Assignment 6

100 points

Purpose

The purpose of this assignment is to give you some experience in writing simple classes, using C++ `vector`s and in writing multi-part programs.

Assignment

Write a program to display a small genealogical database.

The program is invoked with a command line argument giving the name of a data file containing simplified genealogical data in a custom data format.

The data file is opened and all the records in the file are read into an array. A three generation pedigree chart is displayed using one of the entries in the array as the root position in the chart. The user is given the option to follow the genealogical line of either the father, mother or child of the person in the root position, if such information exists in the array. After choosing one of these options, a new chart is printed. In this way, the user can traverse the stored family tree until deciding to quit.

Program

You will need to write several functions and several classes for this assignment. All data members in the classes are `private`. All methods are `public`. Each class should be implemented in two parts: 1) a header file, complete with header guards, containing the class definition; 2) a source code file containing the definitions of the member functions for that class.

Each class should have a default constructor (one that takes no arguments) declared, along with any others mentioned below.

`class textCanvas`

The sole purpose of this class is to provide a means of flexibly formatting output on the screen. Through the use of a `textCanvas`, routines can be created to simulate moving the cursor to an arbitrary (x, y) position on the screen.

This class contains a 2D array of characters of size `CHEIGHT` rows and `CWIDTH` columns. These two symbolic constants should be declared as global symbolic constants using the `const int` notation. In deference to the default terminal size, these symbolic constants should have values of 24 and 80 respectively. Because these symbolic constants are related to the `textCanvas` class, they should go in the header file for this class. (But they are *not* members of the class.)

The class also contains integer fields specifying the current row and column position of the cursor in this `textCanvas` for printing. Four more integer fields are also needed to hold the left, right, top and bottom bounds of a box that can be set for printing.

The global symbolic constants `TC_OK`, `TC_OUT_OF_BOUNDS`, and `TC_CURSOR_OUT_OF_BOX` should also be defined in the header file for this class.

Coordinates in the `textCanvas` start at (0, 0) in the upper left hand corner and extend in the positive x direction to the right and in the positive y direction downward. This corresponds naturally to row indices in the 2D array increasing downward and column indices increasing from left to right. The class should have the following methods:

`textCanvas` constructor

The only constructor in this class should take no arguments. It should clear the array to all spaces, set the current cursor position to (0, 0), and set the bounding box boundaries to the edges of the canvas.

`textCanvas::clear()`

This method returns nothing and takes no arguments. It clears out the 2D array of characters in the `textCanvas` to spaces and sets the coordinates of the cursor position to (0, 0). In addition, the left and top box bounds are set to 0, the right box bound is set to CWIDTH, and the bottom box bound is set to CHEIGHT.

`textCanvas::print()`

This method returns nothing and takes no arguments. It simply prints out the 2D array of characters in the current `textCanvas` instance to the standard output. This method changes nothing in the current instance.

`textCanvas::set_box()`

This method returns an int and takes as arguments four integers, which specify the x and y positions of the top corner of a bounding box and the width and height of the box. These values are used to compute and set the bounding box of the current instance.

The purpose of this routine is to limit the printing of the `print_char()` and `print_string()` methods below to print only within the bounding box specified.

The bounds are taken as going from the left and top boundaries up to but not including the right and bottom boundaries. For example a bounding box starting at (10, 10) with a width and height of 40 and 7 respectively, would have a left bound of 10, a top bound of 10, a right bound of 50, and a bottom bound of 17.

If any of the specified bounds are outside the natural bounds of the 2D array as determined by 0, `CWIDTH`, and `CHEIGHT`, this function should return `TC_OUT_OF_BOUNDS`, else it should return `TC_OK`.

This routine should also reset the cursor row and column to the left and top bounds of the new bounding box.

`textCanvas::print_horizontal_line()`

This method returns an integer and takes three integers as arguments. The first two integers specify the x and y coordinates of the starting position of the horizontal line and the third integer specifies the length of the horizontal line. The method places characters in the array starting at the given x and y coordinates and in successive x coordinate positions until a horizontal line of the specified length is drawn. The line is assumed to extend to the right from the given starting point. You may choose what you feel is the most appropriate character for drawing horizontal lines.

If the starting or ending bounds of the line fall outside the array boundaries, this function should return `TC_OUT_OF_BOUNDS`. Otherwise it should return `TC_OK`.

