|  |  |
| --- | --- |
| THE NATIONAL INSTITUTE OF ENGG. |  |
| C++PROJECT:: VIRTUAL ATM STIMULATOR |  |
| PROJECT UNDER GUIDENCE OF:: YOGESH M J SIR & PRATIBHA B S MADAM |  |

|  |
| --- |
|  |

INTRODUCTION:::

**ABOUT THE ATM(AUTOMATIC TELLER MACHINE)**

An automated teller machine or automatic teller machine (ATM), also known as an automated banking machine (ABM) in Canada, and a Cashpoint (which is a trademark of Lloyds TSB), cash machine or sometimes a hole in the wall in British English, is a computerized telecommunications device that provides the clients of a financial institution with access to financial transactions in a public space without the need for a cashier, human clerk or bank teller. ATMs are known by various other names including ATM machine, automated banking machine, and various regional variants derived from trademarks on ATM systems held by particular banks.

On most modern ATMs, the customer is identified by inserting a plastic ATM card with a magnetic stripe or a plastic smart card with a chip, that contains a unique card number and some security information such as an expiration date or CVVC (CVV). Authentication is provided by the customer entering a personal identification number (PIN).

Using an ATM, customers can access their bank accounts in order to make cash withdrawals, debit card cash advances, and check their account balances as well as purchase prepaid cellphone credit. If the currency being withdrawn from the ATM is different from that which the bank account is denominated in (e.g.: Withdrawing Japanese Yen from a bank account containing US Dollars), the money will be converted at an official wholesale exchange rate.

**Input/Output with files**

C++ provides the following classes to perform output and input of characters to/from files:

ofstream: Stream class to write on files

ifstream: Stream class to read from files

fstream: Stream class to both read and write from/to files.

These classes are derived directly or indirectly from the classes istream, and ostream. We have already used objects whose types were these classes: cin is an object of class istream and cout is an object of class ostream. Therfore, we have already been using classes that are related to our file streams. And in fact, we can use our file streams the same way we are already used to use cin and cout, with the only difference that we have to associate these streams with physical files

**Open a file:::**The first operation generally performed on an object of one of these classes is to associate it to a real file. This procedure is known as to open a file. An open file is represented within a program by a stream object (an instantiation of one of these classes, in the previous example this was myfile) and any input or output operation performed on this stream object will be applied to the physical file associated to it.

In order to open a file with a stream object we use its member function open():

open (filename, mode);

Where filename is a null-terminated character sequence of type const char \* (the same type that string literals have) representing the name of the file to be opened, and mode is an optional parameter with a combination of the following flags:

ios::in Open for input operations.

ios::out Open for output operations.

ios::binary Open in binary mode.

If this flag is not set to any value, the initial position is the beginning of the file.

ios::app All output operations are performed at the end of the file, appending the content to the current content of the file. This flag can only be used in streams open for output-only operations.

OR (|). Each one of the open() member functions of the classes ofstream, ifstream and fstream has a default mode that is used if the file is opened without a second argument:

class default mode parameter

ofstream ios::out

ifstream ios::in

fstream ios::in | ios::out

For ifstream and ofstream classes, ios::in and ios::out are automatically and respectively assumed, even if a mode that does not include them is passed as second argument to the open() member function.  
  
The default value is only applied if the function is called without specifying any value for the mode parameter. If the function is called with any value in that parameter the default mode is overridden, not combined.  
  
File streams opened in binary mode perform input and output operations independently of any format considerations. Non-binary files are known as *text files*, and some translations may occur due to formatting of some special characters (like newline and carriage return characters).  
  
Since the first task that is performed on a file stream object is generally to open a file, these three classes include a constructor that automatically calls the open() member function and has the exact same parameters as this member. Therefore, we could also have declared the previous myfile object and conducted the same opening operation in our previous example by writing:

|  |  |
| --- | --- |
|  | ofstream myfile ("example.bin", ios::out | ios::app | ios::binary); |

Combining object construction and stream opening in a single statement. Both forms to open a file are valid and equivalent.  
  
To check if a file stream was successful opening a file, you can do it by calling to member is\_open() with no arguments. This member function returns a bool value of true in the case that indeed the stream object is associated with an open file, or false otherwise:

|  |  |
| --- | --- |
|  | if (myfile.is\_open()) { /\* ok, proceed with output \*/ } |

**Closing a file**

When we are finished with our input and output operations on a file we shall close it so that its resources become available again. In order to do that we have to call the stream's member function close(). This member function takes no parameters, and what it does is to flush the associated buffers and close the file:

|  |  |
| --- | --- |
|  | myfile.close(); |

Once this member function is called, the stream object can be used to open another file, and the file is available again to be opened by other processes.  
  
In case that an object is destructed while still associated with an open file, the destructor automatically calls the member function close().

# strtok

<cstring>

char \* strtok ( char \* str, const char \* delimiters );

**Split string into tokens**

A sequence of calls to this function split *str* into tokens, which are sequences of contiguous characters separated by any of the characters that are part of *delimiters*.  
  
On a first call, the function expects a C string as argument for *str*, whose first character is used as the starting location to scan for tokens. In subsequent calls, the function expects a null pointer and uses the position right after the end of last token as the new starting location for scanning.  
  
To determine the beginning and the end of a token, the function first scans from the starting location for the first character **not** contained in *delimiters* (which becomes the *beginning of the token*). And then scans starting from this*beginning of the token* for the first character contained in *delimiters*, which becomes the *end of the token*.  
  
This *end of the token* is automatically replaced by a null-character by the function, and the *beginning of the token* is returned by the function.  
  
