**OBJECTIVES:-**

1. To learn the basic concepts of SQL Functions.
2. To learn the how database is connect.
3. To learn to how to store values permanently.
4. To learn the how to convert string into data type.
5. To learn how to make Constant String Pointer.

**Introduction**

* The Project is basically how to store the data in Database .
* We Have used SQLite to store the Data.
* We have used various function like atoi(),c\_str(),sql\_open() etc.
* We have performed the Insert, Update, Delete and Display Operation on the Database.

**Various Problem Occurred**

1. First Problem Occurred how to convert the string into constant string

For this we have used cstdlib.h header file function c\_str( ) in which the string is convert into constant char pointer type. Which is allowed by the sql to insert.

1. Next When we try to perform the operation like addition and Division on the data it is of

string type So we are not able to perform it.

For this we have used a function atoi() which converts the string into the numeric type.

1. When we try to copy the data after the operation then it is of numeric type so we are not able

To perform it.

For this we have used the ostringstream class in which we first send the data to it’s object.

And then we have used a function str() which converts the data into string type.

**HEADER FILES :**

1. **Stdio.h :-**

It’s A HEADER file which contain all the I/O functions like printf , scanf etc …

* The C programming language provides many standard library functions for file input and output. These functions make up the bulk of the C standard library header <stdio.h>.
* The first thing you will notice is the first line of the file, the #include "stdio.h" line. This is very much like the #define the Preprocessor , except that instead of a simple substitution, an entire file is read in at this point.
* The system will find the file named "stdio.h" and read its entire contents in, replacing this statement.
* The file named "stdio.h" must contain valid C source statements that can be compiled as part of a program.
* This particular file is composed of several standard #defines to define some of the standard I/O operations.
* The file is called a header file and you will find several different header files on the source disks that came with your C compiler.

* Each of the header files has a specific purpose and any or all of them can be included in any program.
* Your C compiler uses the double quote marks to indicate that the search for the "include" file will begin in the current directory, and if it not found there, the search will continue in the "include" directory as set up in the environment.
* It also uses the "less than" and "greater than" signs to indicate that the file search should begin in the directory specified in the environment.
* Most of the programs in this tutorial have the double quotes in the "include" statements. The next program uses the "<" and ">" to illustrate the usage.

**2 stdlib.h : -**

* It should be include before main line.
* The name stdlib stands for standard library
* stdlib.h is a utility functions such as string conversion routines, memory allocation routines,random number generator,etc.
* **stdlib.h** is the header of the **general purpose standard library** of C programming language which includes functions involving memory allocation, process control, conversions and others. It is compatible with C++ and is known as cstdlib in C++.
* The name "stdlib" stands for "standard library".

**3 iostream.h : -**

* It’s a set of classes, templates, and library functions that are part of the standard library in C++ that are used to perform input, output, and various formatting operations.
* The IO is from Input/Output (IO), and Stream is because it’s a (potentially endless) flow [or “stream”] of characters.
* Operations allow you to build and parse sequences of characters, use different sources and destinations, including strings themselves, all through a handy set of terse operators and flags.
* a header file which contains certain built in functions. Mostly used for cout, cin and cerr. In all the above mentioned functions c stands for console. So cout is console output, cin is console input. Consoles being display device and keyboards respectively.
* **.h**this is an extension or file type just like mp3 or jpg, .h signifies that iostream is a header file.
* In starting of every program we use iostream.h because C++ does not have the function to input and display the output function internally so we use iostream.h to input the data in C++ as well as output it on the screen.
* In starting of every program we use iostream.h because C++ does not have the function to input and display the output function internally so we use iostream.h to input the data in C++ as well as output it on the screen.

**4 Cstdlib.h :-**

The <cstdlib> header file declares a set of general purpose functions such as: atof() to convert string to Double. It also contains few mathematical functions. For example, abs() to find absolute value of a number.

**5 String :-**

C++ has in its definition a way to represent **sequence of characters as an object of class**. This class is called std:: string. String class stores the characters as a sequence of bytes with a functionality of allowing **access to single byte character**

In order to use the string data type, the C++ string header must be included at the top of the program. Also, you’ll need to include using namespace std; to make the short name string visible instead of requiring the cumbersome std::string. Thus, you would have the following #include's in your program in order to use the string type.