`textCanvas::print_vertical_line()`

Similar to the previous method, this one takes three integers specifying the x and y starting points of the line and the height of the line. The line extends downward from the starting point. Again, you may choose the character to be used for drawing the vertical line.

`textCanvas::print_char()`

This method returns an integer and takes a constant character. If the character is a `'\n'`, then the function should increment the cursor row in the current instance and reset the cursor column to the value of the left box boundary. If the character is a `'\r'`, then the function should simply reset the cursor column to the value of the left box boundary for the current instance.

For any other character, if the current cursor position is within the box boundaries, then the character should be placed in the corresponding location in the canvas array and the current cursor column incremented. The function should return `TC_OK` in this case. If the current cursor position is not within the box boundaries then return `TC_CURSOR_OUT_OF_BOX`.

Note that the test for whether or not a character is in or out of the bounding box should take place only if the character is not a new line character or a carriage return character. This is because the cursor can be out of the bounding box and placed back in by printing a single new line character.

Hint: Use a `switch` to implement this function.

`textCanvas::print_string()`

This function returns an integer and takes as argument a C++ style string. This function simply takes the characters of the string and prints them to the canvas using the `print_char()` method described above. This function should return the value returned by the last call to `print_char()`.

`class Date`

This class contains three data members, which are all C++ strings. The first is a two digit day. The second string is a three character abbreviation for the month. The third string is a four digit year.

`Date` constructor

The default constructor should initialize each of the member strings by placing a null terminating character in the beginning position of each array.

`Date` constructor

There should also be a constructor which takes three strings, which are used to initialize the day, month, and year, respecitvely.

`Date::print()`

This method returns no value and takes a`textCanvas` as an argument. This method makes no changes to the instance of the class. This routine should print out the day, month, and year from the current instance on a single line to the `textCanvas`. Any additional formatting is up to you. Use the `print_string()` method of the `textCanvas` class.

`class Name`

This class contains only one data member, an C++ string;

`Name` constructor

The constructor should initialize the member string to the empty string;

`Name` constructor

There should also be a constructor that takes a C++ string and initializes the internal data member.

`Name::print()`

This function returns no value, takes a `textCanvas` as an argument, and makes no changes to the current instance. This routine should print out the string in the data member to the `textCanvas`.

`class Place`

This class contains three data members which are all of the type `Name` as defined above. The data members represent three levels associated with place names. In the United States, these three levels are most often associated with city, county and state, respectively.

`Place` constructor

The default constructor for this class should initialize the three internal string appropriately.

`Place` constructor

There should also be a constructor which takes three strings, which are used to initialize the three internal data members.

`Place::print()`

This function returns no value, takes a reference to a `textCanvas` and makes no changes to the current instance. It prints the three levels of the place on a single line, separated by commas, to the `textCanvas`.

`class Person`

This class contains nine data members. The first two are `Name`s, representing the last and first names of the person. The next member is a `Date` giving the birthdate. This is followed by a `Place` member for the birthplace. The next two members are another `Date` and `Place` respectively, for the date and place of death. The last three members are integers, indicating locations of relatives in an `vector` of `Person`s. These three integer members should be named (in order) `father`, `mother`, and `child`.

`Person` constructor

The constructor for this class should initialize the father, mother, and child fields to -1. All other members are initialized automatically using the default constructor for those classes.

`Person` constructor

The class should contain another constructor which takes values for all of the internal data members and sets them accordingly. I.e., it takes two `Name`s, a birth `Date` and `Place`, a death `Date` and `Place` and three `int`s for the father, mother, and child links.

`Person::get...()`

There should be three accessor methods, `get_father()`, `get_mother`, and `get_child`, each of which return an int, take no arguments, and do not modify the current instance of the class. Their only purpose is to return the value of the corresponding member in the `Person`.

`Person::print()`

This function returns no value, takes as argument a reference to a `textCanvas`, and does not modify the current instance of the class. The last name and first name should be printed, separated by a comma. On subsequent lines should be placed the birthdate, birthplace, death date and death place. Any additional formatting, especially that marking the birth and death information, should be done here.

`main()`

The `main` function should be placed in the file `assign6.cc`. The `main()` function for this assignment should take a command line argument, the name of the data file containing the records. This argument will be found in argv[1]. If the number of arguments is wrong, the program should print a brief error message and exit.

The `main()` function should contain a `vector` of `Person`s.

The filename given on the command line should be opened as a file stream for input. If the file can't be opened, the program should print an error message and exit.

