Midterm 2 Practice

All code should be written in C++. Unless otherwise specified, you may (and I generally will):

- assume that the user of any code you write will be cooperative with the input they supply;
- omit `std::` and the return value of `main()`;
- assume that all necessary libraries have been `#include`d;
- omit `main()` entirely for problems that ask ONLY for function definitions;
- not concern yourself with having optimal solutions (within reason);
- not worry yourself about prompt messages for user input (I sometimes give descriptive prompts to clarify problems);
- not recopy code I have provided, or which you have written in other parts of problems.

1. Write a function named `repeat` which receives one `int` array named `x` and an `int` argument `length`, and returns an `int` pointer. The return value should point to an array that is twice the length of `x`, which contains `x` concatenated with itself: that is, the first `length` entries of the new array will just be the entries of `x`, and the last `length` entries of the return will just be `x` again.

   For example, the following code should print `1 4 9 1 4 9`:

   ```
   int x[] = {1,4,9};
   int *y = repeat(x, 3);
   for(int i = 0; i<6; ++i){
       cout << y[i] << " ";
   }
   ```

2. Consider the following program:

   ```
   int main() {
       string names[] = {"alice", "joe", "andrew", "bob", "carol", "charles"};
       int length = 6;
       char **table = new char*[length];
   }
   ```

   I wish to have each entry of `table` point to an array of `char`s which contains the letters from the corresponding `string`. For example, with the given contents of `names`, the first entry of `table` should point to an array of `char`s of length 5, containing `'a', 'l', 'i', 'c', 'e'` as entries; the second entry should point an array of `char`s of length 3 containing `'j', 'o', 'e'`, etc.

   Write code that fills out `table` as described, which works even if `names` was initialized with different values (but such that `length` still is the length of `names`).

   At the end, also write code that deallocates all dynamically allocated memory.

3. I have a file named `script.txt` which contains several lines of dialogue for a play with several roles. For example, the file could look like

   ```
   Alice : Hello, my name is Alice. Who are you?
   Bob : I'm Bob, and I'm the main character in this play.
   Alice : I disagree, I am the main character. I have the most lines.
   Bob : I think we have an equal number of lines.
   Charlie : What about me? I have a line too!
   ```

   Each line of the file should start with the (single-word) name of a character, followed by a space, followed by a `:` and some dialogue.

   Write code which prints out the name of each character who appears in the play. It is ok if a character's name appears several times.

4. Write a program which reads through the text file from the last problem, and prints out the LONGEST line (the line with the most characters).

5. Consider the following code.

   ```
   int main() {
      int x = 23, y = 5;
      string s;
   ```

```
    char c;
    cin >> x >> y >> s >> c;
    cout << x << " " << y << " " << s << " " << c;
}
```

What would happen if the user entered each of the following? Be as precise as possible.

a. `102 304 hello world goodbye`

b. `a b c d`

c. `12 34hijk.`

6. Write a program that opens a text file named `script.txt`, and then creates a new text file called `tpircs.txt`, by taking each line of `script.txt` and writing the words in reverse order. (A "word" should be considered to be a run of consecutive, non-whitespace characters.) For example, with the file from Problem 20, the contents of the new file would be

```
you? are Who Alice. is name my Hello, : Alice
play. this in character main the I'm and Bob, I'm : Bob
lines. most the have I character. main the am I disagree, I : Alice
lines. of number equal an have we think I : Bob
too! line a have I me? about What : Charlie
```

Accomplish this WITHOUT using arrays (or vectors) of any sort. Instead, utilize string concatenation and stringstreams.

7. LossAmp is a program that plays music files on your computer.

Consider a class called `Playlist`. Each object is meant to represent a playlist: a list of up to 100 filenames of music files on your computer, to be played in order. Each `Playlist` object should have the following **private member variables**:

- `string files[100]`, an array of strings, representing the **names of the files** on your computer;
- `int num_entries`, an integer which represents *the number of entries currently stored in the playlist*, which has maximum 100; and
- `int current`, an integer, which represents **the index of the file currently being played**: this should start out as 0 (representing the first track), but may get updated later.

