
17
18 string Person::getName() const {
19     return name;
20 }
21
22 int Person::getAge() const {
23     return age;
24 }
25
26 void Person::setName(string name) {
27     this->name = name;
28 }
29
30 void Person::setAge(int age) {
31     if (age >= 0 && age <= 130)
32         this->age = age;
33 }

```



```

34
35 string Person::toString() const {
36     string str = "";
37     str += "name: " + name + "\n";
38     str += "age: " + to_string(age) + "\n";
39     return str;
40 }

```

code of Assistant.hpp

```

1  #ifndef Assistant_H
2  #define Assistant_H
3  #include <string>
4  #include "Person.hpp"
5
6  class Assistant : public Person {
7  public:
8      Assistant();
9      Assistant(std::string name, int age, double allowance);
10     double getAllowance() const;
11     void setAllowance(double allowance);
12     virtual std::string toString() const;
13 private:
14     double allowance;
15 };
16 #endif

```

code of Assistant.cpp

```

1  #include "Assistant.hpp"
2  #include <string>
3  using namespace std;
4
5  //invoke default construtor of Person
6  Assistant::Assistant() : Person() {
7      allowance = 100;
8  }
9
10 Assistant::Assistant(string name, int age, double allowance) : Person(name, age) {
11     if (allowance <= 0)
12         this->allowance = 100;
13     else this->allowance = allowance;
14 }
15
16 double Assistant::getAllowance() const {
17     return allowance;
18 }
19
20 void Assistant::setAllowance(double allowance) {
21     if (allowance >= 0 && allowance <= 4)
22         this->allowance = allowance;

```

```

23 }
24
25 string Assistant::toString() const {
26     string str = Person::toString();
27     str += "allowance: " + to_string(allowance) + "\n";
28     //to_string(double) belongs to std namespace
29     return str;
30 }

```

code of TestPersonAssistant.cpp

```

1  #include <iostream>
2  #include "Person.hpp"
3  #include "Assistant.hpp"
4  using namespace std;
5  //sample output:
6  //name: Ann
7  //age: 27
8  //
9  //name: Bob
10 //age: 17
11 //allowance: 200.000000
12 //
13 //name: John Doe
14 //age: 18
15 //allowance: 100.000000
16
17 int main() {
18     int size = 3;
19     Person** personPtr = new Person*[size];
20     //use Person* to store each information of a Person/Assistant
21     //to avoid information of Assistant from slicing
22
23     Person ann("Ann", 27);
24     personPtr[0] = &ann; //personPtr[0] is Person* type
25
26     Assistant bob("Bob", 17, 200);
27     //call non-default constructor of Assistant
28     personPtr[1] = &bob;
29
30     Assistant johnDoe; //call default constructor of Assistant
31     personPtr[2] = &johnDoe;
32
33     for (int i = 0; i < size; i++)
34         cout << personPtr[i]->toString() << endl;
35
36     delete[] personPtr;
37     personPtr = nullptr;
38     return 0;
39 }

```

## 7 (10 points) Define recursive function

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

For example, array with values 'a', 'b', 'a' is palindrome, but array with values 'a', 'b' 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:**

```
1 bool isPalindrome(char* arr, int size) {
2     if (size <= 1)
3         return true;
4
5     return arr[0] == arr[size-1] && isPalindrome(arr+1, size-2);
6 }
```

A complete code is as follows.

```
1 #include <iostream>
2 using namespace std;
3
4 bool isPalindrome(char* arr, int size);
5
6 int main() {
7     char arr[] = {'a', 'b', 'a'};
8     int size = sizeof(arr) / sizeof(arr[0]);
9
10    cout << boolalpha << isPalindrome(arr, size) << endl;
11
12    char arr2[] = {'a', 'b'};
13    int size2 = sizeof(arr2) / sizeof(arr2[0]);
14
15    cout << boolalpha << isPalindrome(arr2, size2) << endl;
16
17    char arr3[] = {'a'};
18    int size3 = sizeof(arr3) / sizeof(arr3[0]);
19
20    cout << boolalpha << isPalindrome(arr3, size3) << endl;
21    return 0;
22 }
23
24 bool isPalindrome(char* arr, int size) {
25     if (size <= 1)
26         return true;
27
28     return arr[0] == arr[size-1] && isPalindrome(arr+1, size-2);
29 }
```