Algorithms and Data Structures - Assignment 1

Basic C++: types, classes, pointers, I/O

Task A) C++ and OOP theory questions

1) Which of the following statements is true?

t a. Definitions and implementations are in the source file (*.cpp*) and the header file can be used for any purpose.
f b. A variable definition holds a reference to a value.
f c. Arrays start at index 1.
t d. In C++ a function has not to be part of any class.

2) Which of the following function types can not be declared as virtual?

f a. Constructor functions.
t b. Static functions.
f c. Destructor functions.
f d. Inline functions.

3) Insufficient use of which operator leads to memory leakage ?

a. sizeof.
b. Bitwise operators.
this c. new and delete.
d. Logical operators.

4) If you want to cast an unsigned char array into a short array, which expression is right? How is the memory being reorganized?

a. unsigned char *myUnsignedChar = dynamic_cast<unsigned char *>(myShort);
this one b. unsigned short *myUnsignedShort = reinterpret_cast<unsigned short *>(myUnsignedChar);
c. unsigned char *myUnsignedChar = reinterpret_cast<unsigned char *>(myShort);

the bytes stay the same, only the interpretation-byte changes.

d. unsigned char *myUnsignedChar = (unsigned char *)(myShort);

5) Given an array int a[10], what does the expression sizeof(a) tell you?

The number of bytes. So 10*the number of bytes per int on the machine.

a. The size of the pointer to the array.
b. The size of the array.
c. The size of the first element of the array.
d. The size of the type of the array.

6) Which of the following ways are valid to declare 2 pointers to an integer?

a. int* a, b;
b. int* my-pointer1, my-pointer2;
c. x int* p1, *p2;
d. x int *1stPointer; int *2ndPointer;

7) How can you declare and initialise an int with a constant?

a. x int a = 123;
b. x int a; a == 0
c. int a->123;
d. x int a(123);

8) Which statement about structs and classes are true?

a. x All declarations inside a struct are public by default.
b. In a struct you can only use basic data types.
c. Classes and structs must be separated into two files.
d. x Classes are successors of structs.

9) Given an integer pointer *ptr* and a reference *ref* to an integer, what does the expression ptr = &ref do?

a. x Memory location referenced by ref is stored in ptr.
b. Value referenced by ref is copied to the location ptr points to.
c. The value referenced by ref is copied by ptr.
d. Undefined behaviour occurs.

Task B) Basic C++ Commands

Note: in this and following exercises, the given skeleton code may at first not even compile without errors (because of lacking declarations, etc.)! Properly completing the tasks means everything will work.

1) checkRooms()

In the file *main.cpp* (folder *Task B*) you have the skeleton of a method named **checkRooms(House* myHouse, Desires myDesire)** which should check if there are enough rooms in the house. Your task is to complete the given skeleton so that the method returns true if the house has at least as many rooms as your desired house, and false otherwise.

What are the effects of passing the argument *myHouse* via a pointer?

2) checkPrice()

In the same file *main.cpp* you have the skeleton of a method named **checkPrice(House* myHouse, desires myDesire)** which should check if the price is in your desired budget. Your task is to complete the given skeleton so that the method returns true if the price is in the budget, or false if the price is not within the budget.

3) getAddress()

In the file *House.cpp* you have the skeleton of a method named **House::getAddress()** which should return a *std::string* with the address in it. Your task is to complete the given skeleton, such that the method returns the address in a string.

4) getPerimeter()

In the file *main.cpp* you also have the skeleton of a method named **getPerimeter(House* myHouse)** which should return the perimeter of the house (assume it is square-shaped). You need to use the **sqrt()** function to obtain the square root; make sure the *cmath* library is included.

5) Constructing and destructing

a. In the file *House.h* there is a constructor defined. Write the implementation of this constructor in *House.cpp*.
b. In the file *House.h*, declare a destructor. In *House.cpp*, define it so that it does nothing.

c. In the file *main.cpp*, declare a *Desires* instance called *myDesire* with a *new* statement. Remember to *delete* it later, when it is no longer needed. What is the purpose of this deletion? to free up memory

6) printHouseDescription()

In the file *main.cpp* folder you have the skeleton of a method named **void printHouseDescription(House* myHouse)** which should print out a description of the given house *myHouse*. Your task is to complete the given skeleton, such that the method prints out a nice formatted description. You can use the method **getAddress()** that you have implemented before.

Task C) Input/Output processing

This exercise aims to strengthen your C++ syntax and file reading skills. We have a list of 8 colors (file *colors.txt*) in the format *name r g b* per line, where r, g and b correspond to its red, green and blue RGB model components, respectively. Each color value x is an integer satisfying $0 \leq x \leq 255$.

Your goal is to implement an application that reads the color list, saves it to a database and then can be repeatedly queried by the user. The necessary files are in the folder *Task C*.

1) Reading and storing the list

In the file *ColorTable.cpp*, complete the function **readColorTable(std::string filename)**, so that reads the file *colors.txt* and stores its information in an appropriate data structure.

Hint: an array of structs is an example of a suitable structure.

