COMP 3311 Database Management Systems Spring 2020

Lab 8. Programming with ODBC 2

Objectives of the Lab

- After this lab, you should be able to:
  - know about the different data-types of ODBC (which is important in calling the functions),
  - issue prepared statements using the ODBC interface,
  - check for error messages under ODBC.

Reminder: connecting to ODBC 1

- You have seen how to connect to the Oracle server through the ODBC interface in Lab 7.
- Four steps are involved in connecting to the Oracle through the ODBC interface:
  - Include the proper headers (<sql.h>, <sqlext.h>) to the C program,
  - Initialize ODBC environment,
  - Allocate a connection handle,
  - Connect to the data source corresponds to the Oracle server.

Reminder: connecting to ODBC 2

- To initialize the ODBC environment:

```
HENV   henv;
/* Allocate environment handle */
SQLAllocEnv( &henv);
```

- To allocate a connection handle:

```
HDBC   hdbc;
/* Allocate connection handle */
SQLAllocConnect(henv, &hdbc);
```

- To call the SQLConnect() function:

```
SQLConnectA(hdbc, (SQLCHAR*) "comp3311.cse.ust.hk",
SQL_NTS, (SQLCHAR*) argv[1], SQL_NTS, (SQLCHAR*)
argv[2], SQL_NTS);
```

This time you need to enter your id and password as input to execute the connect function.

ODBC Data types 1

- To enable real ODBC programming, one needs to be able to declare ODBC data types.
- ODBC defines two sets of data types:
  - C data types – indicate the data type of the data stored in the local variables of the ODBC programs.
  - SQL data types – indicate the data type of data stored at the data source (i.e. the DataBase Management System)

ODBC Data types 2

- Some common C data types

C type identifier (i.e. parameter passed to SQLBindCol and SQLGetData functions to specify target variable datatype)	ODBC C typedef (define variables in the program	Corresponding C data type
SQL_C_CHAR	SQLCHAR *	unsigned char *
SQL_C_WCHAR	SQLWCHAR *	wchar_t *
SQL_C_SSHORT	SQLSMALLINT	short int
SQL_C_USHORT	SQLUSMALLINT	unsigned short int
SQL_C_SLONG	SQLINTEGER	long int
SQL_C_ULONG	SQLUINTEGER	unsigned long int
SQL_C_FLOAT	SQLREAL	float
SQL_C_DOUBLE	SQLDOUBLE, SQLFLOAT	double

ODBC Data types 3

- Some common SQL data types

SQL type identifier (i.e. the SQL data type of the data being stored in the DBMS)	Actual SQL data type	Type description
SQL_CHAR	CHAR(n)	Character string of length n
SQL_VARCHAR	VARCHAR(n)	Variable length character string upto n characters
SQL_DECIMAL	DECIMAL(p,s)	Signed numeric value with precision of at least p and scale of s (p significant digits and s digits after the decimal point, p<=15)
SQL_NUMERIC	NUMERIC(p,s)	Signed numeric value with precision of exactly p and scale of s (p<=15)
SQL_SMALLINT	SMALLINT	Numeric value with precision 5 and scale 0
SQL_INTEGER	INTEGER	Numeric value with precision 10 and scale 0
SQL_FLOAT	FLOAT(p)	Signed numeric value with a binary precision of at least p
SQL_DOUBLE	DOUBLE PRECISION	Signed numeric value with a binary precision 53.

The prepared statement 1

- To Prepare and execute a SQL statement, one needs to:
  - call SQLPrepare() function to prepare the statement (pre-compiled at the server to improve efficiency),
  - call the SQLBindParameter() function to set the value(s) of the parameter(s),
  - call SQLExecute() function to execute the statement.

The prepared statement 2

- The SQL statement could contain place-holders which indicate values obtained from the program during the execution.
- The SQLBindParameter() function binds local variables to the place-holders and specify the data types of the variables and the columns associated with the parameters.