**5 Sqlite\_3.h :-**

* SQLite is an in-process library that implements a [self-contained](https://www.sqlite.org/selfcontained.html), [serverless](https://www.sqlite.org/serverless.html), [zero-configuration](https://www.sqlite.org/zeroconf.html), [transactional](https://www.sqlite.org/transactional.html) SQL database engine.
* The code for SQLite is in the [public domain](https://www.sqlite.org/copyright.html) and is thus free for use for any purpose, commercial or private. SQLite is the [most widely deployed](https://www.sqlite.org/mostdeployed.html) database in the world with more applications than we can count, including several [high-profile projects](https://www.sqlite.org/famous.html)
* SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file.
* The database [file format](https://www.sqlite.org/fileformat2.html) is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between [big-endian](http://en.wikipedia.org/wiki/Endianness) and [little-endian](http://en.wikipedia.org/wiki/Endianness)architectures. These features make SQLite a popular choice as an [Application File Format](https://www.sqlite.org/appfileformat.html).
* SQLite is a compact library. With all features enabled, the [library size](https://www.sqlite.org/footprint.html) can be less than 600KiB, depending on the target platform and compiler optimization settings.
* There is a tradeoff between memory usage and speed. SQLite generally runs faster the more memory you give it.
* SQLite is [very carefully tested](https://www.sqlite.org/testing.html) prior to every release and has a reputation for being very reliable. Most of the SQLite source code is devoted purely to testing and verification.

**Functions Used :**

1 **Callback() :**

* + A callback is a callable (see further down) accepted by a class or function, used to customize the current logic depending on that callback.
  + One reason to use callbacks is to write **generic** code which is independant from the logic in the called function and can be reused with different callbacks.
  + callback function is a function that is called through a function pointer. If you pass the pointer (address) of a function as an argument to another, when that pointer is used to call the function it points to it is said that a call back is made.

**Why Should You Use Callback Functions?**

Because they uncouple the caller from the callee. The caller doesn't care who the callee is; all it knows is that there is a callee with a certain prototype and probably some restriction (for instance, the returned value can be int, but certain values have certain meanings).

2 **Sqllite Open() :**

* This function open a connection to a new or existing SQLite database. The constructor for sqlite3.
* This routine opens a connection to an SQLite database file and returns a [database connection](https://www.sqlite.org/c3ref/sqlite3.html) object. This is often the first SQLite API call that an application makes and is a prerequisite for most other SQLite APIs. Many SQLite interfaces require a pointer to the [database connection](https://www.sqlite.org/c3ref/sqlite3.html) object as their first parameter and can be thought of as methods on the [database connection](https://www.sqlite.org/c3ref/sqlite3.html) object. This routine is the constructor for the [database connection](https://www.sqlite.org/c3ref/sqlite3.html) object.
* This routine opens a connection to an SQLite database file and returns a database connection object to be used by other SQLite routines .

**Syntax :**

sqlite3\_open(const char \*filename, sqlite3 \*\*ppDb)

**3 sqlite3close() :**

* This routine closes a database connection previously opened by a call to sqlite3\_open(). All prepared statements associated with the connection should be finalized prior to closing the connection.
* If any queries remain that have not been finalized, sqlite3\_close() will return SQLITE\_BUSY with the error message Unable to close due to unfinalized statements.

**Syntax :**

sqlite3\_close(sqlite3\*)

**4 sqlite3\_exec() :**

* This routine provides a quick, easy way to execute SQL commands provided by sql argument which can consist of more than one SQL command.
* Here, the first argument *sqlite3* is an open database object, *sqlite\_callback* is a call back for which *data* is the 1st argument and errmsg will be returned to capture any error raised by the routine.
* SQLite3\_exec() routine parses and executes every command given in the **sql** argument until it reaches the end of the string or encounters an error.

**Syntax :**

sqlite3\_exec(sqlite3\*, const char \*sql, sqlite\_callback, void \*data, char **\*\*errmsg)**

**5 fprintf() :**

* The C library function **int fprintf(FILE \*stream, const char \*format, ...)**sends formatted output to a stream.
* The fprintf() function writes the string pointed to by format to the stream stream. The string format may contain format specifiers starting with % which are replaced by the values of variables that are passed to the fprintf() function as additional arguments.
* If successful, the fprintf() function returns number of characters written. On failure it returns a negative value.

**Syntax :**

int fprintf(FILE\* stream, const char\* format, ...);

## **fprintf() Parameters :**

* stream: Pointer to an output file stream where the contents are written.
* format: Pointer to a null terminated string that is written to the file stream. It consists of characters along with optional format specifiers starting with %.
  + The format specifiers are replaced by the values of respective variables that follows the format string.
  + The format specifier has the following parts:
  + A leading % sign
  + Flags: Optional one or more flags that modifies the conversion behavior.
    - - : Left justify the result within the field. By default it is right justified.
    - + : The sign of the result is attached to the beginning of the value, even for positive results.
    - Space: If there is no sign, a space is attached to the beginning of the result.
    - # : An alternative form of the conversion is performed.
    - 0 : It is used for integer and floating point number. Leading zeros are used to pad the numbers instead of space.
  + Width: An optional \* or integer value used to specify minimum width field.
  + Precision : An optional field consisting of a . followed by \* or integer or nothing to specify the precision.
  + Length : An optional length modifier that specifies the size of the argument

**6 sql3\_free() :**

* The SQLite core uses these three routines for all of its own internal memory allocation needs. "Core" in the previous sentence does not include operating-system specific VFS implementation. The Windows VFS uses native malloc() and free() for some operations.
* Calling sqlite3\_free() with a pointer previously returned by sqlite3\_malloc() or sqlite3\_realloc() releases that memory so that it might be reused. The sqlite3\_free() routine is a no-op if is called with a NULL pointer. Passing a NULL pointer to sqlite3\_free() is harmless. After being freed, memory should neither be read nor written. Even reading previously freed memory might result in a segmentation fault or other severe error. Memory corruption, a segmentation fault, or other severe error might result if sqlite3\_free() is called with a non-NULL pointer that was not obtained from sqlite3\_malloc() or sqlite3\_realloc().

