

**DATA**

**STRUCTURES   
AND**

**ALGORITHMS**

Project by: **M. Anas Baig**

01-134152-037

BS(CS)-4A



[Document subtitle]

May 23, 2017

Title: **Spending(Budget) Tracking Application**

**INTRODUCTION: .**

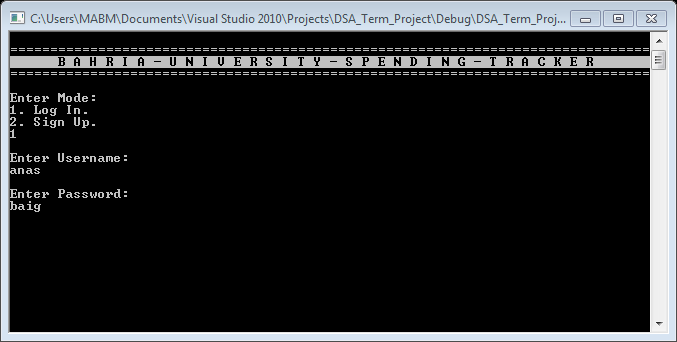
Spending Tracker is basically an application which keeps record of the daily **Income** and **Expenses** of the user. It helps the user to maintains his daily budget and saves money by keeping an eye on his spending.  
 **PROGRAMMING CONCEPTS USED: .**

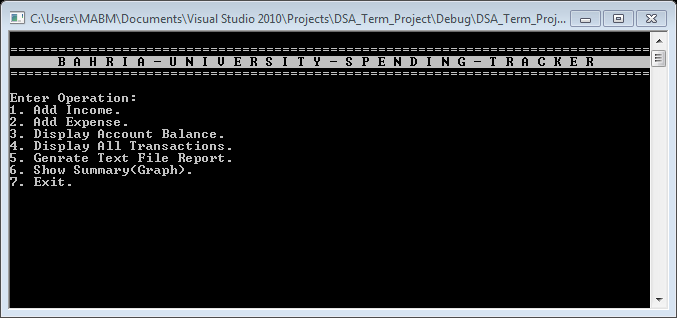
1. Linked Lists.
2. Dynamic Memory Allocation.
3. Traversing Linked Lists.
4. Composition
5. Insertion.
6. Searching.
7. Pointers.
8. Functions.
9. Strings.

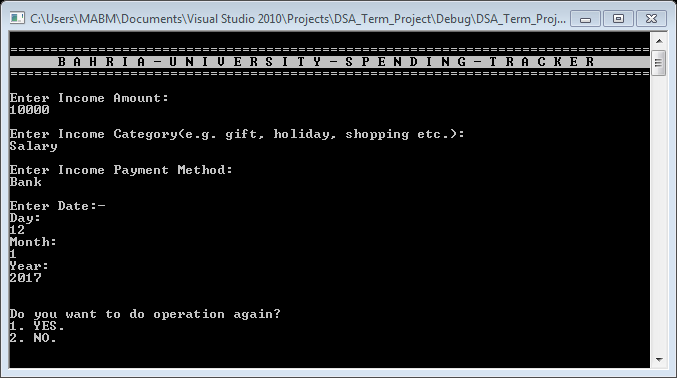
**APPLICATION FEATURES: .**

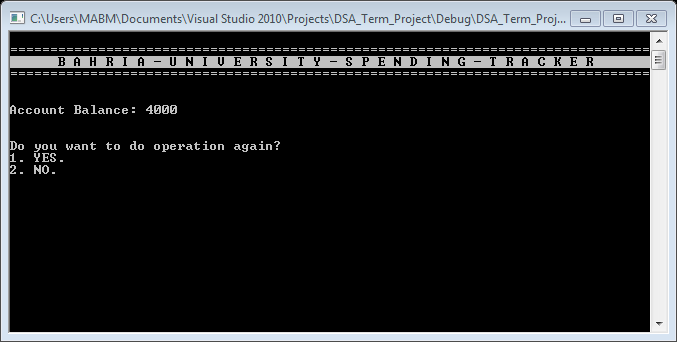
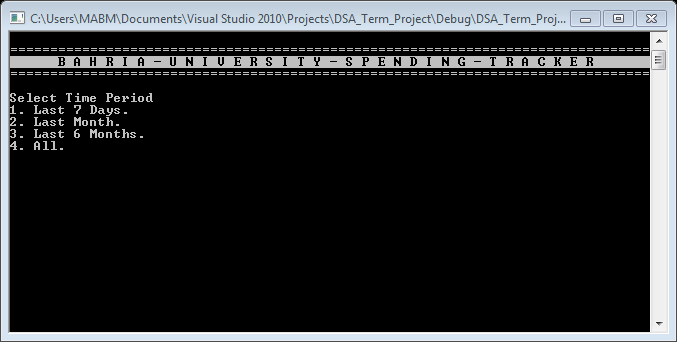
* **Secure Sign Up and Log In:**Completely secure Sign up with storing user credentials in ***Binary File Format*** which is unreadable by human. ***No duplication in Usernames.***
* ***Text File Report Generation*** which user can use for any purpose like printing or storing as backup.
* ***Summary Graphs*** and ***Percentage Analysis*** to each expense category which makes it easy for user to determine its expenses.
* ***Separate file*** to store transactions for each user which can’t be accessed without password.
* Add ***Incomes/Expenses*** with information such as ***Amount, Date, Payment Method used and Category***.
* ***Displays Final Balance*** in Account after each transaction is made.
* ***Displays Transactions List*** according to user’s choice as last ***7 Days, last Month or Yearly.***
* Best and most important feature is that it can be used on a **Public PC** where each user has its own ***USERNAME*** and ***PASSWORD*** and has its own separate file which can’t be accessed without password.

