CS 7B - Fall 2018 - Final Exam

Write responses to following on separate paper. Computers may be used during the second half.

1. Explain what the following program does:

```
#include <iostream>
using namespace std;
int main() {
   int num;
   float sum=0, term=1;
   cout<<"Enter the number:";
   cin>>num;
   for(int n=num; n > 0; n /= 10 ) {
      int digit = n % 10;
      sum += digit * digit * digit;
   }
   if( sum == num )
      cout << "This is an Angstrom number" << endl;
   else
      cout << "This is not an Angstrom number" << endl;
}
```

ANS: The big picture is that it adds up the cubes of the digits of a positive integer and, if that sum is the same as the integer, announces "This is an Angstrom number," or, if not the same, "This is not an Angstrom number". It does this by isolating the digit in the one's place and adding its cube to a running total, `sum` and then replacing the given integer by its integer quotient after division by 10 (essentially just right-shifting the base-ten number) and repeating the process until there are no more digits left.

Here is a typical run, showing that, while 135 is not an Angstrom number, 153, 407, 370 and 371 are Angstrom numbers.

```
Enter the number:135
This is not an Angstrom number
Enter the number:153
This is an Angstrom number
Enter the number:407
This is an Angstrom number
Enter the number:370
This is an Angstrom number
Enter the number:371
This is an Angstrom number
```

2. This problem has three parts:

   (a) Create a `Time` class that stores a time as a single number of type `time_t` (this is just another name for a certain type of integer). The constructor should take one argument – the initial value to set the internally stored time to. This argument should have a default value of time(0) (the time function is defined in the C++ Standard Library header ⟨ctime⟩. Also create a `getter` function for the time stored in `Time` objects, and a `setter` function to allow changing the time the `Time` object stores later on.
   SOLN: This is in lines 5-16 of the code that follows.

   (b) Rewrite the constructor to use the member initializer syntax instead of an assignment statement.
   SOLN: `Time(time_t tin) : t(tin) {}`

(c) Write a driver with `main()` to test your class, producing user dialog on the console like this:

```
tNow = 1544552083
The time is now 17
Enter a new value for time: 18
The new time is 18
```

SOLN: Here's what I've got:

```
#include <ctime>
#include <iostream>
using namespace std;

class Time {
  time_t t;
public:
  Time() { t = time(0); }
  Time(time_t tin) {
    t = tin;
  }
  time_t getTime() { return t; }
  void setTime(time_t tin) {
    t = tin;
  }
};

int main() {
  time_t time;
  Time tNow;
  cout << "tNow = " << tNow.getTime() << '\n';
  Time t(17);
  cout << "The time is now " << t.getTime() << '\n';
  cout << "Enter a new value for time: ";
  cin >> time;
  t.setTime(time);
  cout << "The new time is " << t.getTime() << '\n';
  cin.ignore();
  cin.get();
}

/* Sample run:
tNow = 1545061998
The time is now 17
Enter a new value for time: 1545062011
The new time is 1545062011

*/
```

3. Debug the following program, whose intent is clear enough. Include the entire fully functional program as your answer.

```
class member {
    int memberNum = 25;
    float memberPay;
public
    void Input(cin >> memberNum >> memberPay);
    void Output;
}
int main() {
    Member mem;
```

```
10      cin >> mem.memberNum >> mem.memberPay;
        cout << mem.memberNum << '\t' << mem.memberPay;
12  }

14  void Output::Member() {
        cout << mem.memberNum << '\t' << mem.memberPay;
16  }
```

SOLN: ¡Shudder¿ That was terrible! Here's something that works and captures the general intent/-functionality of that class:

```
#include <iostream>
2   using namespace std;

4   class Member {
      int memberNum = 25;
6     float memberPay;
    public :
8     Member() {};
      Member(int mn, float mp) : memberNum(mn), memberPay(mp) {    }
10    void Input();
      void Output();
12  };

14  int main() {
      Member mem;
16    cout << "Enter member number and fee: " << '\n';
      mem.Input();
18    cout << "The number and fee you entered: ";
      mem.Output();
20    cin.ignore();
      cin.get();
22  }

24  void Member::Input() {
      cin >> memberNum >> memberPay;
26  }
    void Member::Output() {
28    cout << memberNum << '\t' << memberPay;
    }
30
    /* Typical Output:
32  Enter member number and fee:
    10111
34  3.50
    The number and fee you entered: 10111    3.5
36  */
```

4. Fill in the blanks in each of the following statements:
   (a) A base class's protected members can be accessed only in the base-class definition or in derived-class definitions.
   (b) A base class's public members are accessible within that base class and anywhere that the program has a handle to an object of that base class or to an object of one of its derived classes.
   (c) A base class's protected access members have a level of protection between those of public and private access.