**7 atoi() :**

* **atoi** is a function in the C programming language that converts a string into an integer numerical representation. atoi stands for *ASCII to integer*. It is included in the C standard library header file stdlib.h.
* The str argument is a string, represented by an array of characters, containing the characters of a signed integer number. The string must be null-terminated. When atoi encounters a string with no numerical sequence, it returns zero (0).
* There are several variants of the **atoi** function, **atol**, **atof** and **atoll** , which are used to convert a string into a long, double, or long long type, respectively. The **atoll** was formerly known as **atoq** and was included into C99.
* It is impossible to tell whether the string holds valid sequence of digits that represents the number 0 or invalid number as the function returns 0 in both cases. The newer function strtol does not have this deficiency.
* **atoi** is neither thread-safe, nor async-cancel safe on some operating systems.[[1]](https://en.wikibooks.org/wiki/C_Programming/stdlib.h/atoi#cite_note-codecogs_atoi-1)
* Also, **atoi** only converts base ten ascii values (this may also be a benefit depending on perspective). strtol and other functions support alternate bases such as hexadecimal and octal.

**Syntax :**

int atoi(const char \*str);

**8 c\_str() :**

It returns a pointer to an array that contains a null-terminated sequence of characters (i.e., a C-string) representing the current value of the string object. It does not have any parameter .

It returns a pointer to an array that contains a null-terminated sequence of characters (i.e., a C-string) representing the current value of the string object

If you have a C++ <string> you need to give the contents to a C routine, you need to convert it to the correct type using the c\_str() method to get the right format of data.

**Syntax :**

const char\* c\_str() const;

**9) Sqlite3\_errmsg() :**

The sqlite3\_errmsg() and sqlite3\_errmsg16() return English-language text that describes the error, as either UTF-8 or UTF-16 respectively. Memory to hold the error message string is managed internally. The application does not need to worry about freeing the result. However, the error string might be overwritten or deallocated by subsequent calls to other SQLite interface functions.

If the most recent sqlite3\_\* API call associated with database connection D failed, then the sqlite3\_errcode(D) interface returns the numeric result code or extended result code for that API call. The sqlite3\_extended\_errcode() interface is the same except that it always returns the extended result code even when extended result codes are disabled.

**Syntax :**

int sqlite3\_errcode(sqlite3 \*db);

int sqlite3\_extended\_errcode(sqlite3 \*db);

const char \*sqlite3\_errmsg(sqlite3\*);

const void \*sqlite3\_errmsg16(sqlite3\*);

const char \*sqlite3\_errstr(int);

**Conclusion**

We understand how to link the C++ Program with SQLite.

We observe that how the data is important to store.

We observe that storing the data instead on file it is much better to store the data in the database

So we can perform various operation easily.

We understand how to tacal with the problem occurred using various methods of the class.

**Code:-**

#include <stdio.h>

#include <stdlib.h>

#include<iostream>

#include <sstream>

#include<string>

#include <sqlite3.h>

#include<cstdlib>

using namespace std;

/\*Global Variable \*/

sqlite3 \*db;

char \*zErrMsg = 0;

int rc;

char \*sql;

const char\* data = "Callback function called";

static int callback(void \*NotUsed, int argc, char \*\*argv, char \*\*azColName)

{

int i;

for(i = 0; i<argc; i++)

{

printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");

}

printf("\n");

return 0;

}

class Student

{

protected:

string roll,clas;

string name,phone;

};

class Result:public Student

{

string m1,m2,m3;

string total,avg;

public:

void accept();

void display();

void update();

void deletep();

void create();

};

void Result::create()

{

/\* Create SQL statement \*/

sql = "CREATE TABLE STUDENT(" \

"ROLL\_NO INT PRIMARY KEY ," \

"NAME CHAR(20)," \

"CLASS INT NOT NULL," \

"PHONENO BIGINT," \

"MATH CHAR(20),"\

"SCIENCE CHAR(20),"\

"ENGLISH CHAR(20),"\

"TOTAL INT ,"\

"AVERAGE FLOAT );";

/\* Execute SQL statement \*/

rc = sqlite3\_exec(db, sql, callback, 0, &zErrMsg);

if( rc != SQLITE\_OK )

{

fprintf(stderr, "SQL error: %s\n", zErrMsg);

sqlite3\_free(zErrMsg);

}

else

{

fprintf(stdout, "Table created successfully\n");

}

}

void Result::accept()