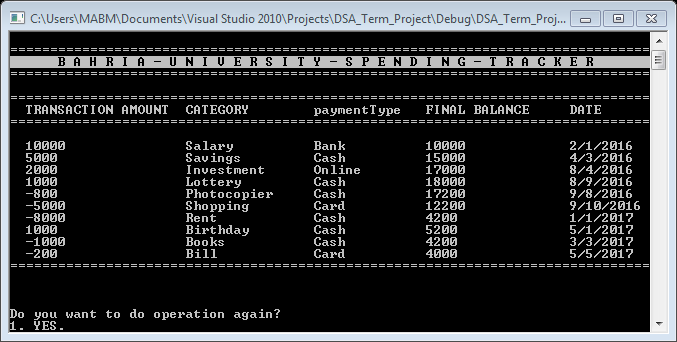
**CONSOLE OUTPUT: .**

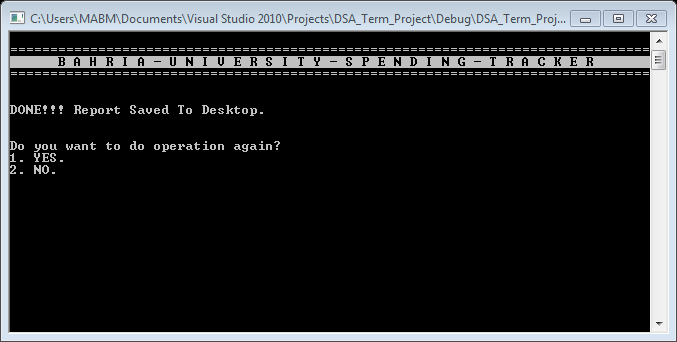


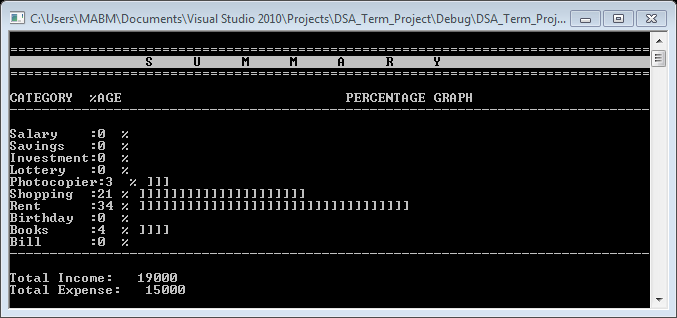




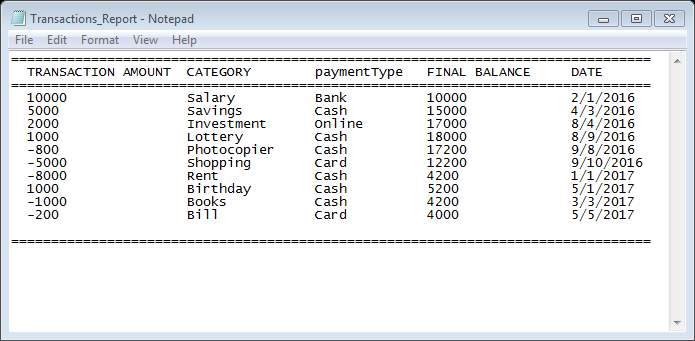
 







**Report Text File Generated:**

****

**APPLICATION CODE: .**

**date.h File:**

1. #pragma once
3. #include<iostream>
4. **using** **namespace** std;
5. **class** date
6. {
7. **public**:
8. **int** day, month, year; //date variables
9. **public**:
10. date(**void**);
11. **void** getInput(); //to get input date
12. **void** displayDate(); //to display date
13. };

**date.cpp File:**

1. #include "date.h"
2. #include<iostream>
3. **using** **namespace** std;
5. date::date(**void**)
6. {
7. }
9. **void** date::getInput()
10. {
11. cout<<endl<<"Enter Date:-"<<endl;
12. cout<<"Day:"<<endl;
13. cin>>day;
14. cout<<"Month:"<<endl;
15. cin>>month;
16. cout<<"Year:"<<endl;
17. cin>>year;
18. }
20. **void** date::displayDate()
21. {
22. cout<<day<<"/"<<month<<"/"<<year;
23. }