The class should also support the following **methods**:

- a default construct which simply sets `num_entries` and `current` to 0.
- `void add_track(string)`, which adds the parameter as the next entry in `files`, if there is room – in that case, other member variables should be updated appropriately.
- `string current_track()`, which returns the filename of the track currently being played (the name in `files` corresponding to the integer `current`).
- `int length()`, which returns the number of entries in the playlist.
- `void fwd()`, which moves the playlist 1 track forward: more specifically, it adds `1` to `current`. *If this causes `current` to equal the length of the playlist, the playlist should go back to the first track!*

a. Write the declaration for the class `Playlist`, and the definitions for all methods.

b. Write code in `main()` which creates a `Playlist` object called `tunes`, and adds `"mmm_bop.mp3"`, `"what_is_love.mp3"`, and `"what_do_i_get.mp3"` to the list.

c. Write additional code in `main()` which checks whether `"mac_arthur_park.mp3"` is in `tunes`, using the methods given above.

8. WaitList is a program that helps doctor's offices manage their wait-lists.

Consider a class called `Waitlist`. Each object is meant to represent a wait-list: a list of names of patients, to be seen in order. Each `Waitlist` object should have the following **private member variables**:

- `vector <string> patients`, a vector of strings, representing the **names of patients**;
- `int num_patients`, an integer, representing the number of patients in the list.

The class should also support the following **public member functions**:

- a default constructor which takes NO (outside) arguments: it should just initialize `patients` to be an empty vector and `num_patients` to be 0.

- `void add(string)`, which takes one string `n` as outside argument, and adds this name to `patients` (and updates `num_patients` appropriately.
- `string call()`: if there is at least one patient, this function should remove the FIRST patient from the waiting list, and change `num_patients`, and return the name of that patient. If there are no patients, the function should return the empty string and do nothing else.

a. Write the class declaration for the class `Waitlist`. Make member functions `const` as appropriate.

b. Write the implementations of the methods as described above.

c. In `main()`, create a `Waitlist` object named `www`. Then, add two patients to it: one named `Evan`, and one named `Frank`.

d. Still in `main()`, suppose that `www` has accumulated more patients. Write a loop which prints out all the patients in the `Waitlist`, in order, until there are no more patients.

9. A 2×2 matrix is an arrangement of numbers of the form $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$. A 2×1 column vector is a pair of numbers arranged vertically, like $\begin{bmatrix} x \\ y \end{bmatrix}$. If M represents the matrix $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ and v represents the column vector $\begin{bmatrix} x \\ y \end{bmatrix}$, then the product $M * v$ is given by the column vector $\begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$. You can also add two matrices element-wise: e.g. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 4 \\ 1 & 10 \end{bmatrix} = \begin{bmatrix} 6 & 6 \\ 4 & 14 \end{bmatrix}$.

   Write two class definitions, with enough member functions so that the following code works, and prints out `66, 54`.

```
int main() {
   Matrix m1(1,2,3,4), m2(5, 4, 1, 10);
   ColVector v(10, 1);
   ColVector s;
   s = (m1 + m2) * v;
   cout << s;
}
```

   Note: for this problem, the `Matrix` class and `ColVector` class reference each other, which makes it difficult to know which class to declare first. To get around this, you can provide a *forward declaration* of the `ColVector` class, by simply putting

```
class ColVector;
```

   ahead of the definition of the `Matrix` class, and then providing the `ColVector` class later.

10. Fakemon is a game that is popular with the kids. In Fakemon, there are a number of species of fake animals that each have a number and a stupid name. For example, Fakemon # 57 is named Snurpdup. Of course, Fakemon # 0 is the famous Fikachu.

    A `Fakedex` is an object which contains two arrays: an array of strings representing the names corresponding to each number, and an array of `bool`s indicating whether or not a player has the Fakemon with that index.