{

cout<<"\n\n Enter Roll number and Name: ";

cin>>roll>>name;

cout<<"\n\n Enter class and phone number: ";

cin>>clas>>phone;

cout<<"\n\n Enter 3 subject marks ";

cout<<"\n Order Math,Science and English: ";

cin>>m1>>m2>>m3;

int temp\_m1 = atoi(m1.c\_str());

int temp\_m2 = atoi(m2.c\_str());

int temp\_m3 = atoi(m3.c\_str());

int temp\_total=temp\_m1+temp\_m2+temp\_m3;

float temp\_avg=temp\_total/3.0;

// declaring output string stream

ostringstream str1,str2;

str1 << temp\_total;

str2 << temp\_avg;

total=str1.str();

avg=str2.str();

string query="insert into student values('"+roll+"','"+name+"','"+clas+"','"+phone+"','"+m1+"','"+m2+"','"+m3+"','"+total+"','"+avg+"');";

const char\* q=query.c\_str();

/\* Execute SQL statement \*/

rc = sqlite3\_exec(db,q, callback, 0, &zErrMsg);

if( rc != SQLITE\_OK )

{

fprintf(stderr, "SQL error: %s\n", zErrMsg);

sqlite3\_free(zErrMsg);

}

else

{

fprintf(stdout, "Records created successfully\n");

}

}

void Result::display()

{

/\* Create SQL statement \*/

sql = "SELECT \* from STUDENT";

/\* Execute SQL statement \*/

rc = sqlite3\_exec(db, sql, callback, (void\*)data, &zErrMsg);

if( rc != SQLITE\_OK )

{

fprintf(stderr, "SQL error: %s\n", zErrMsg);

sqlite3\_free(zErrMsg);

}

else

{

fprintf(stdout, "Operation done successfully\n");

}

}

void Result:: update()

{

cout<<"\n\n Enter Class to update enter roll no where to update :";

cin>>clas>>roll;

/\* Create merged SQL statement \*/

string query="update student set class='"+clas+"' where roll\_no='"+roll+"' "; \

"SELECT \* from STUDENT";

const char\* q=query.c\_str();

/\* Execute SQL statement \*/

rc = sqlite3\_exec(db, q, callback, (void\*)data, &zErrMsg);

if( rc != SQLITE\_OK )

{

fprintf(stderr, "SQL error: %s\n", zErrMsg);

sqlite3\_free(zErrMsg);

}

else

{

fprintf(stdout, "Operation done successfully\n");

}

sqlite3\_close(db);

}

void Result:: deletep()

{

cout<<"\n\n Enter Student Roll Number to Delete: ";

cin>>roll;

/\* Create merged SQL statement \*/

string query="delete from student where roll\_no='"+roll+"';"\

"SELECT \* from STUDENT";

const char\* q=query.c\_str();

/\* Execute SQL statement \*/

rc = sqlite3\_exec(db, q, callback, (void\*)data, &zErrMsg);

if( rc != SQLITE\_OK )

{

fprintf(stderr, "SQL error: %s\n", zErrMsg);

sqlite3\_free(zErrMsg);

}

else

{

fprintf(stdout, "Operation done successfully\n");

}

}

int main(int argc, char\* argv[])

{

int ch;

Result R;

/\* Open database \*/

rc = sqlite3\_open("test.db", &db);

if( rc )

{

fprintf(stderr, "Can't open database: %s\n", sqlite3\_errmsg(db));

return(0);

}

else

{

fprintf(stdout, "Opened database successfully\n");

}

while(1)

{

cout<<"\n\n Menu: ";

cout<<"\n 1.Create Table ";

cout<<"\n 2.Update ";

cout<<"\n 3.Delete ";

cout<<"\n 4.Display ";

cout<<"\n 5.Insert";

cout<<"\n 6.Exit";

cout<<"\n\n Enter your Choice(1/2/3/4/5): ";

cin>>ch;

switch(ch)

{

case 1: R.create();

break;

case 2: R.update();

break;

case 3: R.deletep();

break;

case 4: R.display();

break;

case 5: R.accept();

break;

case 6: exit(0);

default: cout<<"\n\n Wrong Choice. ";