**user.h File:**

1. #pragma once
2. #include "string.h"
3. #include <iostream>
4. **using** **namespace** std;
6. **class** user
7. {
8. **public**:
9. string username, password; //to store login credentials
10. **public**:
11. user(**void**);
12. **void** getData(**int**); //to get login credentials rom user
13. string getUsername(); //get username function
14. string getPassword(); //get password function
15. };

**user.cpp File:**

1. #include "user.h"
2. #include "string"
3. #include "transaction.h"
4. #include "string.h"
5. #include "fstream"
6. #include <iostream>
7. **using** **namespace** std;
9. user::user(**void**)
10. {
11. }
13. **void** user::getData(**int** type)
14. {
15. **if**( type==1 ) //log in
16. {
17. **bool** checkpoint; //flag to check credentials accuracy
18. string temp\_username; //temporary username variable
19. string temp\_password; //temporary password variable
20. **do** //loop to get input again and again untill correct username is entered
21. {
22. checkpoint = **false**; //flag set to false
24. cout<<endl;
25. cout<<"Enter Username:"<<endl;
26. cin>>temp\_username; //username input
28. cout<<endl<<"Enter Password:"<<endl;
29. cin>>temp\_password; //password input
31. ifstream read\_file("credentials.txt", ios::binary); //read mode file open which contains login credentials of all users
33. **for**( ;!(read\_file.eof()); ) //no initialization or increment statements because there we only check loop with EOF
34. {
35. read\_file.read(**reinterpret\_cast**<**char** \*>(**this**), **sizeof**(\***this**)); //reading credentials file
36. **if**((temp\_username == **this**->username) && (temp\_password==**this**->password)) //it checks either username and password are correct or not
37. {
38. checkpoint = **true**; //if username and password are entered correctly
39. }
40. }
42. **if**(checkpoint ==  **false**) //if username or password is incorrectly typed
43. {
44. cout<<"Incorrect USERNAME or PASSWORD!!! TRY AGAIN"<<endl<<endl;
45. }
46. read\_file.close(); //file close
47. }
48. **while**(checkpoint == **false**); //let it get input from user unless correct username and password is entered (i.e. checkpoint=true)
50. **if**(checkpoint == **true**) //condition when username and password are correctly typed
51. {
52. **this**->username = temp\_username; //stores username
53. **this**->password = temp\_password; //stores password
54. }
55. }
56. **if**( type == 2) //sign up
57. {
58. **bool** checkpoint; //flag to check credentials uniqueness
59. string temp\_username; //stores username temporarily
60. string temp\_password; //stores password temporarily
62. **do** //asks user for username and password untill he enters unique username(i.e. no two users can have similar usernames)
63. {
64. checkpoint = **true**; //set flag to false
66. cout<<endl;
67. cout<<"Enter Username:"<<endl;
68. cin>>temp\_username; //username input
70. cout<<endl<<"Enter Password:"<<endl;
71. cin>>temp\_password; //password input
73. ifstream read\_file("credentials.txt", ios::binary); //file open read mode to check either entered username is already registered or not
74. **for**( ;!(read\_file.eof()); ) //checks for EOF, no intialization or increment statements necassary
75. {
76. read\_file.read(**reinterpret\_cast**<**char** \*>(**this**), **sizeof**(\***this**)); //reads object from file
77. **if**(temp\_username == **this**->username) //if entered username is registered
78. {
79. cout<<"USERNAME Already Found!!! TRY AGAIN"<<endl<<endl;
80. checkpoint = **false**; //set flag to true
81. **break**;
82. }
83. }
84. read\_file.close(); //file close
85. }
86. **while**(checkpoint == **false**); //continues loop till checkpoint is false (i.e. username already registered)
88. **if**(checkpoint == **true**) //if duplicate username not found
89. {
90. **this**->username = temp\_username; //stores username
91. **this**->password = temp\_password; //stores password
92. transaction t; //transaction class object
93. t.setZero(**this**->username, **this**->password); //this function sets all variables of new user to 0
94. ofstream write\_file("credentials.txt",ios::binary|ios::app); //file write mode open to save new user login information to file
95. write\_file.write(**reinterpret\_cast**<**char** \*>(**this**), **sizeof**(\***this**)); //writes object to file
96. write\_file.close(); //file close
97. }
98. }
99. }
101. string user::getUsername() //get username fucntion
102. {
103. **return**(username);
104. }
106. string user::getPassword() //get password function
107. {
108. **return**(password);
109. }

**transactionNode.h File:**

1. #include "string.h"
2. #include "date.h"
4. #pragma once
5. **class** transactionNode
6. {
7. **public**:
8. **int** amount; //transaction amount
9. **int** balance; //final balance
10. string category; //type of transaction
11. string paymentType; //paymentType where transaction made
12. date d; //to store date of transaction
13. transactionNode \*next;
14. **public**:
15. transactionNode(**void**);
16. };

**transactionNode.cpp File:**

1. #include "transactionNode.h"
2. #include "string.h"
3. #include "date.h"
5. transactionNode::transactionNode(**void**)
6. {
7. }