The prepared statement 3

- To call SQLPrepare(), one needs to pass a statement handle and the SQL query as the parameters, a question mark in the SQL query indicates the location of a place-holder:

```
SQLPrepareA(hstmt, (SQLCHAR*) "SELECT room_number FROM departments WHERE department_id=?", SQL_NTS);
```

- The possible return codes of SQLPrepare() are SQL_SUCCESS, SQL_ERROR, SQL_SUCCESS_WITH_INFO, and SQL_INVALID_HANDLE

The prepared statement 4

- To bind parameters to the placeholders in the SQL query, one needs the SQLBindParameter() function. The following example binds the string deptid to the question mark on the last slide:

```
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 0, 0, deptid, 50, &deptid_n);
```

- The possible return codes of SQLBindParameter() are SQL_SUCCESS, SQL_ERROR, SQL_SUCCESS_WITH_INFO, and SQL_INVALID_HANDLE

The prepared statement 5

- The full syntax of calling the SQLBindParameter() function:

```
SQLBindParameter( SQLHSTMT
 StatementHandle, SQLUSMALLINT
 ParameterNumber, SQLSMALLINT
 InputOutputType, SQLSMALLINT ValueType,
 SQLSMALLINT ParameterType, SQLULEN
 ColumnSize, SQLSMALLINT DecimalDigits,
 SQLPOINTER ParameterValuePtr, SQLLEN
 BufferLength, SQLLEN * StrLen_or_IndPtr);
```

The prepared statement 6

- ParameterNumber indicates the particular placeholder (question mark) you want the local variable to be bound with.
- InputOutputType indicates the type of the variable to be bound. Possible values for this parameter are :
  - SQL_PARAM_INPUT, SQL_PARAM_OUTPUT, SQL_PARAM_OUTPUT_STREAM, SQL_PARAM_INPUT_OUTPUT and SQL_PARAM_INPUT_OUTPUT_STREAM
  - (see http://msdn.microsoft.com/en-us/library/ms710963(v=VS.85).aspx for the details)
- ValueType indicates the C data type of the parameter (see slide 6 for the list of data types)
- ParameterType indicates the SQL data type of the parameter (see slide 7 for the list)
- ColumnSize indicates the max length of the parameter, it is used by the function if ParameterType equals to SQL_CHAR, SQL_GRAPHIC, SQL_DECIMAL or SQL_NUMERIC. For other parameter types, this parameter is unused/ignored.
- DecimalDigits indicates the number of digits of the parameter, it is used by the function if the ParameterType equals to SQL_DECIMAL,SQL_NUMERIC, or SQL_TIMESTAMP. For other parameter types, this parameter is unused/ignored.
- ParameterValuePtr is the pointer that points to a buffer that contains the actual data for the parameter, i.e. a local variable/array is been bound to the SQL parameter through this pointer.
- BufferLength indicates the length of the buffer pointed to by the ParameterValuePtr (50 for the previous example).
- strLen_or_IndPtr holds the length of the parameter value stored in *ParameterValuePtr , in the previous example we initialize it to SQL_NTS.

The prepared statement 7

- To execute the SQL statement, one needs to call the SQLExecute() function:

  SQLExecute(hstmt);

  Where hstmt is the statement handle.

- The possible return codes of SQLExecute() are SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_NO_DATA, SQL_INVALID_HANDLE, and SQL_PARAM_DATA_AVAILABLE.

The prepared statement 8

- Finally retrieve the result by binding the result to a local variable.

```
SQLBindCol(hstmt,1, SQL_C_SLONG,&room,1,&room_n);
```

- The possible return codes of SQLBindCol() are SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_NO_DATA, SQL_INVALID_HANDLE, and SQL_PARAM_DATA_AVAILABLE.