}

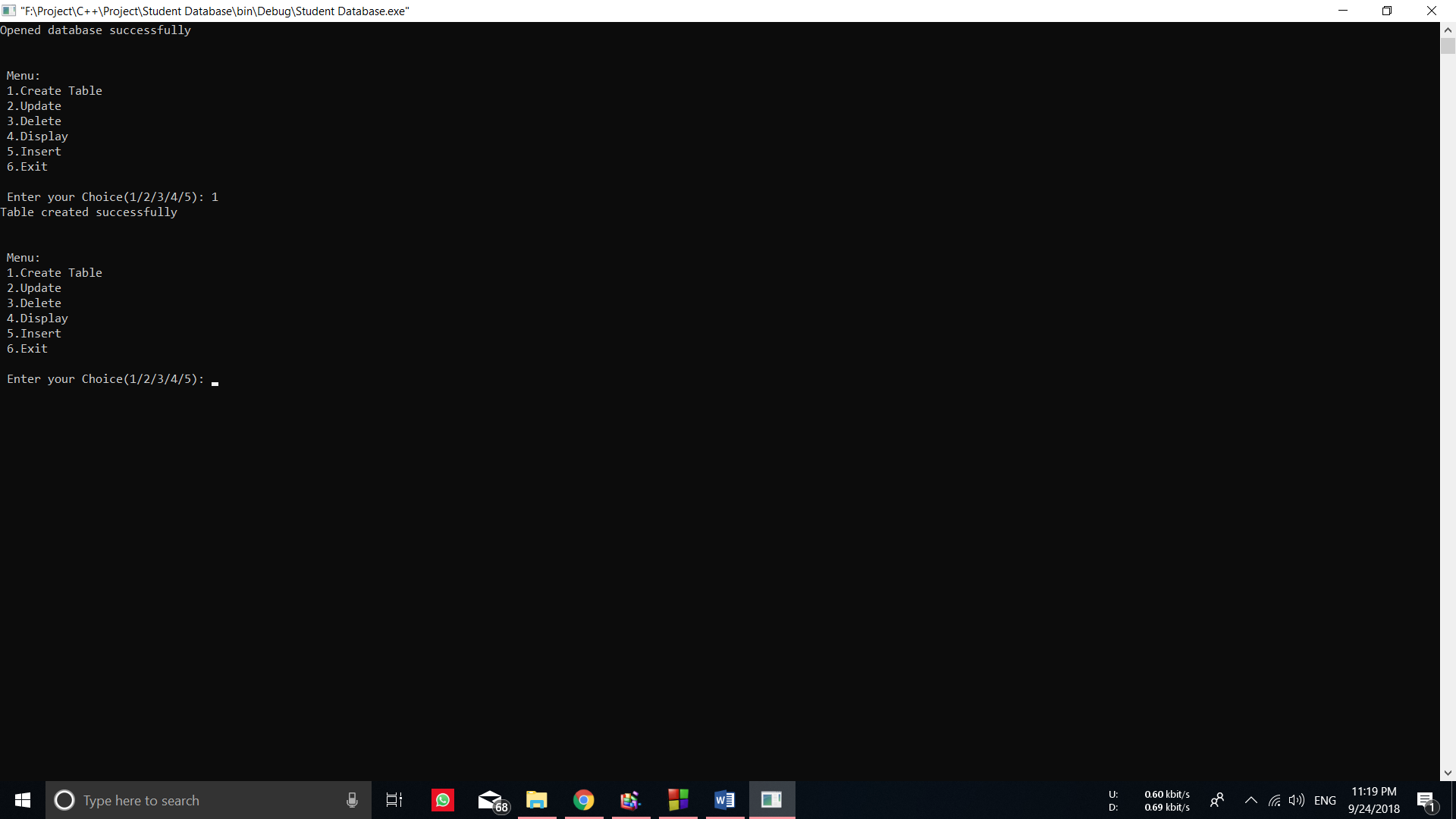
}

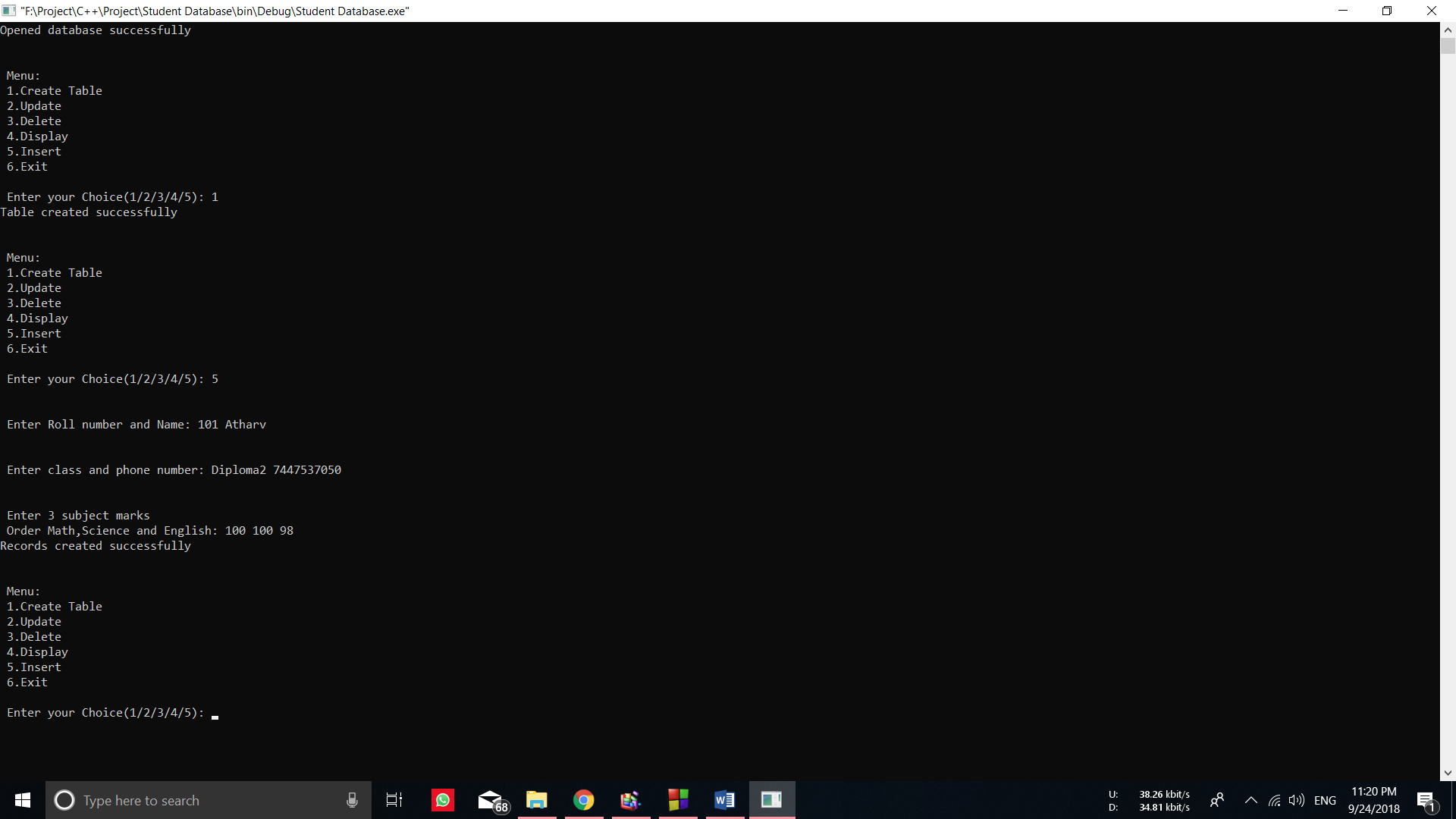
