Program 1: Control, Classes, and CDs
CPS 100E, Fall 1995

Owen Astrachan Susan Rodger

Due: Part I, Friday September 15; Part II, Monday September 18

Introduction

The files for this assignment can be found in `~ola/cps100e/prog1`. There are many files, they are itemized below and a listing of the header files is attached as an appendix to this document as are some of the implementations (.cc files).

Makefile, roulette.cc, cdlist.h cdlist.cc usedb.cc

You should create a directory `cps100e` in which all work for this class will be done (you'll do this in the first lab). Each assignment should be done in its own subdirectory, for this assignment create a directory `prog1`. All work for your assignment will then be done in the directory `~/cps100e/prog1`.

You should copy the files for this assignment into your `prog1` directory. To do this, `cd` into the `prog1` directory you created and type `cp ~ola/cps100e/prog1/* .` (don't forgot the final '.').

Outline of Assignment

There are two parts to this assignment. The first part is due in roughly two weeks, the second part in two-and-a-half weeks. You should be thinking about and working on the second part while you're doing the first part. The first part has two subparts. These are outlined below; particulars can be found in the sections that follow.

Part I
1. For the first part of Part I, you must write a program *totem.cc* that creates random totem poles (heads with different features).
2. For the second part of Part I, you must write a program *roulette.cc* that can be used to simulate a game of roulette.

Part II For this part you will be implementing a program that permits a user to keep track of a database of CDs (compact disks). You'll also be modifying a class declaration and implementing member functions.

1 Part I

1.1 Random Totem Poles

This program is worth 5 points. Points will be allocated as follows: 3 for correctly implementing the program, 2 for the style in which the program is written.

Using the sketch program outlined on page 158-159 of the red course pack as a point of reference, write a program named *totem.cc* that draws a "random" totem-pole of faces. The height of the pole, measured in faces, should be specified by the user. For example, a sample run below shows a totem-pole of height 2.

output

```
prompt> totem
how many faces: 2
  ||||||/////////
  |              |
  |              |
  |    0    0    |
  |              |
 _|              |_
|_                _|
  |   --------   |
  |              |
  |||||||||||||||
  |              |
  |              |
  |    .    .    |
  |              |
 _|              |_
|_                _|
  |   |______|   |
  |              |
```

Each "head" in the totem pole should be random in the sense that their must be at least two parts to a head, and these should be chosen at random. For example, the function below shows one example of random hair. (Assuming a function `Sides()` exists, see `~ola/cps100e/progs/parts.cc`. All code from the book is in the `~ola/cps100e/progs` directory and you can copy any of these files.)

```
void
Hair()
// postcondition: prints random hair
{
    Dice d(3);
    int choice = d.Roll();
    if (1 == choice){
        cout << "  ||||||///////// " << endl;
    }
    else if (2 == choice){
        cout << "  |||||||||||||||  " << endl;
    }
    else if (3 == choice){
        cout << "  |______________|  " << endl;
    }
    Sides();
}
```

Be sure that you use appropriate `Dice` objects, i.e., if there are three different hair styles a three sided die should be used. You should have at least two different parts to a head (e.g., hair, eyes, nose, mouth) and, for each part, at least three choices.

1.2 Roulette

This program is worth 10 points. Points will be allocated as follows: 6 for correctly implementing the program, 4 for the style in which the program is written. In particular, functions should be used "appropriately" for maximal credit.

Write a program named *roulette.cc* that simulates a game of roulette. In roulette a wheel spins and yields a number between 1 and 38 when a ball drops into a slot (randomly). There are also numbers 0 and 00. For the purposes of this program, assume that all odd numbers are black, all even numbers are red, and that both 0 and 00 are green[1].

In your program the user should be prompted for an initial amount of money, the user's bankroll. The user should then be given the opportunity to make one of three different bets, or to quit the program.

- bet on red or black
- bet on high or low (low is 1 – 19, high is 20 – 38)
- bet on a specific number

The first two bets pay even money: a bet of \$5.00 earns \$5.00 if the bet wins. Winning a bet on a specific number pays $38 \times b$ where b is the bet. Thus a winning bet of \$4.00 earns \$152.00. A winning bet earns back the money bet so that betting \$10.00 on red and winning means that you get back \$20.00 — the original wager plus the even money winnings, or $10 + 10 = 20$.

The user should be allowed to bet any amount not exceeding the bankroll. After each bet, the user should be told what the current bankroll is.

You'll want to make good use of functions in this program or writing it will be very complicated. For extra credit you can write a Roulette wheel or game class, but this isn't necessary. You should try to keep the code in `main` to a minimum. Part of a completed program is supplied at the end of this handout (and available in `~ola/cps100e/prog1/roulette.cc`), but you do NOT have to follow this. In particular, you can earn maximal points by using an enumerated type, or enum, instead of a string for the variable `betKind` in *roulette.cc*.

1.3 Submitting Part I

You should create a `README` file for this and all assignments. All `README` files should include your name as well as the name(s) of anyone with whom you collaborated on the assignment and the amount of time you spent. You should also record any comments you have about the assignment. For part I of the assignment submit as follows:

```
submit100e prog1.1 README totem.cc roulette.cc
```

You can submit by typing `make submit1` if the correct README file is in the directory from which you submit. You can always edit the Makefile to add or change filenames. For example, if you have a class in the roulette program, you can include the declaration and implementation in the same file but you may want to experiment with multiple files (remember, a class is extra credit and not required).

[1]In the real game the colors black and red are distributed differently, but the 0 and 00 are green.