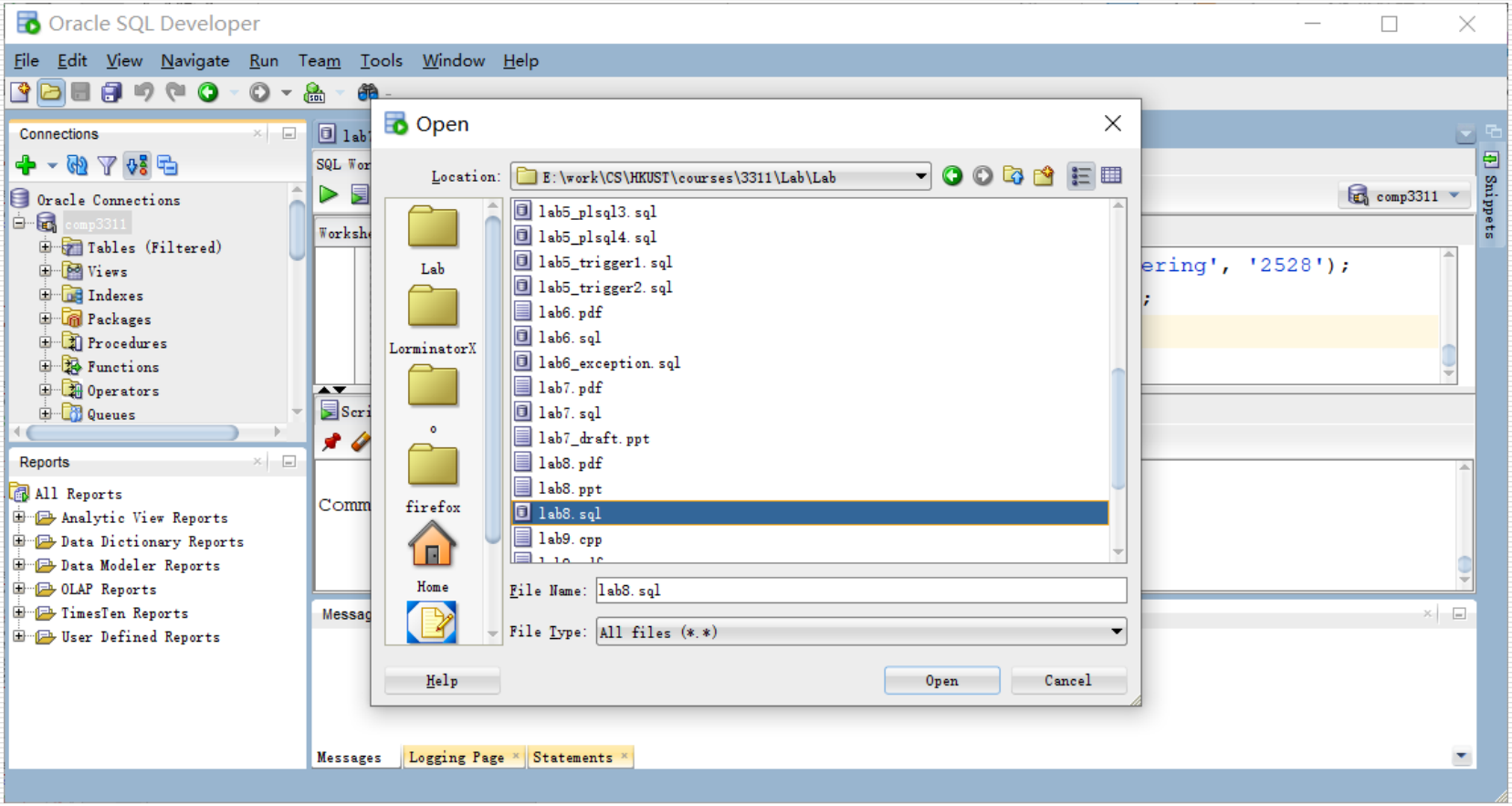
- The script file for building the database is available at[+]: http://course.cse.ust.hk/comp3311/labs/lab8.sql
- The complete piece of code is available at: http://course.cse.ust.hk/comp3311/labs/odbc2.cpp

[+] make sure you run "commit;" in SQL Developer after running the script. Before you "commit" (or "exit"), the data will not be written.