sqlite3\_close(db);

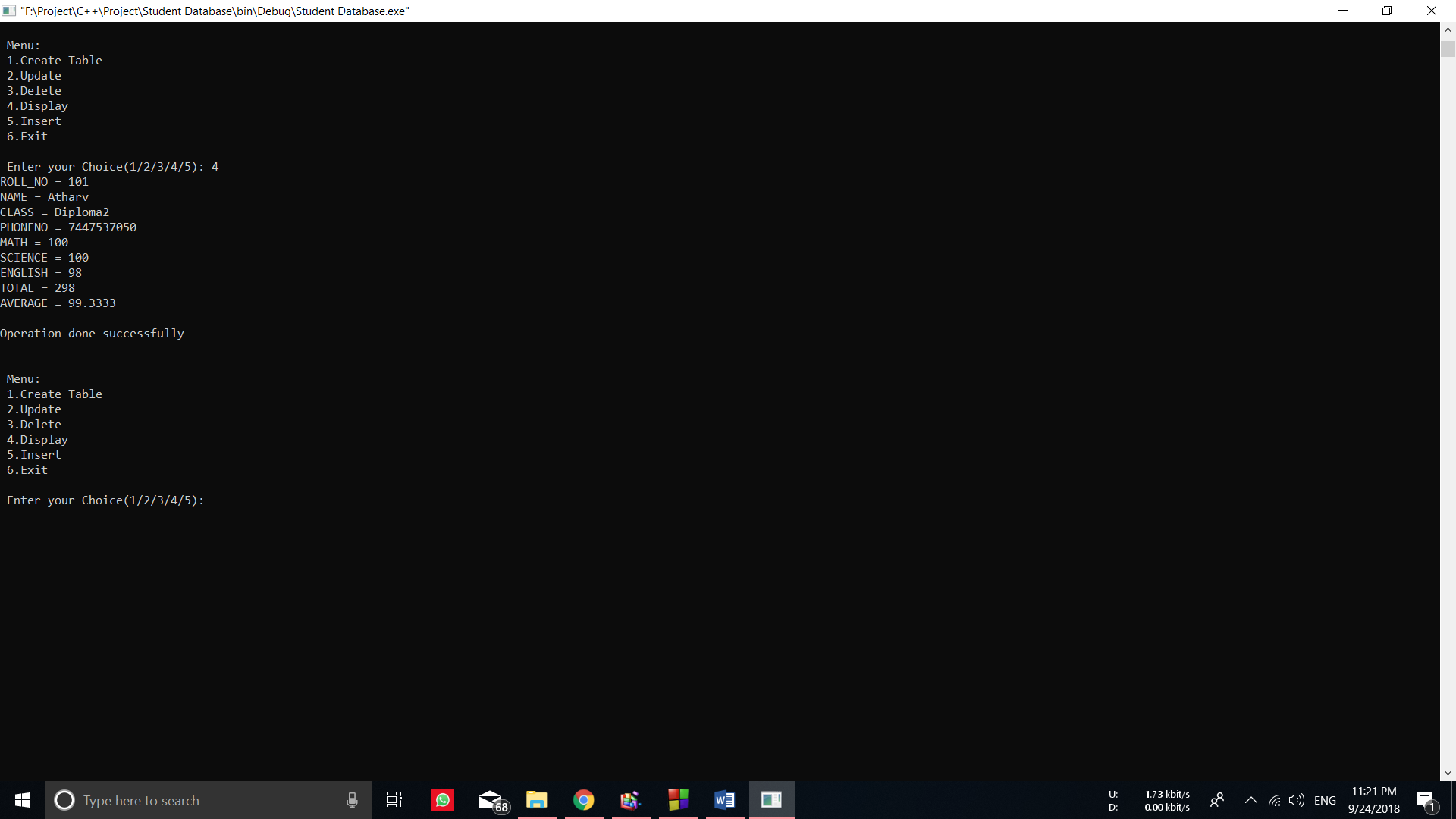
return 0;