2 Part II:

In this program you will modify a *client* program, *usedb.cc* and a program that implements a list of CDs (and serves as a database), *cdlist.cc*. The class `CDlist` implemented in *cdlist.cc* is declared in *cdlist.h*.

The program *usedb.cc* is barely functional. It does implement some operations, but because the `CDlist` class isn't fully functional, and because the function `DoRead` in *usedb.cc* isn't implemented, you can't really use the program although you can run it.

When implementing different parts of the program, you should probably test the program on a small database as well as the large (3900+ CD's) database provided. You'll also need to test with a database that has a small capacity to ensure that your program handles a "full" database properly.

2.1 Getting Started

You are to implement several functions in *usedb.cc* and member functions in *cdlist.cc*. In order to have some data to work with, you'll need to implement the `DoRead` function in *usedb.cc*.

2.2 Implementing `DoRead`

In order to implement `DoRead` you'll also need to implement the parameterized constructor `void CDlist(int size)` described in the header file *cdlist.h*.

To implement `DoRead` you'll need to write code to read information from the file specified by the user, and repeatedly call the *Add* member function to add each CD whose info is stored in the file. The data stored in *cd.dat* is in the format below where colons delimit each item and a less-than sign < follows the title.

$$\textit{ID number} : \textit{price} : \textit{artist/group} : \textit{title} <$$

Part of the file *cd.dat* is reproduced below.

```
100121 : 15.98 : R.E.M. : Automatic for the People<
100122 : 14.98 : Happy Mondays : Yes, Please<
100126 : 14.98 : 10,000 Maniacs : Our Time In Eden<
100127 : 11.98 : Skid Row : B-Side Ourselves<
100128 : 14.98 : Brubeck, Dave Quartet : Once When I Was Very Young<
100130 : 15.98 : Great White : Psycho City<
100132 : 14.98 : Morgan, Frank : You Must Believe In Spring<
100133 : 15.98 : Every Mother's Nightmare : Wake Up Screaming<
```

You can test `DoRead` by reading from a file and selecting the print option.

All input from cin should be done using `getline` rather than using >>. Do NOT mix these two methods of getting input without being very careful.

2.3 Other DB operations

You must implement the following options in *usedb.cc*:

- DoAdd
- DoGroup (currently no prototype, searches for group matches)
- DoDelete
- DoIDnumber (currently no prototype, searches for given ID)

The function `DoIDnumber` should search for a specific ID number as specified by the user. To implement this you should use the `CDlist` member functions `Reset`, `Next`, `IsDone`, and `Current`; you should NOT write new member functions to implement this option. You'll need to add a character and string entry to the arrays of choices at the top of the program for this option (use 'i' for id-number).

The function `DoDelete` should be added (along with a string and character so that it is displayed in the menu of choices). Use 'd' for delete. The "work" of deleting is done by requesting the id number of the CD to delete from the user, then calling `list.Delete(id_num)`. To have this work you'll need to implement the member function `Delete` which removes the item with the specified ID number by shifting all items one place "to the left". You'll need to decrement the count of CD's too.

If you can think of other operations you'd like to add, do so.

2.4 Other Member functions

In order for `DoGroup` and `DoTitle` to work you'll need to implement the member functions `TitleSearch` and `ArtistSearch`. These should look for partial matches as described in the header file *cdlist.h* and below. Searching for the artist "Jimmy", for example, would match Jimmy Webb, Jimmy Buffett, Jimmy Cliff, Jimmy Page, and Jimmy Durante (at least). All the CD's for these entries would be added to the parameter `match`.

To facilitate matches you can use three `String` functions: `Contains`, `Downcase`, and `substr`.

The prototype for `Contains` is given below.

```
int Contains(const String & s);
// postcondition: returns 1 (true) iff string contains s as substring
```

It is used as follows:

```
string s = "daydream";
if (s.Contains("dream")) cout << "yes 1" << endl;
if (s.Contains("ayd"))   cout << "yes 2" << endl;
if (s.Contains("Day"))   cout << "yes 3" << endl;
```

The first two statements evaluate to true (1), the third call of `Contains` returns 0 (false) since `s` does not contain `"Day"`.

The member function `Downcase` returns a lowercase version of a string. The statements below print `"hello"`.

```
string s = "HeLLo";
cout << s.Downcase() << endl;
```

The prototype is `void string Downcase();`. Note that this function **DOES NOT** change the string it is applied to, but returns another string that is a lower-case version.

The function `substr` has prototype `void substr(int pos, int len)`, it returns a string of `len` characters starting with character `pos` (recall that the first character has index 0).

For example:

```
string s = "bloodstream";
cout << s.substr(0,5) << endl;
cout << s.substr(5,6) << endl;
cout << s.substr(2,3) << endl;
```

would print, on three separate lines, "blood", "stream", and "ood". If the parameter `len` is too big, only as many characters as are in the string are taken. If `pos` is too small or too big, the empty string is returned.

2.5 Submitting Part II

You should submit all programs that you modified as part of this assignment. This should include *usedb.cc*, *cdlist.cc*, and *cdlist.h*. As always, submit a *README* file indicating about how long it took for this assignment, who (if anyone) you worked with, and any comments you have about the assignment.

Submit using

```
   submit100e prog1.2 README cdlist.h cdlist.cc usedb.cc
README sequence.h sequence.cc seqiterator.h
             seqiterator.cc prog1.h prog1.cc
```

You can do this by typing `make submit2` assuming all files are in the same directory.