Once the terminating null character of *str* has been found in a call to *strtok*, all subsequent calls to this function with a null pointer as the first argument return a null pointer.

### Parameters

str

C string to truncate. The contents of this string are modified and broken into smaller strings (tokens).  
Alternativelly, a null pointer may be specified, in which case the function continues scanning where a previous successful call to the function ended.

delimiters

C string containing the delimiters.  
These may vary from one call to another.

### Return Value

A pointer to the last token found in string.  
A null pointer is returned if there are no tokens left to retrieve.

**strtol**

<cstdlib>

long int strtol ( const char \* str, char \*\* endptr, int base );

Convert string to long integer

Parses the C string str interpreting its content as an integral number of the specified base, which is returned as a long int value.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes as many characters as possible that are valid following a syntax that depends on the base parameter, and interprets them as a numerical value. Finally, a pointer to the first character following the integer representation in str is stored in the object pointed by endptr.

If the value of base is zero, the syntax expected is similar to that of integer constants, which is formed by a succession of:

An optional plus or minus sign

An optional prefix indicating octal or hexadecimal base ("0" or "0x" respectively)

A sequence of decimal digits (if no base prefix was specified) or either octal or hexadecimal digits if a specific prefix is present

If the base value is between 2 and 36, the format expected for the integral number is a succession of the valid digits and/or letters needed to represent integers of the specified radix (starting from '0' and up to 'z'/'Z' for radix 36). The sequence may optionally be preceded by a plus or minus sign and, if base is 16, an optional "0x" or "0X" prefix.

If the first sequence of non-whitespace characters in str is not a valid integral number as defined above, or if no such sequence exists because either str is empty or it contains only whitespace characters, no conversion is performed.

Parameters

str

C string containing the representation of an integral number.

endptr

Reference to an object of type char\*, whose value is set by the function to the next character in str after the numerical value.

This parameter can also be a null pointer, in which case it is not used.

Return Value

On success, the function returns the converted integral number as a long int value.

If no valid conversion could be performed, a zero value is returned.

If the correct value is out of the range of representable values, LONG\_MAX or LONG\_MIN is returned, and the global variable errno is set to ERANGE.

Inheritance-object is of the derived class-“atm1”……….base class is “atm”

Inheritance is a mechanism of reusing and extending existing classes without modifying them, thus producing hierarchical relationships between them.

Inheritance is almost like embedding an object into a class. Suppose that you declare an object x of class A in the class definition of B. As a result, class B will have access to all the public data members and member functions of class A. However, in class B, you have to access the data members and member functions of class A through object x

Friend function:::

Friend functions

In principle, private and protected members of a class cannot be accessed from outside the same class in which they are declared. However, this rule does not affect friends.

Friends are functions or classes declared with the friend keyword.

If we want to declare an external function as friend of a class, thus allowing this function to have access to the private and protected members of this class, we do it by declaring a prototype of this external function within the class, and preceding it with the keyword friend:

Default constructor

Header file-🡪 #include<ctime>

ctime

<ctime>

char \* ctime ( const time\_t \* timer );

Convert time\_t value to string

Converts the time\_t object pointed by timer to a C string containing a human-readable version of the corresponding local time and date.

The returned string has the following format:

Www Mmm dd hh:mm:ss yyyy

Where Www is the weekday, Mmm the month in letters, dd the day of the month, hh:mm:ss the time, and yyyy the year.

The string is followed by a new-line character ('\n') and the terminating null-character.

This function is equivalent to: asctime(localtime(timer)).

Parameters

timer

Pointer to a time\_t object that contains a calendar time.

Return Value

A C string containing the date and time information in a human-readable format.

The array which holds this string is statically allocated and shared by both the ctime and asctime functions. Each time either one of these functions is called the content of this array is overwritten.

Clock() function.

**Flow of control in the programming:::::**

**In main function::**

>pin is inserted and verified

>options are given to the user.(display menu)

>if( **mini statement**)

>Opening the bal.txt

>verification of password

>tokenization of text file

>printing the last 10 transtion.

>closing of bal.txt

>fun::**balance update**

>opening of balance and text file.

> first token(password) is verified to pick the line.

>updating the balance

>bal.txt is removed

>renaming the temp file

If(**withdrawal**)

>temp file is opened in append mode

>password is verified to select required line.

>flow goes to enter\_w().

>the tokenized string is converted into long int

>condition of withdrawal is being checked.

>the flow is send to balance is updated.

>balance file is updated.

If (**displaying** )

>checking the first token

>tokenizing and going to the balance token

>displaying the current account

If**(account detail**)

>search the pin

>then tokenize each every detail in the line and then display the content.

If**(new password**)

> old password is confirmed

> enter the new password

> tokenizing and printing the detail in text file.

>new password is updated.flow is send to upddation.

**Conclusion and future work:**

There is lot of scope and improvement in this project. The project can further be extended in

1. the graphics mode in which the options and screen could be more user friendly

2. various options and sound characters could be added to make it user friendly.

3. further the project could be connected to the server to make more functional and vibrant.

4. codeing could be made useind hardware interface…..eg. actually, verifying the card.

5. project can be opened in the mode of software.

6. codes could be made more efficient and non-time consuming.

7. timer could be added to the project.

8.mouse interface could be added to the project.

REFERENCE MATERIAL USED FOR THE PROJECT::::::::::

1. Complete reference to c++: herbet schildt

2. www.wikipedia.com

3. www.cplusepluse.com

4. TMH:::: mastering C++:

5. random articles by independent users.