CS 2413 Data Structures – Spring 2015 – Programming Project 1
Due January 28, 2015 – 11:59 PM

Objectives

1. [10 pts] Use basic input/output in C++ (cin and cout).
2. [10 pts] Use arrays in C/C++.
3. [10 pts] Create at least 2 C++ classes.
4. [50 pts] Design and implement the program *according to the project description*. 50 pts are distributed as follows:
   a. [10 pts] Read the input file using *redirected input*; print out each line read from the file.
   b. [10 pts] Add users to the UserDB data structure (adduser).
   c. [10 pts] Print out the information of a specific user in UserDB (finger).
   d. [10 pts] Print out the list of users in UserDB (showUsers).
   e. [10 pts] Generate (print out) the passwd file using the data stored in UserDB (showPasswd).
5. [20 pts] Document your project thoroughly as the examples in the textbook. This includes but not limited to header comments for all classes/methods, explanatory comments for each section of code, meaningful variable and method names, and consistent indentation.

Project Description

The file **/etc/passwd** is a text file used to store account information on UNIX-like operation systems. Each line represents a record of a user account in the system and it has the following format:

```
jsmith:x:1001:1000:Joe Smith,Room 1007,(234)555-8910,(234)555-0044,email:/home/jsmith:/bin/sh
  1    2   3    4                                5                          6          7 (field)
```

As shown above, a record contains 7 fields, which are separated by colons (:). These fields are (1) user login name (2) password (3) user identifier (4) primary group identifier (5) Gecos field (6) user's home directory (7) user's shell.

In this project, you will write a C++ program that allows the users to add, delete, modify or print the users in the system and also be able to print out the user database in this format. You will implement the C++ classes as described in this section and as well as the `main()` method which will read and execute the commands from standard I/O.

AccountInfo Class

The AccountInfo class is used to store the account information of the users, it is defined as follows:

```
class AccountInfo
{
   private:
      char* _userLoginName;  // store the login name of the user
      char* _password;  // explained later
      unsigned int _uid;  //user identifier
      unsigned int _gid;  // identifier of user's primary group
      char* _gecos; // general info of the user
      char* _home; // home directory of the user
      char* _shell;  // shell of the user
      // other private methods necessary for this class
   public:
      // constructors(empty and non-empty), destructor
      // ostream operator<<
      //     output to ostream using the following format
      //
      //     Login: [_userLoginName]
      //     Directory: [_home]
      //     Shell: [_shell]
      //     Gecos: [_gecos]
      //
      // and other public methods (mutators/setters, accessors/getters)
};
```

UserDB Class

The UserDB class is used to store the account information of the users, it is defined as follows:

```
class UserDB
{
   private:
      AccountInfo* _accounts[200];  // store up to 200 accounts
      unsigned int _size;  // number of account stored
      unsigned int _nextUid;  // next user id to be assigned
      unsigned int _defaultGid;  // default group id
      // other private methods necessary for this class
   public:
      // constructors(empty), destructor
      // Note: since the objects stored in _accounts are dynamically
      // allocated, you need delete the objects stored in it in
      // the destructor
      // ostream operator<< // print users in '/etc/passwd' format
                            // for the fields that are missing,
                            // don't print out anything. The only
                            // exception is 'x' will be shown if
                            // password is not set.
      void adduser( AccountInfo* newUser); // add a new user to
                         // _accounts, also increment _size.
                         // print out the following message after the
                         // user is added:
                         // "[userLoginName] with [uid] is added."
      void showUsers(); // print out "List of users:" at the first
```

```
                        // line, then print out all user login names
                        // (one line each user login name), then print
                        // out the following at the end according to
                        // the number of users stored
                        // 0 => "There's no users found in the system."
                        // 1 => "1 user found in the system."
                        // k => "k users found in the system." for k>1
    void showPasswd(); // call the ostream operator
    void finger(char* userLoginName);  // call ostream operator of
                                       // AccountInfo class
    int size();  // return the number of accounts stored (_size)
    // and other public methods (mutators/setters, accessors/getters)

};
```

main() method

The main() method reads different user commands from standard I/O. The available commands are:

1. adduser
   adduser command is used to add a user and has the following syntax:

   ```
   adduser user_login_name [-d home_directory] [-u user_id] [-g
   group_id] [-p password] [-s shell] [-c gecos]
   ```

   The first parameter will always be the *user login name* (maximum length: 8 characters not including NULL character). Other parameters are optional (the ones that are sandwiched by [and]) and can be in any order. But in the case that –c exists, it will be the last parameter and everything after –c will be used for gecos. More details about these parameters are described below.

   home_directory: The home directory of user, which is store in a C string. The default value is "home /home/userLoginName" where userLoginName is given in the first parameter. The max. length of home_directory is 32 not including NULL character.

   user_id: The unique user identifier of the user, which is an unsigned integer. It will be assigned to the user if it's given in the parameters. There is a variable _nextUid in UserDB class, which is used to generate user_id for a user when it is not provided. _nextUid starts with the value 1001. When user_id is not provided, _nextUid will be used as user_id for the new added user. Then _nextUid is incremented by 1.

   group_id: The primary group ID of the user, which is an unsigned integer. The default value is 1001.

   password: The password of the user, stored in a C string. The default value is NULL if not provided. The max. length of password is 16 characters (not including NULL).

   shell: The default shell of the user, stored in a C string. The default value is "/bin/bash". The max. length of shell is 16 not including NULL character.

gecos: A C string that store the general information about the user. The default value is NULL if not provided. The max. length of gecos is 64 not including NULL character.