Running the example 1

- First, make sure the data source has been set up properly as "comp3311.cse.ust.hk" (refer to the "Steps to set up ODBC environment for visual studio" section of lab 7 for the detailed steps).
- Second, make sure you have ran the script lab8.sql in SQL Developer (and "commit" it by runnning "commit;" as shown in the "Building the SQL database 2" part of lab 7).
- Finally, start a new project and compile the code under Visual Studio (as shown in lab 7), and run the compiled program.

Running the example 1



Running the example 1

File Edit View Navigate Run Source Team Tools Window Help

Connections

Oracle Connections
- comp3311
  - Tables (Filtered)
  - Views
  - Indexes
  - Packages
  - Procedures
  - Functions
  - Operators
  - Queues

Reports
- All Reports
- Analytic View Reports
- Data Dictionary Reports
- Data Modeler Reports
- OLAP Reports
- TimesTen Reports
- User Defined Reports

lab7.sql | lab8.sql

SQL Worksheet | History

comp3311

Run Statement (Ctrl+Enter)

```
insert into departments values ('ELEC', 'Electronic Engineering', '2528');
insert into departments values ('BUS', 'Business', '4528');
commit;
```

Script Output

Task completed in 0.041 seconds

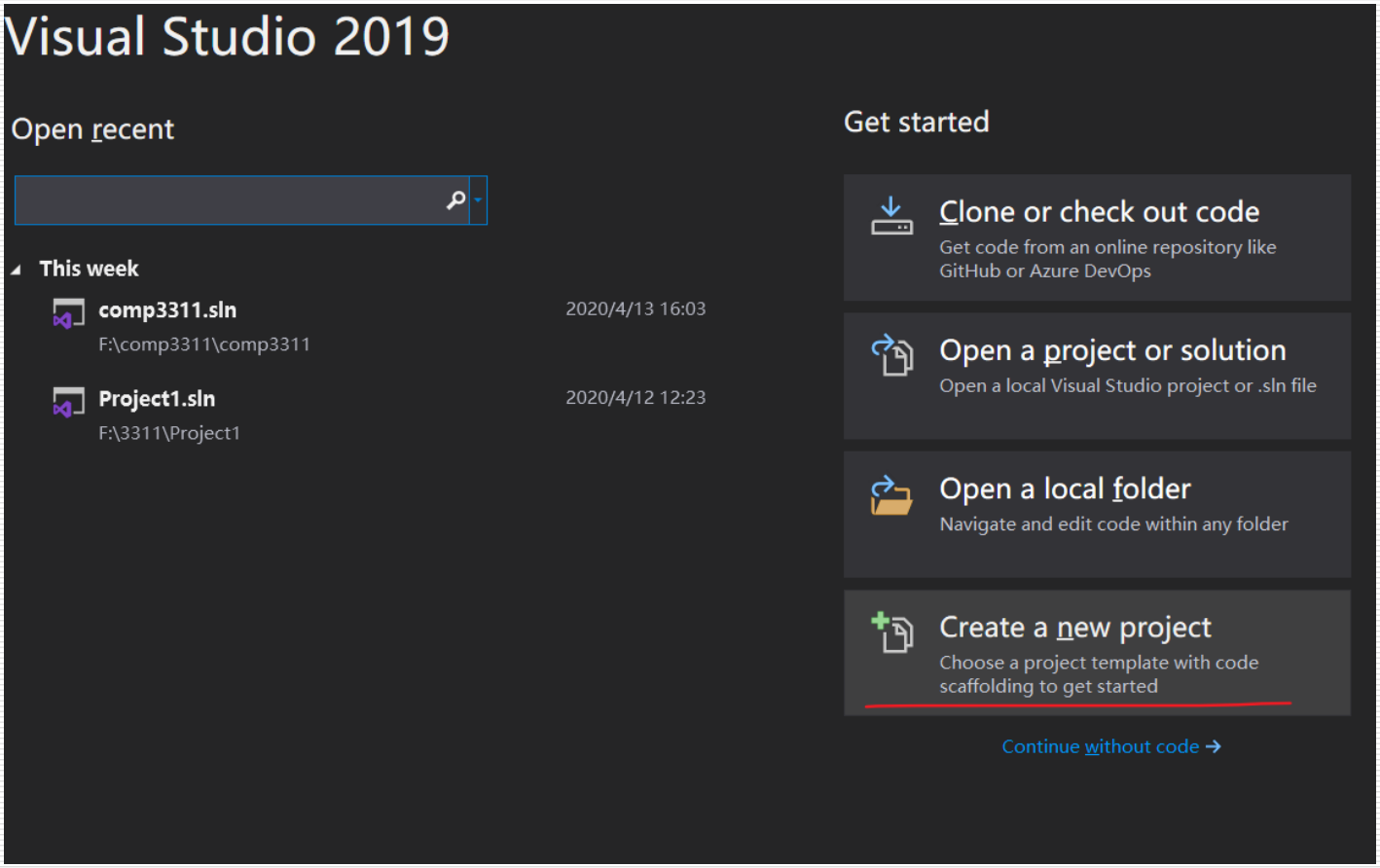
```
Commit complete.
```

Messages - Log

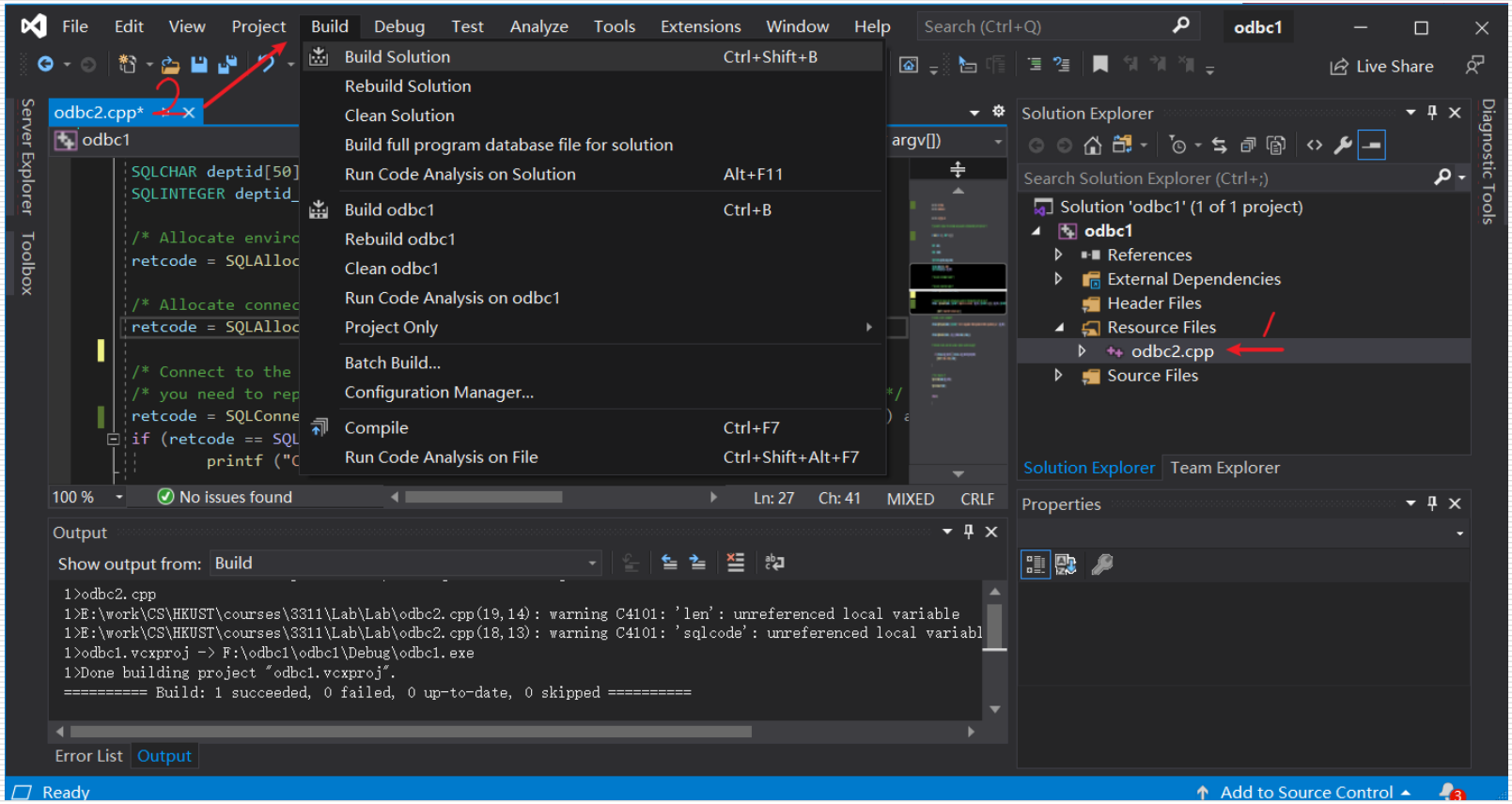
Messages | Logging Page | Statements

Line 17 Column 8 | Insert | Modified | Windows: CR

Running the example 2



Running the example 2



Running the example 2

```
F:\odbc1\odbc1\Debug>odbc1.exe comp3311ta1 [redacted]
Connected to Oracle.
room = 3528

F:\odbc1\odbc1\Debug>
```

Executable_file User_name Password

Connection to Oracle through ODBC is successful!

Prepared statement ran successfully, retrieved value is bound to a C++ local variable "room" and is displayed to the screen

Getting error information 1

- To obtain ODBC error information, one can use the SQLGetDiagRec() function.
- The function will return the
  - SQLSTATE,
  - the native error code
  - the diagnostic message

  for the error.

Getting error information 2

- The syntax of the SQLGetDiagRec() function:

```
SQLGetDiagRec(    SQLSMALLINT HandleType,
SQLHANDLE Handle,      SQLSMALLINT RecNumber,
SQLCHAR * SQLState,      SQLINTEGER * NativeErrorPtr,
SQLCHAR * MessageText,      SQLSMALLINT BufferLength,
SQLSMALLINT * TextLengthPtr);
```

- The possible return codes are SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, and SQL_INVALID_HANDLE.

Getting error information 3

- HandleType is a handle identifier and the value can be SQL_HANDLE_ENV, SQL_HANDLE_DBC, SQL_HANDLE_STMT or SQL_HANDLE_DESC.
- Handle is the input handle for getting the specific error. It must be of the same type as declared by HandleType.
- When there are multiple errors, RecNumber allows the programmer to indicate which error to be retrieved. The first error message starts at RecNumber=1.
- SQLState is a pointer that points to the buffer that the five-character SQLSTATE code will be stored.
- NativeErrorPtr is a pointer that points to the buffer where the native error code will be stored. The native error code is specific to the particular data source (DMBS).
- MessageText is a pointer to the buffer where the diagnostic message (a character string) will be stored.
- BufferLength is the length of the MessageText buffer in characters.
- TextLengthPtr is a pointer to the buffer where the size of the MessageText string (in number of characters) is stored.

Getting error information 4

- A piece of code that calls the SQLGetDiagRec() function to retrieve the error message:

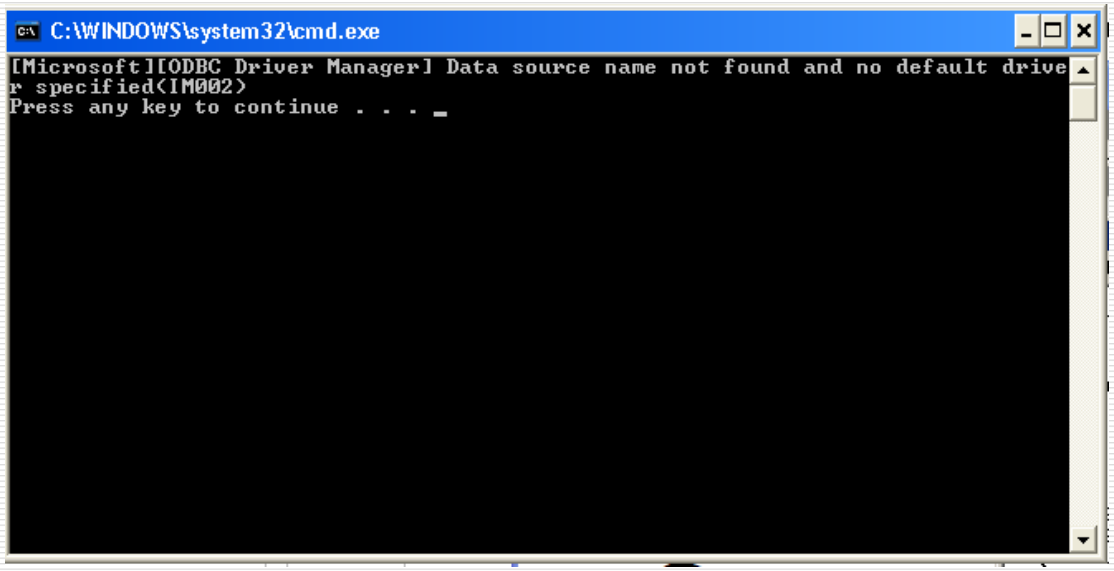
```
retcode = SQLConnectA(hdbc, (SQLCHAR*) "comp1234.cse.ust.hk", SQL_NTS,
(SQLCHAR*) argv[1], SQL_NTS, (SQLCHAR*) argv[2], SQL_NTS);

if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO){
    printf ("Connected to Oracle.\n");}
else {
    SQLGetDiagRecA(SQL_HANDLE_DBC,hdbc,1,sqlstate, &sqlcode,
  msg,4000,&len);
    printf("%s(%s)\n",msg,sqlstate);
    exit;}
```

- In the above code, the data source name is incorrect. So we should expect the error message to complain about that.