**transaction.h File:**

1. #pragma once
2. #include "fstream"
3. #include "date.h"
4. #include "string"
5. #include "transactionNode.h"
6. #include "string.h"
7. #include <iostream>
8. **using** **namespace** std;
10. **class** transaction
11. {
12. **public**:
13. transactionNode \*head;
14. **public**:
15. transaction(**void**); //constructor
16. **void** setZero(string username, string password); //set all values of new user to zero
17. **void** getAmount(string, string, **int**); //function to get input about transaction details from user
18. **void** displayTransaction(string); //to display transactions  with respect to user's choice time period
19. **void** finalBalance(string); //to display final balance in account
20. transactionNode\* fileToList(string);  //converts file data to linked list
21. **void** generateReport(string); //to generate textual report of transactions
22. **void** summary(string);
23. };

**transaction.cpp File:**

1. #include "transaction.h"
2. #include "fstream"
3. #include "string"
4. #include "conio.h"
5. #include "iomanip"
6. #include <stdio.h>
7. #include <stdlib.h>
8. #include <windows.h>
9. #include <iostream>
10. **using** **namespace** std;
12. transaction::transaction(**void**) //constructor
13. {
14. head = **new** transactionNode;
15. }
17. **void** transaction::setZero(string username, string password)
18. {
19. head->amount = 0; //set to zero
20. head->balance = 0; //set to zero
21. head->d.day = head->d.month = head->d.year = 0;
22. head->next = NULL;
24. //below are empty file creation for new user with its username to store final balance
25. ofstream username\_write(username.c\_str(), ios::binary); //file write mode open to store final balance
26. username\_write.write(**reinterpret\_cast**<**char** \*>(head), **sizeof**(\*head)); //writes object to file
27. username\_write.close(); //file close
29. //below are empty file creation for new user with its password to store all transactions
30. ofstream password\_write(password.c\_str(), ios::binary); //file write mode open to store all transactions
31. password\_write.write(**reinterpret\_cast**<**char** \*>(head), **sizeof**(\*head)); //writes object to file
32. password\_write.close(); //file close
33. }
35. **void** transaction::getAmount(string username, string password, **int** type) //function to get input from user
36. {
37. **if**( type == 1 ) //income
38. {
39. transactionNode \*ptr = **new** transactionNode; //node created to add new transaction
40. **int** temp\_amount; //temporary amount storing variable
42. cout<<"Enter Income Amount:"<<endl;
43. cin>>temp\_amount; //income amount input
45. ifstream username\_read(username.c\_str(), ios::binary); //file open read mode to get final balance
46. username\_read.read(**reinterpret\_cast**<**char** \*>(ptr), **sizeof**(\*ptr)); //reads object from file
47. ptr->balance = ptr->balance + temp\_amount; //adds amount to final balance (i.e. updating final balance)
48. username\_read.close(); //file close
50. ptr->amount = temp\_amount; //sets amount
52. cout<<endl<<"Enter Income Category(e.g. gift, holiday, shopping etc.):"<<endl;
53. cin>>ptr->category; //transaction type
55. cout<<endl<<"Enter Income Payment Method:"<<endl;
56. cin>>ptr->paymentType; //transaction location
58. ptr->d.getInput(); //date input
60. ofstream username\_write(username.c\_str(), ios::binary); //file write mode open to write final balance reading
61. username\_write.write(**reinterpret\_cast**<**char** \*>(ptr), **sizeof**(\*ptr)); //writes object to file
62. username\_write.close(); //file close
64. //-----------------------Above is work with username.txt file which only has 1 record for final balance------
65. //-----------------------Below is work with password.txt file which contains transactions list------
67. ofstream password\_write(password.c\_str(), ios::binary|ios::app); //file write mode open to write transactions details
68. password\_write.write(**reinterpret\_cast**<**char** \*>(ptr), **sizeof**(\*ptr)); //writes object to file
69. password\_write.close(); //file close
70. }
71. **if**(type ==2) //expense
72. {
73. transactionNode \*ptr = **new** transactionNode; //node created to add new transaction
74. **int** temp\_amount; //temporary amount storing variable
76. cout<<"Enter Expense Amount:"<<endl;
77. cin>>temp\_amount; //expense amount input
79. temp\_amount = (temp\_amount \* (-1)); //set debit amount to -ve so that when all transactions are displayed user could identify which one is income and which one is expense
81. ifstream username\_read(username.c\_str(), ios::binary); //file read mode open to get final balance
82. username\_read.read(**reinterpret\_cast**<**char** \*>(ptr), **sizeof**(\*ptr)); //reads object form file
84. ptr->balance = ptr->balance + temp\_amount; //adds amount to final balance (i.e. updates final balance)
85. username\_read.close(); //file close
87. ptr->amount = temp\_amount; //amount set
89. cout<<endl<<"Enter Exoense Category(e.g. gift, holiday, shopping etc.):"<<endl;
90. cin>>ptr->category; //gets transaction type
92. cout<<endl<<"Enter Expense Payment Method:"<<endl;
93. cin>>ptr->paymentType; //gets transaction location
95. ptr->d.getInput(); //gets date input
97. ofstream username\_write(username.c\_str(), ios::binary); //file write mode open to store final balance
98. username\_write.write(**reinterpret\_cast**<**char** \*>(ptr), **sizeof**(\*ptr)); //writes object to file
99. username\_write.close(); //file close
101. //-----------------------Above is work with username.txt file which only has 1 record for final balance------
102. //-----------------------Below is work with password.txt file which contains transactions list---------------
104. ofstream password\_write(password.c\_str(), ios::binary|ios::app); //file write mode open to store transaction details
105. password\_write.write(**reinterpret\_cast**<**char** \*>(ptr), **sizeof**(\*ptr)); //writes object to file
106. password\_write.close(); //file close
107. }
108. }
110. **void** transaction::finalBalance(string filename) //to display final balance in account
111. {
112. transactionNode \*ptr = **new** transactionNode;
114. ifstream file\_read(filename.c\_str(), ios::binary); //file read mode open to read final balance
115. file\_read.read(**reinterpret\_cast**<**char** \*>(ptr), **sizeof**(\*ptr)); //reads object form file
117. cout<<endl<<"Account Balance: "<<ptr->balance<<endl;
119. file\_read.close(); //file close
120. }
122. transactionNode\* transaction::fileToList(string filename) //converts file data to linked list
123. {
124. transactionNode \*ptr = **new** transactionNode;
125. transactionNode \*temp = **new** transactionNode;
126. transactionNode \*list = **new** transactionNode;
128. ifstream file\_read(filename.c\_str(), ios::binary); //file read mode open to read transactions
130. file\_read.read(**reinterpret\_cast**<**char** \*>(ptr), **sizeof**(\*ptr)); //reads object form file
131. temp = ptr;
132. list = temp;
134. **while**(!file\_read.eof())
135. {
136. file\_read.read(**reinterpret\_cast**<**char** \*>(ptr), **sizeof**(\*ptr)); //reads object form file
137. temp->next = ptr; //connecting each file read data (node) to linked list
138. ptr = **new** transactionNode;
139. temp = temp->next;
140. }
141. temp->next = NULL;
143. **return** list;
144. }
146. **void** transaction::displayTransaction(string filename) //to display transactions  with respect to user's choice time period
147. {
148. **int** duration;
150. cout<<"Select Time Period"<<endl;
151. cout<<"1. Last 7 Days."<<endl;
152. cout<<"2. Last Month."<<endl;
153. cout<<"3. Last 6 Months."<<endl;
154. cout<<"4. All."<<endl;
155. cin>>duration;

