PS6

- Don't wait till the last moment.

`Q1`

One problem with dynamic arrays is that once the array is created using the new operator the size cannot be changed. For example, you might want to add or delete entries from the array similar to the behavior of a `vector`. This project asks you to create a class called `DynamicStringArray` that includes member functions that allow it to emulate the behavior of a vector of strings. The class should have:

- A private member variable called `dynamicArray` that references a dynamic array of type `string`.
- A private member variable called `size` that holds the number of entries in the array.
- A default constructor that sets the dynamic array to `NULL` and sets size to `0`.
- A function that returns size.
- A function named `addEntry` that takes a string as input. The function should create a new dynamic array one element larger than `dynamicArray`, copy all elements from `dynamicArray` into the new array, add the new string onto the end of the new array, increment size, delete the old `dynamicArray`, and then set `dynamicArray` to the new array.
- A function named `deleteEntry` that takes a string as input. The function should search `dynamicArray` for the string. If not found, return `false`. If found, create a new dynamic array one element smaller than `dynamicArray`. Copy all elements except the input string into the new array, delete `dynamicArray`, decrement size, and return `true`.
- A function named `getEntry` that takes an integer as input and returns the string at that index in `dynamicArray`. Return `NULL` if the index is out of `dynamicArray`'s bounds.
- A copy constructor that makes a copy of the input object's dynamic array.
- Overload the assignment operator so that the dynamic array is properly copied to the target object.
- A destructor that frees up the memory allocated to the dynamic array.

Use the following main function (Change the object data):

```
int main()
{
    DynamicStringArray names;

    // List of names
    names.addEntry("Frank");
    names.addEntry("Wiggum");
    names.addEntry("Nahasapeemapetilon");
    names.addEntry("Quimby");
    names.addEntry("Flanders");

    // Output list
    cout << "List of names:" << endl;
    for (int i = 0; i < names.getSize(); i++)
        cout << names.getEntry(i) << endl;
    cout << endl;

      // Add and remove some names
      names.addEntry("Spuckler");
      cout << "After adding a name:" << endl;
    for (int i = 0; i < names.getSize(); i++)
        cout << names.getEntry(i) << endl;
    cout << endl;

      names.deleteEntry("Nahasapeemapetilon");
      cout << "After removing a name:" << endl;
    for (int i = 0; i < names.getSize(); i++)
        cout << names.getEntry(i) << endl;
    cout << endl;

      names.deleteEntry("Skinner");
```

```
    cout << "After removing a name that isn't on the list:" << endl;
  for (int i = 0; i < names.getSize(); i++)
      cout << names.getEntry(i) << endl;
  cout << endl;

    names.addEntry("Muntz");
    cout << "After adding another name:" << endl;
  for (int i = 0; i < names.getSize(); i++)
      cout << names.getEntry(i) << endl;
  cout << endl;

    // Remove all of the names by repeatedly deleting the last one
    while (names.getSize() > 0) {
      names.deleteEntry(names.getEntry(names.getSize() - 1));
    }

    cout << "After removing all of the names:" << endl;
  for (int i = 0; i < names.getSize(); i++)
      cout << names.getEntry(i) << endl;
  cout << endl;

    names.addEntry("Burns");
    cout << "After adding a name:" << endl;
  for (int i = 0; i < names.getSize(); i++)
      cout << names.getEntry(i) << endl;
  cout << endl;

  cout << "Testing copy constructor" << endl;
  DynamicStringArray names2(names);
  // Remove Burns from names
  names.deleteEntry("Burns");
    cout << "Copied names:" << endl;
  for (int i = 0; i < names2.getSize(); i++)
      cout << names2.getEntry(i) << endl;
  cout << endl;

  cout << "Testing assignment" << endl;
  DynamicStringArray names3 = names2;
  // Remove Burns from names2
  names2.deleteEntry("Burns");
    cout << "Copied names:" << endl;
  for (int i = 0; i < names3.getSize(); i++)
      cout << names3.getEntry(i) << endl;
  cout << endl;

  cout << "Enter a character to exit." << endl;
  char wait;
  cin >> wait;
  return 0;
}
```

Q2

This Programming Project explores how the unnamed namespace works. Listed below are snippets from a program to perform input validation for a username and password. The code to input and validate the username is in a separate file than the code to input and validate the password.

File header `user.cpp`:

```
namespace Authenticate
```

```
{
  void inputUserName()
  {
    do
    {
     cout << "Enter your username (8 letters only)" << endl;
     cin >> username;
    } while (!isValid());
  }

  string getUserName()
  {
     return username;
  }
}
```

Define the `username` variable and the `isValid()` function in the unnamed namespace so the code will compile. The `isValid()` function should return true if `username` contains exactly eight letters. Generate an appropriate header file for this code. Repeat the same steps for the file `password.cpp`, placing the `password` variable and the `isValid()` function in the unnamed namespace. In this case, the `isValid()` function should return true if the input password has at least 8 characters including at least one non-letter:

File header `password.cpp`:

```
namespace Authenticate
{
  void inputPassword()
  {
    do
    {
     cout << "Enter your password (at least 8 characters " <<
             "and at least one non-letter)" << endl;
     cin >> password ;
    } while (!isValid());
  }

  string getPassword()
  {
     return password;
  }
}
```

At this point you should have two functions named `isValid()`, each in different unnamed namespaces. Place the following `main function` in an appropriate place. The program should compile and run.

```
int main()
{
  inputUserName();
  inputPassword();
  cout << "Your username is " << getUserName() <<
          " and your password is: " <<
          getPassword() << endl;
  return 0;
}
```

Test the program with several invalid usernames and passwords.

Turn In

- Make and submit a zip file(`<your_full_name>_PS6.zip`) which includes the following:

– Source code of `Q1`
– Source code of `Q2`
– Run `Q1` and `Q2` and attach screenshots(in `jpg` format, not exceeding `300kb` each) which show that your programs are running.
– At least `1` screenshot for each question.