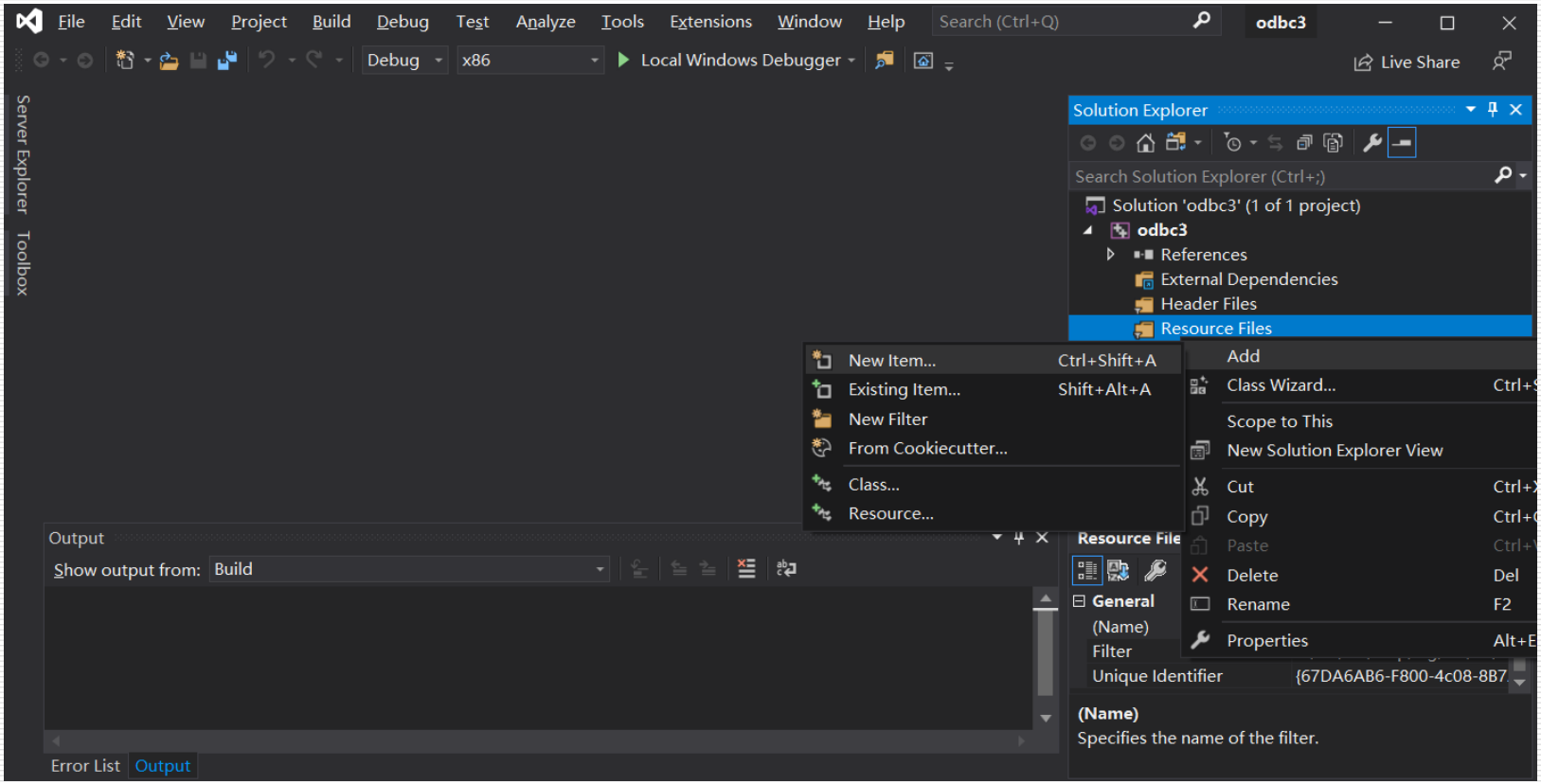
Getting error information 5

- The following is the error message returned:



- The complete code is available at:
  http://course.cse.ust.hk/comp3311/labs/odbc3.cpp

Getting error information 5



Getting error information 5

```
C:\Users\Murphy>cd C:\Users\Murphy\source\repos\Project3\Debug

C:\Users\Murphy\source\repos\Project3\Debug>Project3.exe comp3311ta10 123456
[Microsoft][ODBC Driver Manager] Data source name not found and no default driver specified(IM002)

C:\Users\Murphy\source\repos\Project3\Debug>
```

Getting error information 6

- Some common error codes:

SQLSTATE	Error
01000	General warning
08002	Connection name in use
08003	Connection not open
08007	Connection failure during transaction
22012	Division by zero
28000	Invalid authorization specification

- The complete list of error codes is available at: http://msdn.microsoft.com/en-us/library/ms714687(v=VS.85).aspx

Conclusion

- We covered the following topics in this lab:
  - the different data-types of ODBC,
  - the prepared statement,
  - function for getting ODBC error messages.

Appendix 1: List of all ODBC functions and datatypes

- The following page contains detailed information about all the ODBC functions:

http://msdn.microsoft.com/en-us/library/ms712628(v=VS.85).aspx

- More C datatypes are available at:

http://msdn.microsoft.com/en-us/library/ms714556(v=VS.85).aspx

- More SQL datatypes are available at:

http://msdn.microsoft.com/en-us/library/ms710150(v=VS.85).aspx

About char[] usage in C++

- What does the %d, %s means?

http://www.cplusplus.com/reference/cstdio/printf/

- A simple way to do char[] string operation in C++.

http://www.cplusplus.com/reference/cstdio/sprintf/

Using sprintf

- sprintf can be used to prepare statements for SQLDirectExecA

```
char dept_id[100]="MATH";
char query[100];
sprintf(query,"SELECT * from departments where department_id=\'%s\'",dept_id);
```

- The result of this statement is that the variable query will contain "SELECT * from departments where department_id='MATH'"
- Now, these two statements will have the same effect:

```
SQLExecDirectA(hstmt, (SQLCHAR *) "select * from departments where department_id=\'MATH\'",SQL_NTS);

SQLExecDirectA(hstmt, (SQLCHAR *) query, SQL_NTS);
```

Note than in strings in C++, \' is need to get ' in the result