CS256 — Exercise 4

January 27, 2016

Due: Monday, February 1, 2016 before midnight (40 points)

1 Preparing the Project

1. Go to https://codebank.xyz and create a new project named `CS256-EX4`.
2. On your local machine, from a terminal or git bash navigate to the folder you use for storing CS256 related files and create a new directory to store this exercise. From now on, we'll call this directory the *working directory*.
3. `cd` in to the working directory and run:

   ```
   $ git init
   $ git remote add origin https://codebank.xyz/username/CS256-EX4.git
   ```

   where `username` is your bronconame or you can clone the empty repository with:

   ```
   $ git clone https://codebank.xyz/username/CS256-EX4.git
   ```

4. Now, the directory on your machine is a git repository with a reference to the remote repository on https://codebank.xyz.

2 Memory

Create a C++ source code file called `Memory.cpp` that will contain the following two functions.

2.1 Allocation

Create a function with the following prototype:

```
int*** alloc3D(int x, int y, int z);
```

This function should dynamically allocate a 3D array of `int`s with the dimensions passed as arguments. Remember, you need to allocate in several steps similar to the 2D allocation.

2.2 Deallocation

Create a function with the following prototype:

```
void dealloc3D(int*** a, int x, int y);
```

This function is the reverse of `alloc3D`, it should deallocate the array passed as argument `a`. Remember, you need to deallocate in several steps similar to the 2D allocation.

You should also include the following `main` function to test your allocation:

```
int main()
{
    while (true)
    {
        int*** a = alloc3D(100, 100, 100);
        dealloc3D(a, 100, 100);
    }
    return 0;
}
```

3 Concatenation

On Unix-like systems, the `cat` program will list out the contents of the files passed as command line arguments. Replicate this behavior in a C++ file named `Cat.cpp`. Your program should go through each command line argument passed as an input file and output its contents exactly as contained in the input file to `cout`. If you encounter an error trying to read one of the files or the argument passed is not a valid file, print an error message and exit.

3.1 Sample Output

Suppose we have the following two files in our working directory: `1.txt` and `2.txt`.

The contents of `1.txt` are:

```
abc
def
ghi
```

The contents of `2.txt` are:

```
123
456
789
```

Running our `Cat` program should give us the following output:

```
$ ./Cat 1.txt 2.txt
abc
def
ghi
123
456
789
```

Here's another example, where the non-existant `3.txt` is also given as an argument:

```
$ ./Cat 1.txt 2.txt 3.txt
abc
def
```

```
ghi
123
456
789
Invalid file: 3.txt
```

4 Submission

Once you have completed the program, you can use `git add`, `git commit`, and `git push` to push the changes to https://codebank.xyz. You can make as many commits and push as many times as desired until the deadline.