2) Querying a color

Complete the function **getColor(std::string name, Color *foundColor)** so that allows the user to search a color in your data structure by typing its name. It should return whether the color has been found or not. In positive case, *foundColor* should point to it.

3) User interaction

Use the previous functions to implement in *main.cpp* an interactive program that reads

the colors and then gives RGB information about the colors the user asks. After the input "quit", the program should exit. A sample execution follows:

```
Please enter a color name: magenta
magenta = (255,0,255)
Please enter a color name: yellow
yellow = (255,255,0)
Please enter a color name: blahblah
Could not find color "blahblah"
Please enter a color name: quit
Thanks and goodbye!
```

Task D) Strings

In this exercise, you have to implement functions for string processing. You are not allowed to make use of the library *string.h*.

1) Size of a string

In the file *strings.cpp* (folder *Task D*) we provide you a function skeleton named **getSize(char* string)** which should compute the length/size (i.e. the number of characters) of the given C-string argument. Remember that a C-style string is in fact an array of characters. Your task is to complete the function, in order that it fulfills its task.

2) Joining two strings

Complete the function **void concatenate(char str1[], char str2[])**, so that it joins together its two arguments and prints the result to the console via standard output.

3) STL strings: isPalindrome()

Complete the function **isPalindrome(std::string str)**, which checks if the given Standard Template Library (STL) string is a palindrome and returns the result as a boolean value.

Note: A palindrome is a string that reads the same forwards and backwards, character by character.

Example: *aabbcbbaa* is a palindrome, whereas *abacba* is not.

4) Make a figure out of characters

Complete the function **printShape(int level, char character)** so that a triangle of the following kind is printed in the standard output:

```
         *
        ***
       *****
      *******
     *********
    ***********
   *************
  ***************
*****************
```

The parameter *level* indicates the requested height of the triangle (in the example, 9). The parameter *character* determines which symbol must be used to fill it (in the example, '*').

Task E) Inheritance, dynamic binding, enumerations

This task aims to get you used to C++ inheritance. Enumerations are covered as well. Please keep in mind that a correct sample output does not necessarily mean your implementation is correct in general.

1) Inheritance

Do the necessary changes to make the *Car* and the *Ship* classes inherit from *Vehicle*. They must have access to the field *maximumVelocity*, so that it can be consulted. In addition, this field has to be initialized during object construction so as to produce the following output (uncomment the *TODO 1* lines from the file *Inheritance.cpp*, in the folder *Task E*):

```
Vehicles velocity:
Car: 10
Ship: 15
```

2) getLength()

Change the given code to make the function **getLength()** behave differently on each of the subclasses *Car* and *Ship*: it should return the normal length in the first case, and the length plus the engine length in the second case. Initialize the normal length during

object construction to 10, so as to produce the following output (uncomment the *TODO 2* lines):

```
Vehicles length:
Car: 10
Ship: 12
```

3) Enumerations

Create a class called *AmphibicVehicle*, which inherits from *Car* and *Ship*. Furthermore, an *AmphibicVehicle* must be able to change its usage mode (behaving either as a *Car* or as a *Ship*). Store the current state by means of an enumeration, implement the functions **AmphibicVehicle::printUsageMode()** and **AmphibicVehicle::changeUsageMode()**, and check they work by uncommenting the *TODO 3* lines.

4) Adding wheels to your car

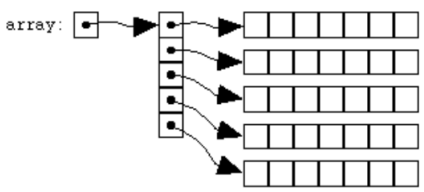
Make sure that whenever you construct a car, you can choose its wheel models (by means of a parameter). Construct the wheels with the *new* statement. Implement also the function **Car::hasSummerWheels()**, which should return true if and only if all the wheels are summer ones. Uncomment the *TODO 4* lines and construct an instance *myCar* so that the output is:

```
Has summer wheels: 1
```

Task F) Pointers in C++

In this task, two $n \times n$ matrices have to be subtracted with the help of pointers. To do this you are provided with a framework, which you must complete. The function **fill(int** &x, int row, int col)** fills the matrix via standard input. The parameters *row* and *col* indicate the number of rows and columns, respectively.

Hint: a pointer to another pointer is actually pointing to a block of pointers, each of which points to a block of memory. The result is therefore a reference to a 2-dimensional area (see the following picture).



1) Initialise memory

In the file *matrix_subtraction.cpp* (folder *Task F*), complete the function **initialiseMemory(int **&x, int row, int col)**, so that the memory referenced by the given double pointer x is filled with zeros.

2) Subtract matrices

Complete the function **int** subtractMatrices(int **x, int **y, int row1, int col1, int row2, int col2)**, so that it returns the subtraction of the matrices x and y.

Hint: the subtraction of two equally-sized matrices has again the same size, and is defined as the element-wise subtraction.

3) Deallocate memory

Complete the function **deallocateMemory(int **&val, int x, int y)**, so that all the memory referenced by the 2-dimensional pointer *val* gets deallocated.