The program should read records from the file into the `vector` The `vector` should be filled until there is no more data in the file. The format of the data file is described below. After reading, the file stream should then be closed.

The program should have an integer variable indicating the root position in the pedigree to be printed. This variable should be initialized to zero.

In a loop, the program should print the pedigree anchored at the root position using the `print_pedigree` function given below, and then ask the user for a command option to move to either the father, mother, child, or to quit.

The proper processing of the commands depends on the `father`, `mother`, and `child` fields of the current root person in the `vector`. The value of these fields indicates the index position of that person in the `vector`. A value of -1 indicates that no such person exists.

Therefore, if the user wants to move to the father, if the father field of the current person does not equal -1, then the root index can take on the new value of the father field of the current person.

Similar logic holds for moving to the mother or child of the current root individual. If the desired person does not exist, the command is effectively ignored. The next iteration of the loop prints the pedigree chart using the same root index as before.

Of course, if the user decides to quit, then the program should exit.

Use a `switch` statement to process the user command. Don't forget to process both upper and lower case versions of the command.

`print_pedigree()`

This routine should stay in the same source code file as the `main()` function. This routine has some subtle potential problems that would make this assignment far too large to deal with here. So here is the code for this function. Enjoy.

```
void print_pedigree(textCanvas& tc, const vector<Person> & person, int root)
{

  tc.clear();

  // Print tree diagram
```

```
const int boxwidth = CWIDTH / 3;     // Three generations
const int boxheight = CHEIGHT / 4;   // Four ancestors in third generation

const int root_x = 0;
const int root_y = int(1.5 * boxheight);

const int father_x = boxwidth;
const int father_y = int(0.5 * boxheight);

const int mother_x = boxwidth;
const int mother_y = int(2.5 * boxheight);

const int pgf_x = 2 * boxwidth;  // Paternal grandfather
const int pgf_y = 0;

const int pgm_x = 2 * boxwidth;  // Paternal grandmother
const int pgm_y = boxheight;

const int mgf_x = 2 * boxwidth;  // Maternal grandfather
const int mgf_y = 2 * boxheight;

const int mgm_x = 2 * boxwidth;  // Maternal grandmother
const int mgm_y = 3 * boxheight;


// Line for root
tc.print_horizontal_line(root_x, root_y, boxwidth);

// Horizontal line for father
tc.print_horizontal_line(father_x, father_y, boxwidth);

// Horizontal line for mother
tc.print_horizontal_line(mother_x, mother_y, boxwidth);

// Horizontal lines for grandparents
tc.print_horizontal_line(pgf_x, pgf_y, boxwidth);
tc.print_horizontal_line(pgm_x, pgm_y, boxwidth);
tc.print_horizontal_line(mgf_x, mgf_y, boxwidth);
tc.print_horizontal_line(mgm_x, mgm_y, boxwidth);

// Vertical lines
tc.print_vertical_line(father_x, father_y, 2 * boxheight);
tc.print_vertical_line(pgf_x, pgf_y, boxheight);
tc.print_vertical_line(mgf_x, mgf_y, boxheight);

// Print the pedigree
int person_id = root;
tc.set_box(root_x, root_y, boxwidth, boxheight);
person[person_id].print(tc);

// Print father's line
if(person[person_id].get_father() != -1){
  person_id = person[root].get_father();
  tc.set_box(father_x, father_y, boxwidth, boxheight);
  person[person_id].print(tc);

  if(person[person_id].get_father() != -1){
    tc.set_box(pgf_x, pgf_y, boxwidth, boxheight);
    person[person[person_id].get_father()].print(tc);
    }
    if(person[person_id].get_mother() != -1){
      tc.set_box(pgm_x, pgm_y, boxwidth, boxheight);
      person[person[person_id].get_mother()].print(tc);
    }
  }

  // Print mother's line
  if(person[root].get_mother() != -1){
    person_id = person[root].get_mother();
    tc.set_box(mother_x, mother_y, boxwidth, boxheight);
    person[person_id].print(tc);

    if(person[person_id].get_father() != -1){
      tc.set_box(mgf_x, mgf_y, boxwidth, boxheight);
      person[person[person_id].get_father()].print(tc);
    }
    if(person[person_id].get_mother() != -1){
      tc.set_box(mgm_x, mgm_y, boxwidth, boxheight);
      person[person[person_id].get_mother()].print(tc);
    }
  }

  tc.print();
}
```

Input

The data file format for each `Person` in the file is specified with each field on a separate line. The order of the lines is extremely important. The data fields (in order) are:

1. Last name
2. First name
3. Birth date - day
4. Birth date - month
5. Birth date - year
6. Birth place - city
7. Birth place - county
8. Birth place - state/country
9. Death date - day
10. Death date - month
11. Death date - year
12. Death place - city
13. Death place - county
14. Death place - state/country
15. Father index
16. Mother index
17. Child index

You should use the `getline(istream&, string&)` function to input lines of the file. The function `int stoi(const string& )` will be useful in converting strings to integer values.