158. system("CLS");
159. cout<<endl<<"================================================================================";
160. **HANDLE** m\_hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);
161. SetConsoleTextAttribute (  m\_hConsole,
162. BACKGROUND\_RED |
163. BACKGROUND\_GREEN |
164. BACKGROUND\_BLUE );
165. std::cout <<"      B A H R I A - U N I V E R S I T Y - S P E N D I N G - T R A C K E R       ";
166. SetConsoleTextAttribute ( m\_hConsole,
167. FOREGROUND\_RED |
168. FOREGROUND\_GREEN |
169. FOREGROUND\_BLUE );
170. cout<<"================================================================================"<<endl;
171. transactionNode \*temp = fileToList(filename);
173. cout<<"================================================================================";
174. cout<<"  "<<setw(20)<<left<<"TRANSACTION AMOUNT";
175. cout<<setw(16)<<left<<"CATEGORY";
176. cout<<setw(14)<<left<<"paymentType";
177. cout<<setw(18)<<left<<"FINAL BALANCE";
178. cout<<"DATE"<<endl;
179. cout<<"================================================================================"<<endl;
181. **if**( duration == 1 ) //1 is specified for last 7 days(week)
182. {
183. **int** sevenDay = (temp->d.day) - 7;
184. **int** thisDay = temp->d.day; //last transaction's day
185. **int** thisMonth = temp->d.month; //last transaction's month
186. **int** thisYear = temp->d.year; //last transaction's year
188. temp = fileToList(filename); //converts file data to linked list
190. **while**( temp->next != NULL )
191. {
192. **if**( temp->d.day >= sevenDay && temp->d.day <= thisDay && temp->d.month == thisMonth && temp->d.year == thisYear ) //condition for last 7 days transactions
193. {
194. cout<<"  "<<setw(20)<<left<<temp->amount<<setw(16)<<left<<temp->category<<setw(14)<<left<<temp->paymentType<<setw(18)<<left<<temp->balance;
195. temp->d.displayDate();
196. cout<<endl;
197. }
198. **else**
199. {
200. }
201. temp = temp->next;
202. }
203. }
204. **else** **if**( duration == 2 ) //2 is specified for last 30 days(month)
205. {
206. **int** thirtyDay = (temp->d.day) - 30;
207. **int** thisDay = temp->d.day; //last transaction's day
208. **int** thisMonth = temp->d.month;
209. **int** thisYear = temp->d.year;
211. temp = fileToList(filename); //converts file data to linked list
213. **while**( temp->next != NULL )
214. {
215. **if**( temp->d.day >= thirtyDay && temp->d.day <= thisDay && temp->d.month == thisMonth && temp->d.year == thisYear ) //condition for last 30 days transactions
216. {
217. cout<<"  "<<setw(20)<<left<<temp->amount<<setw(16)<<left<<temp->category<<setw(14)<<left<<temp->paymentType<<setw(18)<<left<<temp->balance;
218. temp->d.displayDate();
219. cout<<endl;
220. }
221. **else**
222. {
223. }
224. temp = temp->next;
225. }
226. }
227. **else** **if**( duration == 3 ) //3 is specified for last 6 months
228. {
229. **int** month = (temp->d.month) - 6;
230. //int thisDay = temp->d.day; //last transaction's day
231. **int** thisMonth = temp->d.month;
232. **int** thisYear = temp->d.year;
234. temp = fileToList(filename);
236. **while**( temp->next != NULL )
237. {
238. **if**( temp->d.month >= month && temp->d.month <= thisMonth && temp->d.year == thisYear ) //condition for last year transactions
239. {
240. cout<<"  "<<setw(20)<<left<<temp->amount<<setw(16)<<left<<temp->category<<setw(14)<<left<<temp->paymentType<<setw(18)<<left<<temp->balance;
241. temp->d.displayDate();
242. cout<<endl;
243. }
244. **else**
245. {
246. }
247. temp = temp->next;
248. }
249. }
250. **else** //display all transactions
251. {
252. **while**( temp->next != NULL )
253. {
254. cout<<"  "<<setw(20)<<left<<temp->amount<<setw(16)<<left<<temp->category<<setw(14)<<left<<temp->paymentType<<setw(18)<<left<<temp->balance;
255. temp->d.displayDate();
256. cout<<endl;
257. temp = temp->next;
258. }
259. }
260. cout<<"================================================================================"<<endl;
261. }
263. **void** transaction::generateReport( string filename ) //to generate textual report of transactions
264. {
265. transactionNode \*temp = fileToList(filename); //converts file data to linked list
267. ofstream report\_write("C:\\Users\\MABM\\Desktop\\Transactions\_Report.txt"); //file write mode open to write report