    A **PARTIAL** class declaration is provided below:

```
class Fakedex {
private:
    int capacity; // Number of Fakemon currently capable of being stored
    string *names;
    bool *have;
    void reserve(int); // Should change ‘capacity‘ and the lengths of arrays pointed to by ‘names‘ and ‘have‘
                       // so that the all are equal to the argument

public:
    Fakedex(): capacity{100}, names{new string[100]}, have{new bool[100]} {}
    void new_name(int, string); // Introduces a new name into ‘*names‘ at a given index,
                                // sets corresponding entry in ‘*have‘ to be ‘false‘
    void catch_em(string); // Set corresponding index in ‘*have‘ to ‘True‘,
                           // if the parameter is present in ‘*names‘
    bool operator[](string); // If the parameter is in ‘*names‘, return the corresponding value of ‘*have‘,
                             // otherwise return ‘false‘


    ~Fakedex() {delete[] names; delete[] have;}
```

```
};
```

a. Implement `void reserve(int n)`, so that if `n > capacity`, it changes the value of `capacity`, and reallocates `*names` and `*have` so that they have at least `n` entries, of course saving older entries. Implement in such a way that there are no memory leaks!

b. Implement `void new_name(int x, string y)` so that the `y` is added to `*names` at index `x`, and the corresponding entry of `*have` should be set to `false`. If `x` is big enough, you may need to call `reserve()`.

c. Implement `void catch_em(string y)` so that if `y` is currently present in `*names`, the corresponding index in `*have` is set to `True`. (If `y` is NOT present, do nothing.)

d. Implement `bool operator[](string y)` so that if `y` is currently present in `*names`, the corresponding index in `*have` is returned. If `y` is NOT present, return `false`.

e. The class as it stands is rather likely to have various types of memory errors, even when used in fairly reasonable ways. What additional methods should be implemented? Add them to the class declaration, and implement them.

11. Consider the class shown below:

```
class Something {
private:
   int x;
   string* y;
public:
   Something(int a): x{a}, y{new string[100]} {}
   ~Something() {delete[] y;}
   void operator=(const Something &rhs){
       x = rhs.x;
    }
   string entry(int n) {return y[n];}
   bool same (Something rhs) {
      for (int i = 0; i <= 99; ++i) {
         if (y[i] != rhs.y[i]) {
            return false;
         }
      }
      return true;
   }
};
```

For each of the following, explain clearly the error that would occur when we try to compile and run the following bit of code, being specific about the line that produces the error. Would the error be picked up by the compiler?

a.

```
int main() {
   Something one(5), two;
   cout << one.entry(50) << endl;
   cout << two.entry(50) << endl;
}
```

b.

```
int main() {
   Something three(5);
   Something four(6);
   cout << three.x + four.x << endl;
}
```

c.

```
int main() {
   Something blah(20), blech(24);
   cout << blah.entry(5) << endl;
   cout << blah.same(blech) << endl;
}
```

d. For this one, what small change can you make to the class that will cure the error?

```
int main() {
   const Something q(5);
   Something z(10);
   z = q;
   string w = q.entry(44);
   string j = z.entry(44);
}
```

12. Consider the `List` class from lecture. Write the implementations of the following functions:
    - `void erase(int n)`, which removes the element with index `n` from the list, moving all later elements forward, and updating the other members appropriately (note: the array holding the contents should NOT be replaced in the operation).
    - `void insert(int n, string s)`, which inserts the string `s` into the list at index `n`, and moves all the elements at index `n` or later back one position.
    - `List& operator+=(const List& rhs)`, which concatenates the contents of `rhs` on to the end of the left-hand-side `List`.
13. Consider the following code:

```
class Record {
public:
    string name;
    int id_num;
    Record(string n, int x): name{n}, id_num{x} {cout << "R1\n";}
    Record(string n): name{n}, id_num{0} {cout << "R2\n";}
    ~Record() {cout << "~Des\n";}
    Record(const Record &rhs) {
        cout << "R3\n";
        name = rhs.name;
        id_num = rhs.id_num;
    }
    void operator=(const Record &rhs){
        cout << "Op=\n";
        name = rhs.name;
        id_num = rhs.id_num;
    }
};

void print(Record arg){
   cout << arg.name << endl;
}

int main(){
   Record x("Hello", 123);
   Record y = x;
   print(x);
   x = y;
}
```

    a. What would print out?

    b. What would print out if the signature line of `print()` was changed to `void print(Record &arg)`?