5. Define an `Array` class that expands the functionality of C++ arrays. Show how you would split your definition into an `Array.h` file and an `Array.cpp` file.

   The class should contain one data member that is a pointer to a dynamically allocated array of integers, and an integer that stores the current `size` of the array. The `size` integer should be `const` – it should not be changeable after the class constructor is called.

   The class should have 3 constructors: one that takes just a number of elements to allocate (initializing them all to 0), another that takes a number of elements and an array of initial elements, and a copy constructor that allocates a new array of the same size as the `Array` that is being copied, and then copies the elements one by one.

   Define 4 member functions: `getLength()` to return the number of elements in the `Array`; `getElement(int n)` to take an integer `n` and return a modifiable reference to the `(n+1)`st element of the array; another `getElement()` function that is declared `const` and returns a non-modifiable reference; and a `print()` function that takes a separator string and prints the elements one by one separated by the separator string. (For instance, `myArray.print("\n")` should print the array elements with newlines between each pair.)

   Also define a `destructor` to deallocate the memory that was allocated to the internal array when the `Array` object is destroyed.

   You do not need to define a main function that actually uses this class, but it will presumably be useful for testing purposes to try creating a few `Array` objects and manipulating them.

   SOLN: Here's my `Array.h`:

```
#pragma once
//Array.h
//-----------------------------------------------------

#include <iostream>
#include <string>
using namespace std;

class Array {
  const size_t sz;
  int* elem;
public:
public:
  Array(int numElements);
  Array(int initialValues[], int numElements);
  Array(Array &); // const Array &);
  int getLength() { return sz; }
  int &getElement(const int);
  const int &getElement(const int) const;
  void setElement(int,int);
  void print(const string &separator);
  ~Array();
};
// end Array.h
```

And the `Array.cpp` file:

```
//Array.cpp
#include "Array.h"
using namespace std;
Array::Array(int numElements) : sz(numElements) {
  elem = new int[sz];
  for (int i = 0; i < sz; i++)
    elem[i] = 0;
}
Array::Array(int init[], int numElements) : sz(numElements) {
  elem = new int[sz];
  for (int i = 0; i < sz; i++)
    elem[i] = init[i];
}
Array::Array(Array &a) : sz((size_t)a.getLength()) {
  //const Array &a) : sz((size_t)a.getLength()) {
  elem = new int[sz];
  for (int i = 0; i < sz; i++)
    elem[i] = a.elem[i];
}
int &Array::getElement(const int n) {
  return elem[n];
}
const int &Array::getElement(int n) const {
  return elem[n];
}
void Array::setElement(int n, int v) {
  elem[n] = v;
}
void Array::print(const string &separator) {
  cout << '{' << elem[0];
  for (int i = 1; i < sz; i++)
    cout << separator << elem[i];
  cout << '}';
}
Array::~Array() {
  delete[] elem;
};
```

To test this, I wrote:

```
//driver
#include "Array.h"

int main() {
  Array A();
  Array B(3);
  B.print(",");
  int a[3]{ 1,2,3 };
  cout << "\nC=";
  Array C(a,3);
  C.print(",");
  Array D(C);
  cout << "\nBefore setting D[2] to 4, D[2] = " << D.getElement(2);
  D.setElement(2, 4);
  cout << "\nAfter setting D[2] to 4, D[2] = " << D.getElement(2);
  cin.get();
}
```

Whose output is

```
{0,0,0}
C={1,2,3}
Before setting D[2] to 4, D[2] = 3
After setting D[2] to 4, D[2] = 4
```

Note: Problems 6,7,8 and 9 on the next page go together.

6. Write a function `createFile(ofstream& ofs, vector<string>);` to store text files and call it twice in a `main()` function to create two text files `file1.txt` and `file2.txt`. Store the following content in these two text files:

File 1:
```
The
is
the
```

File 2:
```
cat
in
hat.
```

Starter code:

```
1 void createFile(ofstream& ofs, vector<string> vs) {
    for (string s : vs) \\code here;
3   ofs.close();
}
```

Hints: Use `ofs << s.c_str()` where `ofs` is an output file stream and `s` is a string from the string library that needs to be converted to a c-string to be inserted into the output file stream.

