Instructions:

- You have until 11:50 to complete this test.
- You may use nothing to aid you, not even a calculator.
- Put away ALL papers, books, and electronic devices.
- There are no syntax errors anywhere in the code on this exam.
- Please state clearly any assumptions that you have to make in interpreting a question.
- Please create an alias above that you can remember but that no one else could use to guess who you are. We will use it to post grades on the course website. Keep it clean. If you don’t create one, we will make up a random number.
1. **(20 points)** Create an Appointment class that has three private member variables: one representing the day of the appointment, one representing the hour of the appointment, and one describing the appointment. To keep things simple, for the day simply represent it as a Julian day (1..365), for the hour simply represent it as an integer in the range 0..23 (military time) rather than worrying about am or pm, and for the appointment just use a string.

(a) Give the class declaration, including the member variables and the prototypes for the following functions: (i) a constructor from a Julian day, an hour and a description, (ii) a copy constructor, (iii) an operator== as a member function, and (iv) a friend function operator<<. (Of course, more functions than these would be needed, but this is enough for this problem.)

**Solution:**

```cpp
class Appointment {
public:
    Appointment( int d, int h, string des );
    Appointment( const Appointment& old );
    bool operator==( const Appointment& other );
    friend ostream& operator<<( ostream& ostr, const Appointment& appt );
private:
    int day;
    int hour;
    string description;
};
```
(b) Give the function definitions (the implementations) for the four functions you declared in part (a). You can assume that the values provided to the constructor are in the proper ranges.

Solution:

    Appointment::Appointment( int d, int h, string des )
    : day(d), hour(h), description(des)
    {} 

    Appointment::Appointment( const Appointment& old )
    : day(old.day), hour(old.hour), description(old.description)
    {} 

    bool Appointment::operator==( const Appointment& other )
    {
        return day == other.day && hour == other.hour && description == other.description;
    } 

    ostream& operator<<( ostream& ostr, const Appointment& appt )
    {
        ostr << "Day: " << appt.day << ", Hour: " << appt.hour
        << ", Description: " << appt.description;
        return ostr;
    }
2. (6 points) In one sentence each, define

(a) **Deep copying**
   
   **Solution:** Copying the logical structure of an object to create an individual, but identical copy rather than a superficial copy that involves pointer.

(b) **Abstract data type:**
   
   **Solution:** A data type or object specified only but the operations that can be applied to it and not by its implementation

3. (9 points) Given an array of integers, \texttt{intarray}, and a number of array elements, \texttt{n}, write a short code segment that uses **pointer arithmetic and dereferencing** to add every second entry in the array. For example, when \texttt{intarray} is

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 16 & 4 & -3 & 2 & 76 & 9 & 3 & 6
\end{array}
\]

and \texttt{n==9}, the segment should add $1 + 4 + 2 + 9 + 6$ to get 22. Store the result in a variable called \texttt{sum}.

**Solution:**

\[
\texttt{sum} = 0;
\]

\[
\texttt{for ( int \ast p = \texttt{intarray}; p < \texttt{intarray+n}; \ast p+=2 )} \\
\texttt{\hspace{1cm} \texttt{sum += \ast p;}}
\]
4. (10 points) Clearly show the output from the following code segment.

```cpp
int x = 45;
int y = 30;
int *p = &x;
*p = 20;
cout << "a: x = " << x << endl;

int *q = &y;
int temp = *p;
*p = *q;
*q = temp;
cout << "b: x = " << x << " , y = " << y << endl;

int * r = p;
p = q;
q = r;
cout << "c: *p = " << *p << " , *q = " << *q << endl;
cout << "d: x = " << x << " , y = " << y << endl;
```

Solution:

a: x = 20
b: x = 30, y = 20
c: *p = 20, *q = 30
d: x = 30, y = 20
5. **(20 points)** Write a program that counts and outputs the number of sentences in a text file. Part of the program is given below; you need to write the rest. Count one sentence when the program reads a small letter and then reads a `. ' This prevents counting decimal points that appear in numbers as ends of sentences. It also prevents counting periods that follow initials as a sentence. As an example, three sentences should be counted in the following two lines of text.

George W. Bush went to Washington. 3.14159 is an approximation to pi. But.....

Hints: remember, the `ifstream` class has a member function with the prototype

```cpp
bool ifstream::get( char& c );
```

that reads the next char and returns false if the end of the file has been reached. Also, remember the `cctype` library contains a function `bool islower(char c)` that returns `true` if and only if the character is a lower-case letter.

```cpp
#include <iostream>
#include <fstream>
#include <cctype>
using namespace std;

int main( int argc, char* argv[] )
{
    if ( argc != 2 ) {
        cerr << "Usage: " << argv[0] << " text_file\n";
        return 0;
    }

    ifstream text_str( argv[1] );
    if ( !text_str ) {
        cerr << "Can’t open " << argv[1] << "\n";
        return 0;
    }

    Solution:

    char c, prev_c = ' ';
    int count
    while ( text_str.get( c ) )
    {
        if ( c == '.' && islower( prev_c ) )
            ++ count;
        prev_c = c;
    }
    cout << "Sentence count = " << count << endl;
    text_str.close();
}
6. (15 points) Write a `String` class member function that creates a new string from the current string and has the same characters but in reverse order. The function prototype is

    String String::reverse( ) const;

Recall that `String` class objects have three member variables:

- `char * sPtr;`  // pointer to the array
- `int len;`  // the number of chars (not including '\0') in the string
- `int alloc;`  // the size of the array currently allocated

Solution:

    String String::reverse( ) const
    {
        String new_string;
        delete [] new_string.sPtr;
        new_string.len = len;
        new_string.alloc = alloc;
        new_string.sptr = new char[ alloc ];
        for ( int i=0; i<len; ++i )
            new_string.sptr[i] = sptr[ len-1-i ];
        new_string.sptr[ len ] = '\0';
    }
7. **(20 points)** Write a function that takes an array of floating point numbers and copies its values into two new arrays that must be allocated in the function, one containing only the negative numbers from the original array, and the other containing the non-negative numbers from the original array. For example, if the original array is

```
0 1 2 3 4 5 6 7 8
-1.3 5.2 8.7 0.0 -4.5 7.8 -9.1 3.5 6.6
```

Then the resulting array containing the negative values would be

```
0 1 2
-1.3 -4.5 -9.1
```

and the resulting array containing the non-negative values would be

```
0 1 2 3 4 5
5.2 8.7 0.0 7.8 3.5 6.6
```

(a) Start by writing the function prototype. Think about what parameters you need, what their types should be, and how they should be passed.

**Solution**

```c
void split_array( float floatarr[], int n,  
                 float * & negatives, int & neg_count,  
                 float * & positives, int & pos_count )
```

(b) Now write the code of the actual function. You do NOT need to write the prototype over again. Do not allocate any more space for the new arrays than is necessary. You may use the back of this page or the back of the preceding page if you need more room.

**Solution:**

```c
{
  int i;
  neg_count = 0;
  for ( i=0; i<n; ++i )
    if ( floatarr[ i ] < 0 )
      neg_count ++ ;
  pos_count = n - neg_count;

  negatives = new float[ neg_count ];
  positives = new float[ pos_count ];
  int ni = 0, pi = 0;
  for ( i=0; i<n; ++i )
    if ( floatarr[ i ] < 0 )
      {
        negatives[ ni ] = floatarr[ i ];
        ++ ni;
      }
    else
      {
        positives[ pi ] = floatarr[ i ];
        pi ++ ;
      }
```