A dynamically allocated object (tempAccount) will be created using the information provided with this command. Then adduser() of UserDB class will be called to add the user to the _accounts array, _size is incremented by 1.

2. finger
   Finger command has one parameter, which is a *user login name*. Then finger() of UserDB is called to print out the information of the user using the ostream operator of AccountInfo class.

3. showusers
   showusers does not take any parameter. In this command, showUsers() will be called to print out the userLoginNames stored in UserDB

4. showpasswd
   showpasswd does not take any parameter. showPasswd() of UserDB class will be called in this command.

You will implement the main() method as the code provided below.

```
#include <iostream>

// define and implement the required classes here,
// or in separate files, you will need to include the headers
/  for these classes in the latter case.

using namespace std;

int main()
{
   char buffer[256];
   AccountInfo *tempAccount;
   UserDB users;
   // other local variables used to store data temporally
   // such as user login name, home directory ... from adduser
   // or other commands

   while( !cin.eof())  // while end of line is not reached
   {
      // read a line into buffer

      // print out the line
      cout << buffer << endl;

      // check the user command and perform required actions

      // if it is an "adduser" command, dynamically create an
      //    AccountInfo object using tempAccount.
      //    Then add it to "users" by calling adduser(...)
```

```
        // if it is a "finger" command, print out the account info for
        // he specified user using the ostream operator of AccountInfo

        // if it is "showusers", print out all the users using the
        // ostream operator of UserDB class. (call showUsers() )

        // if it is "showpasswd", print out all the users in /etc/passwd
        // format using the print_passwd() method of UserDB class
        // (call showPasswd())

        // consume all the white spaces ( such as '\n', '\r' ...)

    }

    return 0;
}
```

Input File

The input file will consist an unknown number of lines and each line will be a command with the parameters of that command (if any). A line starts with '#' symbol is used for documentary and can be ignored. A short input file is provided below and a longer one will be posted to D2L later this week.

```
# This line is a comment
# add some users
adduser lisa
adduser hicks -d /home/trainers/hicks -c Master Weaponsmith
adduser randal -d /home/trainers/randal -c Riding Trainer
# show the user login names
showusers
# query the user with the name 'lisa'
finger lisa
# add some more users
adduser drusilla -d /home/trainers/drusilla -u 2001 -C Warlock Trainer
adduser wilhelm -p mypassword -d /home/trainers/wilhelm -C Paladin Trainer
adduser paxton -C Librarian
# show in passwd file format
showpasswd
# query the user with name wilhelm
finger wilhelm
```

Note: you can assume the input file is well-formed.

Redirected Input

Redirected input provides you a way to send a file to the standard input of a program without typing it using the keyboard. To use redirected input in Visual Studio environment, follow these steps: After you have opened or created a new project, on the menu go to project, project properties, expand configuration properties until you see Debugging, on the right you will see a set of options, and in the command arguments type "< **input filename**". The < sign is for redirected input and the **input filename** is the name of the input file (including the path if not in the working directory). A simple program that reads character by character until it reaches enf-of-file can be found below.

```
#include <iostream>

using namespace std;

//The character for end-of-line is '\n' and you can compare c below with this
//character to check if end-of-line is reached.

int main () {

      char c;

      cin.get(c);
      while (!cin.eof()) {
            i = c;
            cout << c;
            cin.get(c);
      }
      return 0;
}
```

C String

A string in the C Programming Language is an array of characters ends with '\0' (NULL) character. The NULL character denotes the end of the C string. For example, you can declare a C string like this:

```
char aCString[9];
```

Then you will be able to store up to 8 characters in this string. You can use **cout** to print out the string and the characters stored in a C string will be displayed one by one until '\0' is reached. Here are some examples:

0	1	2	3	4	5	6	7	8	cout result	Length
u	s	e	r	n	a	m	e	\0	username	8
n	a	m	e	\0					name	4
n	a	m	e	\0	1	2	3	4	name	4
\0									(nothing)	0

Similarly, you can use a for loop to determine the length of a string (NULL is NOT included). We show this in the following and also show how you can dynamically create a string using a pointer

```
char aCString[] = "This is a C String.";  // you don't need to provide
                                           // the size of the array
                                           // if the content is provided
char* anotherCString;                      // a pointer to an array of
                                           // characters

unsigned int length = 0;

while( aCString[length] != '\0')
{
   length++;
}
// the length of the string is now known

anotherCString = new char[length+1];  // need space for NULL character

// copy the string

for( int i=0; i< length+1; i++)
   anotherCString[i] = aCString[i];

cout << aCString << endl;       // print out the two strings

cout << anotherCSring << endl;

delete [] anotherCString;     // release the memory after use
```

You check http://www.cs.bu.edu/teaching/cpp/string/array-vs-ptr/, other online sources or textbooks to learn more about this.

Constraints

1. In this project, the only header you will use is #include <iostream>.
2. None of the projects is a group project. Consulting with other members of this class on programming projects is strictly not allowed and plagiarism charges will be imposed on students who do not follow this.