UNIVERSITY OF CALIFORNIA, MERCED

BERKELEY · DAVIS · IRVINE · LOS ANGELES · MERCED · RIVERSIDE · SAN DIEGO · SAN FRANCISCO

SANTA BARBARA · SANTA CRUZ

CSE24: Advanced Programming

Final Examination

Fall 2021

Instructions

- This is a closed book exam. No notes and/or electronic devices may be used.
- Answer every question on an answer paper/booklet with your name on it.
- Make sure you number every question accurately on your answer paper.
- You have 3 hour to complete this exam.
- If you are unsure of anything, please ask.

Section	**Points**	**Score**
Linux Command Line	20	
Basic Programming	50	
Pointers	30	
Dynamic Memory Management	30	
Custom Data Structures	50	
Standard Template Library	20	
TOTAL	**200**	

1 Linux Terminal Commands [20 points]

1.1. What is the terminal command for navigating to the parent of the folder we are in? [2 points]

1.2. Suppose there is a folder called `stuff` inside your home folder. Assuming we are in some random folder on our hard drive, write the terminal command/s that would display the contents of the `stuff` folder referred to in this question? [3 points]

1.3. Write the terminal commands for performing the following sequence of tasks: [15 points]

- Ensure that we are currently in the home directory
- In the home folder, create two folders, `left` and `right`
- Create a file called `one.txt` in the `left` folder
- Create a file called `two.txt` in the `right` folder
- In the `left` folder, rename the file `one.txt` to `file1.txt`
- Delete the entire `right` folder

2 Basic Programming [50 points]

2.1. Write a complete C++ program that outputs the string `"Hello World!"`. Assuming your program is saved in a file called `hello.cpp`, and your terminal is currently in the same folder as this program, write the terminal commands that you would use to compile and run your program. [5 points]

2.2. Write a C++ program that contains the following declarations in its `main` function. You need to choose appropriate data types for each: [10 points]

- variable called `a` storing the value `17`
- variable called `b` storing the value `2.3`
- variable called `c` storing the value `"Avery"`
- variable called `d` storing the value `true`
- variable called `e` storing the value `"false"`
- variable called `f` storing the value `"A"`
- variable called `g` storing the value `"23"`
- variable called `h` storing the value `0`
- variable called `i` storing the value `126`
- variable called `j` storing the value `'A'`

2.3. Write a C++ program that asks the user for two numbers. Store them in two variables called `x` and `y`. Then ask the user to input an arithmetical operation, namely `+`, `-`, `*`, `/`, which represent addition, subtraction, multiplication, and division, respectively. Store the operation in a variable called `op`. Choose appropriate data types for each variable. Finally, output the result of applying the given operation on the two numbers provided. [10 points]

2.4. Write a C++ program that asks the user to enter their name. Store their input in a variable called `name`. Calculate the number of vowels in their name. Store that result in a variable called `vowelCount`. As a reminder, the vowels are "A", "E", "I", "O", and "U". Output a message for the user, of the form `Your name contains <x> vowel/s`, where `<x>` is the number of vowels in their name. [10 points]

2.5. Write a C++ program that asks the user to enter some numbers and then adds them up, printing our the result. The user should be allowed to enter as many numbers as they wish, and can indicate that they are done by entering -1. The -1 should not be used in the calculation.

Your program should store these numbers in a vector called `nums`. As the user enters the numbers, your program should be adding them to the vector in real time.

Once the numbers have all been entered, your program should add them up, storing the result in a variable called `total`.

Finally, your program should output a message of the form `Your numbers add up to <x>`, where `<x>` is the sum of all elements in the `nums` vector. [15 points].

3 Pointers [30 points]

3.1. Write a complete C++ program and in its `main` function declare an `int` variable called `num` and initialize it to 17. Get a pointer `p` to the address of `num` and use it to print out the address of `num`. Now use your pointer `p` to change the value of `num` to 15. Print out the value of `num` using the line `cout << num << endl;` [5 points]

3.2. Write a complete C++ program. In its `main` function declare three `int` variables, named `x`, `y`, and `z`. Get a pointer `p` to `x` and perform the following tasks using only the pointer `p`: [10 points]

- set the value of `x` to 101
- print out the address of `x`
- print out the address of `y` without moving the pointer `p` to `y`
- move the pointer `p` to `y`
- set the value of `y` to 14
- move the pointer `p` to `z`

3.3. Write a C++ function called `inc`. The function should not return anything but it should be able to take in an `int` variable and increment it by 1. When the function is called, it should behave as in the example below. [5 points]

```
int x = 7;

cout << x << endl; // Output: 7

inc(x);

cout << x << endl; // Output: 8
```

3.4. Write a complete C++ program. In its `main` function declare an `int` variable called `year` and initialize it to 2021. As we have learned in class, an `int` variable occupies 4 bytes of memory. In the case of 2021, these are [229], [7], [0], [0]. Create an `unsigned char` pointer named `p`, and use it to step through the bytes occupied by `year`, and print out the value stored in each byte. [10 points]

4 Dynamic Memory Management [30 points]

4.1. Write a complete C++ program. In its `main` function, set aside space on the heap that is large enough for 5 `int` values. Store the values 1, 2, 3, 4, 5 in the space. Release the heap memory you have occupied. [5 points]

4.2. Write a complete C++ program that illustrates a memory leak. You need to have the complete code, and an explanation as to where and why the memory leak occurs. [10 points]

4.3. Write a complete C++ program. In its `main` function, set aside space on the heap for 5 integers. The pointer to the beginning of that space should be called `store`. Fill it up with values 1, 2, 3, 4, 5. Now increase the size of the `store` so that it can hold 7 `int` and add the values 99, and 101 to it. Take all the necessary precautions to ensure the safety of this operation. [15 points]

5 Custom Data Structures [50 points]

5.1. Create a C++ struct called `Car`. The struct should store the make and model as strings, the year an integer. We should be able to run the following code, assuming the definition of your struct is available: [10 points]

```
Car myCar("Tesla", "Model Y", 2021);

cout << myCar << endl; //Output: Tesla, Model Y, 2021
```

5.2. Now create another struct called `Lot` (as in parking lot), which is a container for `Car` objects. Your `Lot` should be able to store up to 4 `Car` objects on the heap.

We should be able to print the `Lot` with `cout`, we should be able to initialize and assign `Lot` objects as copies of other `Lot` objects, and when we delete `Lot` objects, they should correctly free up the space they occupied on the heap.

The default way to create a `Lot` object is to have it empty (containing no `Car` objects). There should be a method called `add` that takes in 2 `string`s (make and model respectively) and an `int`, builds a `Car` object out of those and stores it in the `Lot`.

There is no need to inflate the `Lot` storage and there is no need to implement a remove method, but if the user tries to add more than 4 cars, they should not be allowed to do it. [40 points]

6 Standard Template Library [20 points]

6.1. Assume we have a vector called `nums` containing `int` values. Write a `for` loop using iterators to print out the values in the vector. [5 points]

6.2. Create a `vector` that contains pointers to `Car` objects. Insert 2 `Car` objects using the `vector`'s `push_back()` method. Iterate over the `vector` using iterators and `cout` each `Car`. [15 points]