There are many blank lines in the file. This is correct and indicates unknown values for names, dates, and places.

Output

A sample screen for the output appears below:

```
                                                    Lincoln, Abraham----------
                                                    b. 13 May, 1744
                                                       , Berks, Pennsylvania
                          Lincoln, Thomas-----------d.  May, 1786
                          b. 6 Jan, 1776               Hughes Station, Jeffers
                             Linville Creek, Rocking|
                          d. 17 Jan, 1851           Herrin(g), Bersheba-------
                             Farmington, Coles, IL  b.  ,
                          |                            , ,
Lincoln, Abraham----------|                         d.  , 1836
b. 12 Feb, 1809           |                            , ,
   , Hardin, KY           |
d. 15 Apr, 1865           |                         Hanks, James--------------
   Washington D.C, ,      |                         b.  ,
                          |                            , ,
                          Hanks, Nancy--------------d.  ,
                          b. 5 Feb, 1784               , ,
                             Brookneal, Campbell, VA|
                          d. 5 Oct, 1818            Shipley, Lucy-------------
                             Little Pigeon Creek, Spb.  ,
                                                       , ,
                                                    d.  ,
                                                       , ,

Move to F)ather, M)other, C)hild or Q)uit?
```

Other Points

You must have a complete Makefile, with a separate rule for compiling a separate object file for the code from each class. The linking rule should link all of your object files together to create one executable.

The name of your executable should be `assign6`.

Don't forget to get rid of all compiler warnings when the `-Wall` compilation option is used.

As always, programs that do not compile on `turing/hopper` automatically receive 0 points.

Submit your program using the electronic submission guidelines posted on the course web site and discussed in class.

Implementation Hints

One of the minor objectives of this course is to get you out of the habit of writing the entire assignment all at once and then trying to debug it.

When faced with a project of this size, implement it a piece at a time. Keep the program functioning at every stage of development even though the functioning is wrong.

In the opinion of the instructor, the best way to implement this assignment is to start with the input. Create an implementation of `Person` that has internal data members of `string`s instead of the `Name`s, `Date`s, and `Place`s. Make a constructor of all strings. Make a print method that prints the `Person` to the standard output. (Methods that are not part of the official assignment should be removed before submitting.)

Input the data for a single `Person`s into a `Person`. Print the `Person`. Input the data for all of the `Person`s into the vector. Print the vector.

Then add the `Name` class and its constructors. Implement the `print()` method to take no arguments and print to the standard output. Change the appropriate data members and constructor of `Person` as needed to work with `Name`. Then do the same with `Date`. Then `Place`.

At each stage, verify that the data is properly input and can be printed out. The importance of having the program work at the end of each stage can not be overstated. If you don't, you will likely spend much more time rewriting code than you otherwise would.

Finally, implement the `textCanvas` class. Modify the `print()` methods of the other classes to use the `textCanvas` class. Add the `print_pedigree` function and use it.

Then work in the functionality to print the pedigree and query the user for further action.

At the end, you should have a main source code file containing `main()` and `print_pedigree()` and separate header and source code files for each class.

Other Points

- You should have a source code file for `main()` and all the non-member functions. You should have a source code file and header file for each class mentioned in the assignment.
- Although they may be short, none of your class methods should be inline.
- No accessor methods should be written for any of the classes unless otherwise specified.
- Header files are required to have header guards.
- Use `const` as appropriate, both on methods and function arguments.
- Use references appropriately.
- Of course, a `Makefile` is required for this assignment.
- Symbolic constants should be used to avoid magic numbers. You should use `const` to make your symbolic constants.
- The name of your source code file containing main should be `assign6.cc`.
- Function prototypes are required for all functions you write (except `main()` of course)
- Programs that do not compile on `turing/hopper` automatically receive 0 points.
- Submit your program using the electronic submission guidelines.