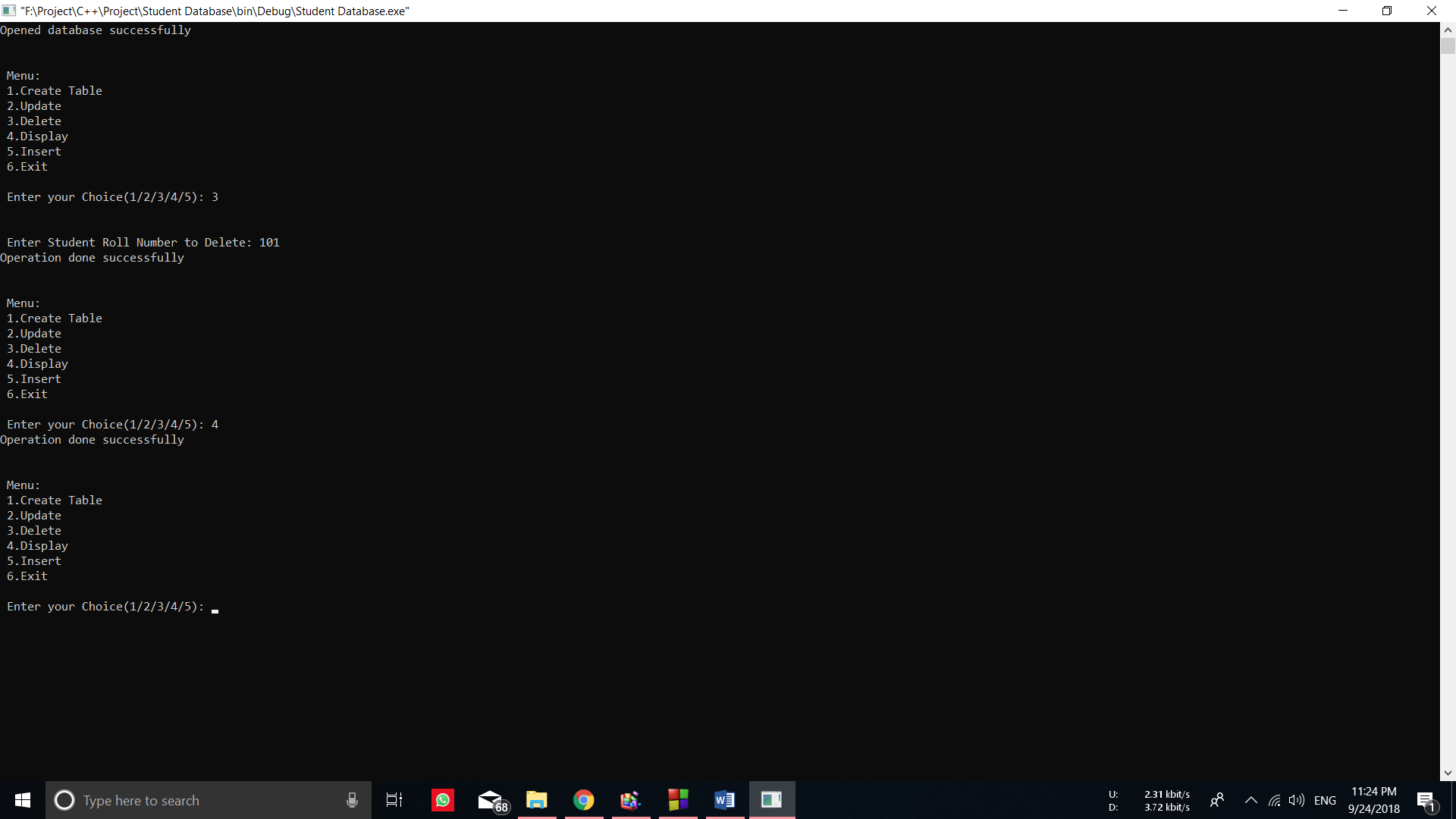
}

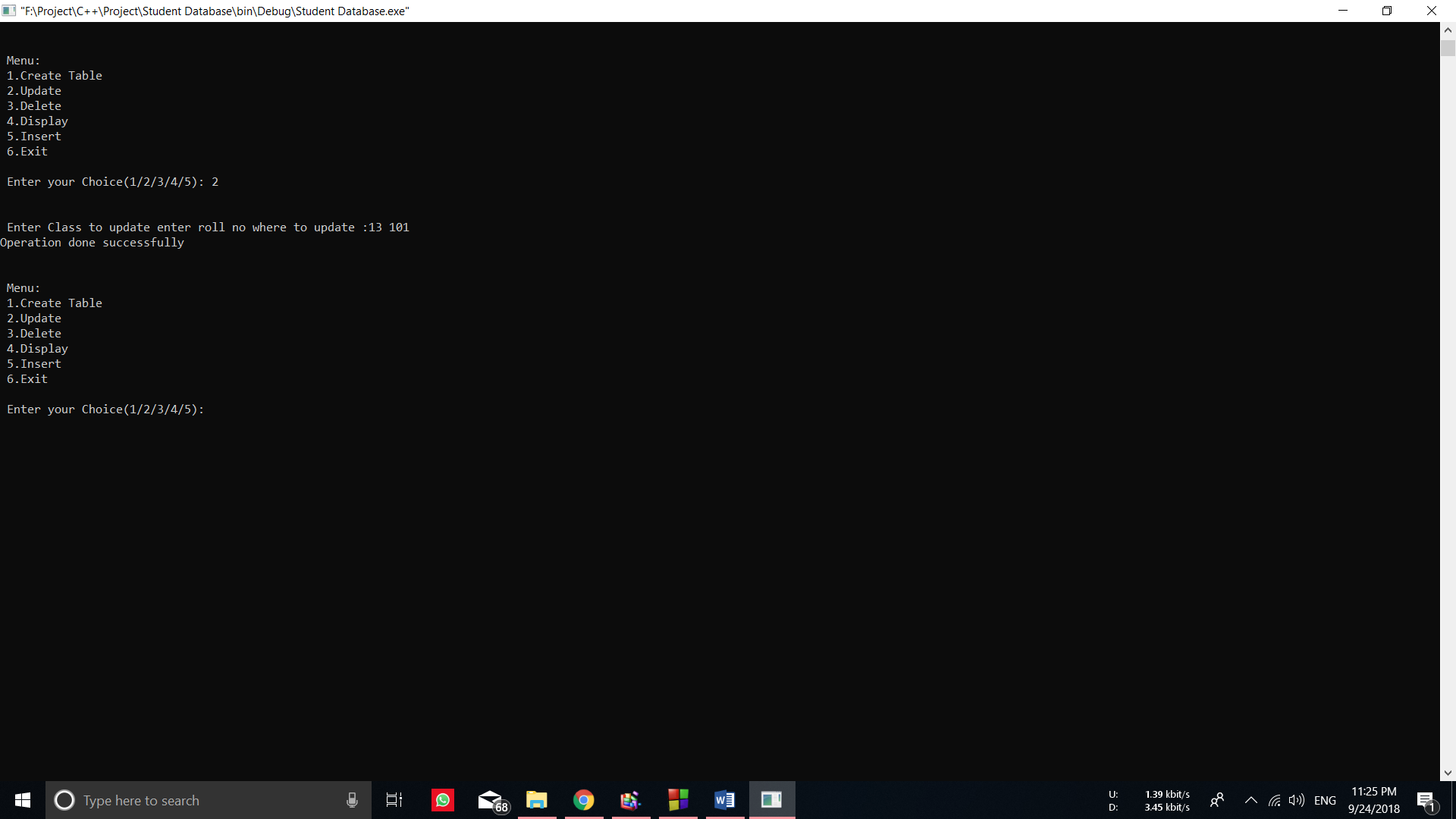
**Output:**











**EVALUATION SHEET FOR MICRO PROJECT**

**(Academic Year: 2018-19)**

**Name of the Student: Roll No:**

**Course: Object Oriented Programming Using C++(22316) Course Code: CO 3I**

**Title of the Project: Student Database with SQL Lite**

**Cos addressed by Micro Project:**

1. Formulate correct coordinates on screen
2. Summarize the object.
3. Use relevant functions.

**Major learning outcomes achieved by students by doing the project:**

1. **Practical outcome:**
2. Learnt to create objects in C.
3. **Unit outcomes in Cognitive Domain:**

1) Use various functions of graphics in C

1. **Out comes in affective domain:**

1) Function as a team member

2) Follow ethics

**Comments/suggestions about teamwork/leadership/interpersonal communications (if any):**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Roll No** | **Student Name** | **Marks out of 06 for performance in group activity (D5 Column 08)** | **Marks out of 04 for performance in oral or presentation (D5 Column 09)** | **Total out of 40** |
|  |  |  |  |  |

**Name & Signature of Faculty:**