269. report\_write<<"================================================================================";
270. report\_write<<endl;
271. report\_write<<"  "<<setw(20)<<left<<"TRANSACTION AMOUNT";
272. report\_write<<setw(16)<<left<<"CATEGORY";
273. report\_write<<setw(14)<<left<<"paymentType";
274. report\_write<<setw(18)<<left<<"FINAL BALANCE";
275. report\_write<<"DATE";
276. report\_write<<endl;
277. report\_write<<"================================================================================";
278. report\_write<<endl;
280. **while**( temp->next != NULL )
281. {
282. report\_write<<"  "<<setw(20)<<left<<temp->amount<<setw(16)<<left<<temp->category<<setw(14)<<left<<temp->paymentType<<setw(18)<<left<<temp->balance;
283. report\_write<<temp->d.day<<"/"<<temp->d.month<<"/"<<temp->d.year;
284. report\_write<<endl;
285. temp = temp->next;
286. }
288. report\_write<<endl;
289. report\_write<<"================================================================================";
291. cout<<endl;
292. cout<<"DONE!!! Report Saved To Desktop."<<endl;
294. report\_write.close(); //file close
295. }
297. **void** transaction::summary(string filename)
298. {
299. //cout<<"================================================================================"<<endl;
300. //cout<<"                 S     U     M     M     A     R     Y                          "<<endl;
301. //cout<<"================================================================================"<<endl;
303. system("CLS");
304. cout<<endl<<"================================================================================";
305. **HANDLE** m\_hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);
306. SetConsoleTextAttribute (  m\_hConsole,
307. BACKGROUND\_RED |
308. BACKGROUND\_GREEN |
309. BACKGROUND\_BLUE );
310. std::cout <<"                 S     U     M     M     A     R     Y                          ";
311. SetConsoleTextAttribute ( m\_hConsole,
312. FOREGROUND\_RED |
313. FOREGROUND\_GREEN |
314. FOREGROUND\_BLUE );
315. cout<<"================================================================================"<<endl;
317. cout<<setw(10)<<left<<"CATEGORY"<<"";
318. cout<<setw(3)<<left<<"%AGE";
319. cout<<left<<"                            PERCENTAGE GRAPH"<<endl;
320. cout<<"--------------------------------------------------------------------------------"<<endl;
322. transactionNode \*temp1 = fileToList( filename );
324. **int** incomeSum = 0; //to store sum of incomes
325. **int** expenseSum = 0; //to store sum of expenses
327. **while**(temp1->next != NULL)
328. {
329. **if**(temp1->amount >= 0)
330. {
331. incomeSum = ( temp1->amount + incomeSum ); //to calculate total income of user
332. }
333. **else**
334. {
335. **int** amount = ( temp1->amount \* (-1) ); //to make expense positive
336. expenseSum = ( amount + expenseSum ); //to calculate total income of user
337. }
338. temp1 = temp1->next;
339. }
341. temp1 = fileToList( filename ); //renewing head again to start of linked list
343. **while**( temp1->next != NULL ) //loop to make sum of transactions of similar category and then determining their graph
344. {
345. **int** expenseCategory = 0; //to calculate sum of the transations of same category
347. transactionNode \*temp2 = fileToList(filename); //converts file data to list
349. **while**( temp2->next != NULL ) //compares each item of list to the remaining list
350. {
351. **if**( temp1->category == temp2->category && temp2->amount < 0 ) //if category is same and expense amount only(amount>0)
352. {
353. expenseCategory = ( expenseCategory + ( -1 \* temp2->amount ) );
354. }
355. temp2 = temp2->next;
356. }
358. **int** percentage = ( ( (expenseCategory\*1.0) / (expenseSum\*1.0) ) \* 65.0 ); //calculating percentage 65 is just taken on my own choice to make 65 ']' brackets for 100% and vice versa
360. cout<<setw(10)<<left<<temp1->category<<":";
361. cout<<setw(3)<<left<<percentage<<"% ";
363. **while**( percentage ) //displaying ']' brackets according to percentage of expenses of category
364. {
365. percentage--;
366. cout<<"]";
367. }
368. cout<<endl;
370. temp1 = temp1->next;
371. }
372. cout<<"--------------------------------------------------------------------------------"<<endl;
374. cout<<"Total Income:   "<<incomeSum<<endl;
375. cout<<"Total Expense:   "<<expenseSum<<endl;
376. }