7. Create another function, `mergeFiles(ifstream& ifs1, ifstream& ifs2, ofstream& ofs)`, to write to a third text file `file3.txt`, which will then contain the text of `file1.txt` merged with that of `file2.txt` in the following way:

```
The cat is in the hat.
```

To do this, create 3 separate objects–2 in input mode and one in output mode.
You will also have to use conditional statements to check for the ends of files.

8. Now create `addToFile(ofstream& ofs)` to allow the the user to append some text to the end of `file3.txt` existing text file. Get user input one line at a time and query the user after each entry as to whether or not they want to append more text. The console might look like this:

```
Enter a string to add to the file: But that's a dog's hat!
Do you want to add some more text? Y/N: y
Enter a string to add to the file: Which can only spell trouble...
Do you want to add some more text? Y/N: y
Enter a string to add to the file: with a capital 'T'.
Do you want to add some more text? Y/N: n
```

After which `file3.txt` would contain

```
The cat is in the hat. But that's a dog's hat! Which can only spell trouble...with a capital
'T'.
```

9. Finally, count and print:
   (a) The number of occurrences of a user-specified letter in your file `FILE3.TXT`.
   ```
   Enter a string to add to the file: But that's a dog's hat!
   Do you want to add some more text? Y/N: You can't get in that!
   Enter a string to add to the file: Do you want to add some more text? Y/N: n
   Enter a character to count in file3.txt: t
   There is/are 11 instance(s) of the character t in file3.txt.
   ```
   (b) The number of occurrences a user-specified string in your file `file3.txt` (You can use the `strcmp()` function of the `string` library here, or compare using string objects, with which you can use the `==` operator).

SOLN: Here it all is in one bundle:

```
#include <iostream>
using std::cin;
using std::cout;
#include <fstream>
using std::ifstream;
using std::ofstream;
using std::fstream;
using std::ios;
#include <vector>
using std::vector;
#include <string>
using std::string;

void createFile(ofstream& ofs, vector<string> vs) {
  for (string s : vs) ofs << s.c_str() << '\n';
  ofs.close();
}
void mergeFiles(ifstream& ifs1, ifstream& ifs2, ofstream& ofs) {
  string s1, s2;
  while (ifs1 >> s1 && ifs2 >> s2) {
    ofs << s1 << " " << s2 << " ";
  }
  ofs.close();
}
void addToFile(ofstream& ofs) {
  string s;
  ofs.close();
  ofs.open("file3.txt", std::ofstream::out | std::ofstream::app);
  char ch;
  do {
    cout << "Enter a string to add to the file: ";
    cin.clear();
    cin.ignore();
    getline(cin, s);
    ofs << s;
    cout << "Do you want to add some more text? Y/N: ";
    cin >> ch;
  } while (toupper(ch) != 'N');
  ofs.close();
}
int main() {
  vector<string> vs1{ "The", "is", "the" };
  vector<string> vs2{ "cat", "in", "hat." };
  ofstream ofs1("file1.txt"), ofs2("file2.txt");
  createFile(ofs1, vs1);
  createFile(ofs2, vs2);
  ifstream ifs1("file1.txt");
  ifstream ifs2("file2.txt");
  ofstream ofs3("file3.txt");
  mergeFiles(ifs1, ifs2, ofs3);
  addToFile(ofs3);
  ofs3.close();
  ifs1.close();
  ifs1.open("file3.txt");
  char target, ch{ 'a' };
  unsigned count{ 0 };
  cout << "\nEnter a character to count in file3.txt: ";
  cin >> target;
  while (ifs1 >> ch) {
    if (ch == target) count++;
  }
```

```
62    cout << "\nThere is/are " << count << " instance(s) of the character " << target <<
        " in file3.txt.";
      ifs1.close();
64    ifs1.open("file3.txt");
      string starget, s;
66    count = 0;
      cout << "\nEnter a string to count in file3.txt: ";
68    cin >> starget;
      while (ifs1 >> s) {
70      if (s == starget) count++;
      }
72    cout << "\nThere is/are " << count << " instance(s) of the string " << starget << "
        in file3.txt.";
      ifs1.close();
74    cin.ignore();
      cin.get();
76  }
```