**main.cpp File:**

1. #include "iomanip"
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <windows.h>
5. #include "transaction.h"
6. #include "user.h"
7. #include "string"
8. #include "string.h"
9. #include "conio.h"
10. #include "fstream"
11. #include <iostream>
12. **using** **namespace** std;
14. **int** main()
15. {
16. system("CLS");
17. cout<<endl<<"================================================================================";
18. **HANDLE** m\_hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);
19. SetConsoleTextAttribute (  m\_hConsole,
20. BACKGROUND\_RED |
21. BACKGROUND\_GREEN |
22. BACKGROUND\_BLUE );
23. std::cout <<"      B A H R I A - U N I V E R S I T Y - S P E N D I N G - T R A C K E R       ";
24. SetConsoleTextAttribute ( m\_hConsole,
25. FOREGROUND\_RED |
26. FOREGROUND\_GREEN |
27. FOREGROUND\_BLUE );
28. cout<<"================================================================================"<<endl;
30. **int** choice1, choice2, choice3; //variables to get user choices
31. user \*u; //pointer to user class
32. u = **new** user; //user object
33. transaction \*t = **new** transaction; //dynamic memory allocation to pointer of transaction class
35. cout<<"Enter Mode:"<<endl;
36. cout<<"1. Log In."<<endl;
37. cout<<"2. Sign Up."<<endl;
38. cin>>choice1; //gets mode input
39. u->getData(choice1); //gets login or sign up credentials from user
41. **do**
42. {
43. system("CLS");
44. cout<<endl<<"================================================================================";
45. **HANDLE** m\_hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);
46. SetConsoleTextAttribute (  m\_hConsole,
47. BACKGROUND\_RED |
48. BACKGROUND\_GREEN |
49. BACKGROUND\_BLUE );
50. std::cout <<"      B A H R I A - U N I V E R S I T Y - S P E N D I N G - T R A C K E R       ";
51. SetConsoleTextAttribute ( m\_hConsole,
52. FOREGROUND\_RED |
53. FOREGROUND\_GREEN |
54. FOREGROUND\_BLUE );
55. cout<<"================================================================================"<<endl;
57. cout<<"Enter Operation:"<<endl;
58. cout<<"1. Add Income."<<endl;
59. cout<<"2. Add Expense."<<endl;
60. cout<<"3. Display Account Balance."<<endl;
61. cout<<"4. Display All Transactions."<<endl;
62. cout<<"5. Genrate Text File Report."<<endl;
63. cout<<"6. Show Summary(Graph)."<<endl;
64. cout<<"7. Exit."<<endl;
65. cin>>choice2; //gets operation choice
67. **if**(choice2 == 1)
68. {
69. system("CLS");
70. cout<<endl<<"================================================================================";
71. **HANDLE** m\_hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);
72. SetConsoleTextAttribute (  m\_hConsole,
73. BACKGROUND\_RED |
74. BACKGROUND\_GREEN |
75. BACKGROUND\_BLUE );
76. std::cout <<"      B A H R I A - U N I V E R S I T Y - S P E N D I N G - T R A C K E R       ";
77. SetConsoleTextAttribute ( m\_hConsole,
78. FOREGROUND\_RED |
79. FOREGROUND\_GREEN |
80. FOREGROUND\_BLUE );
81. cout<<"================================================================================"<<endl;
82. t->getAmount(u->getUsername(), u->getPassword(), 1); //files of the user will be created on its username.txt(final balance) and password.txt(transactions details)
83. }
84. **else** **if** (choice2 == 2)
85. {
86. system("CLS");
87. cout<<endl<<"================================================================================";
88. **HANDLE** m\_hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);
89. SetConsoleTextAttribute (  m\_hConsole,
90. BACKGROUND\_RED |
91. BACKGROUND\_GREEN |
92. BACKGROUND\_BLUE );
93. std::cout <<"      B A H R I A - U N I V E R S I T Y - S P E N D I N G - T R A C K E R       ";
94. SetConsoleTextAttribute ( m\_hConsole,
95. FOREGROUND\_RED |
96. FOREGROUND\_GREEN |
97. FOREGROUND\_BLUE );
98. cout<<"================================================================================"<<endl;
100. t->getAmount(u->getUsername(), u->getPassword(), 2); //files of the user will be created on its username.txt(final balance) and password.txt(transactions details)
101. }
102. **else** **if**(choice2 == 3)
103. {
104. system("CLS");
105. cout<<endl<<"================================================================================";
106. **HANDLE** m\_hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);
107. SetConsoleTextAttribute (  m\_hConsole,
108. BACKGROUND\_RED |
109. BACKGROUND\_GREEN |
110. BACKGROUND\_BLUE );
111. std::cout <<"      B A H R I A - U N I V E R S I T Y - S P E N D I N G - T R A C K E R       ";
112. SetConsoleTextAttribute ( m\_hConsole,
113. FOREGROUND\_RED |
114. FOREGROUND\_GREEN |
115. FOREGROUND\_BLUE );
116. cout<<"================================================================================"<<endl;
118. t->finalBalance(u->getUsername()); //to display final balance in account
119. }
120. **else** **if**(choice2 == 4)
121. {
122. system("CLS");
123. cout<<endl<<"================================================================================";
124. **HANDLE** m\_hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);
125. SetConsoleTextAttribute (  m\_hConsole,
126. BACKGROUND\_RED |
127. BACKGROUND\_GREEN |
128. BACKGROUND\_BLUE );
129. std::cout <<"      B A H R I A - U N I V E R S I T Y - S P E N D I N G - T R A C K E R       ";
130. SetConsoleTextAttribute ( m\_hConsole,
131. FOREGROUND\_RED |
132. FOREGROUND\_GREEN |
133. FOREGROUND\_BLUE );
134. cout<<"================================================================================"<<endl;
136. t->displayTransaction(u->getPassword());
137. }
138. **else** **if**(choice2 == 5)
139. {
140. system("CLS");
141. cout<<endl<<"================================================================================";
142. **HANDLE** m\_hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);
143. SetConsoleTextAttribute (  m\_hConsole,
144. BACKGROUND\_RED |
145. BACKGROUND\_GREEN |
146. BACKGROUND\_BLUE );
147. std::cout <<"      B A H R I A - U N I V E R S I T Y - S P E N D I N G - T R A C K E R       ";
148. SetConsoleTextAttribute ( m\_hConsole,
149. FOREGROUND\_RED |
150. FOREGROUND\_GREEN |
151. FOREGROUND\_BLUE );
152. cout<<"================================================================================"<<endl;
154. t->generateReport(u->getPassword());
155. }
156. **else** **if**(choice2 == 6)
157. {
158. system("CLS");
159. cout<<endl<<"================================================================================";
160. **HANDLE** m\_hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);
161. SetConsoleTextAttribute (  m\_hConsole,
162. BACKGROUND\_RED |
163. BACKGROUND\_GREEN |
164. BACKGROUND\_BLUE );
165. std::cout <<"      B A H R I A - U N I V E R S I T Y - S P E N D I N G - T R A C K E R       ";
166. SetConsoleTextAttribute ( m\_hConsole,
167. FOREGROUND\_RED |
168. FOREGROUND\_GREEN |
169. FOREGROUND\_BLUE );
170. cout<<"================================================================================"<<endl;
172. t->summary(u->getPassword());
173. }
174. **else** **if**(choice2 == 7)
175. {
176. **break**;
177. }
178. **else**
179. {
180. system("CLS");
181. cout<<endl<<"================================================================================";
182. **HANDLE** m\_hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);
183. SetConsoleTextAttribute (  m\_hConsole,
184. BACKGROUND\_RED |
185. BACKGROUND\_GREEN |
186. BACKGROUND\_BLUE );
187. std::cout <<"      B A H R I A - U N I V E R S I T Y - S P E N D I N G - T R A C K E R       ";
188. SetConsoleTextAttribute ( m\_hConsole,
189. FOREGROUND\_RED |
190. FOREGROUND\_GREEN |
191. FOREGROUND\_BLUE );
192. cout<<"================================================================================"<<endl;
194. cout<<"SORRY!!! INVALID CHOICE SELECTED."<<endl;
195. }
197. cout<<endl;
198. cout<<endl<<"Do you want to do operation again?"<<endl;
199. cout<<"1. YES."<<endl;
200. cout<<"2. NO."<<endl;
201. cin>>choice3;
202. }
203. **while**(choice3 == 1);
205. getch();
206. }