```
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <string.h>
#include <sqlext.h>

/*
** Define Some useful defines
*/
#if !defined (NULL)
#define NULL  0
#endif


/*
** function: ODBC_error
**
** Purpose: Display to stdout current ODBC Errors
**
** Arguments: henv    _ ODBC Environment handle.
**    hdbc    - ODBC Connection Handle error generated on.
**    hstmt - ODBC SQL Handle error generated on.
**
** Returns: void
**
*/

void ODBC_error (   /* Get and print ODBC error messages */
  SQLHANDLE henv,    /* ODBC Environment */
  SQLHANDLE hdbc,    /* ODBC Connection Handle */
  SQLHANDLE hstmt)    /* ODBC SQL Handle */
{
  UCHAR sqlstate[10];
  UCHAR errmsg[SQL_MAX_MESSAGE_LENGTH];
  SQLINTEGER  nativeerr;
  SQLSMALLINT actualmsglen;
  RETCODE rc;

loop:   rc = SQLError(henv, hdbc, hstmt,
    (SQLCHAR*)sqlstate, &nativeerr, (SQLCHAR*)errmsg,
    SQL_MAX_MESSAGE_LENGTH - 1, &actualmsglen);

  if (rc == SQL_ERROR) {
    printf ("SQLError failed!\n");
    return;
  }

  if (rc != SQL_NO_DATA_FOUND) {
    printf ("SQLSTATE = %s\n",sqlstate);
    printf ("NATIVE ERROR = %d\n",nativeerr);
    errmsg[actualmsglen] = '\0';
    printf ("MSG = %s\n\n",errmsg);
    goto loop;
  }

}

/*
** function:  EnvClose
**
** Purpose: Frees environment and connection handles.
```

```
**
** Arguments: henv    _ environment handle
**    hdbc    - connection to handle
*/
void EnvClose(SQLHANDLE henv, SQLHANDLE hdbc)
{
  SQLDisconnect (hdbc);
  SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
  SQLFreeHandle (SQL_HANDLE_ENV, henv);

}

/*
** function:  fgets_wrapper
**
** Purpose: Handles newline for fgets from stdin.
**
** Arguments: buffer    _ character array ptr
**    buflen    - max characters
*/
char *fgets_wrapper(char *buffer, size_t buflen)
{
    if (fgets(buffer, buflen, stdin) != 0)
    {
        buffer[strcspn(buffer, "\n")] = '\0';
        return buffer;
    }
    return 0;
}

/*
** Defines used by main program.
*/
#define PWD_LEN   32
#define UID_LEN   32
#define DSN_LEN   32
#define USAGE_MSG1 "Usage: %s \n"

/*
** Program: odbc demo
**
** Purpose: ODBC Demo routine.
*/

typedef struct
{
  UCHAR charCol1 [1024];
//  SQLLEN length1 ;
  long length1 ;
  short length2 ;
}DataInfoStruct ;

DataInfoStruct dataStruct[32] ;
DataInfoStruct dataStruct2[32] ;
DataInfoStruct dataStruct3[32] ;

int main(int argc, char * argv[])
{
  SQLHANDLE  hdbc;
  SQLHANDLE  henv;
  SQLHANDLE hstmt;
```

```
RETCODE rc;
UCHAR uid[UID_LEN];
UCHAR pwd[PWD_LEN];
UCHAR driver[DSN_LEN];
UCHAR ver[32];
SQLSMALLINT strLen;
int i;

uid[0] = 0;
pwd[0] = 0;
if (argc > 1) {
  printf(USAGE_MSG1, argv[0]);
  return(1);
}

printf ("\nEnter the DSN : ") ;
fgets_wrapper((char*)driver, DSN_LEN);
printf ("\nEnter the UID : ") ;
fgets_wrapper((char*)uid, UID_LEN);
printf ("\nEnter the PWD : ") ;
fgets_wrapper((char*)pwd, PWD_LEN);

printf ("%s: will connect to data source '%s' as user '%s/%s'.\n",
  argv[0], driver, uid, pwd);

rc = SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO))
{
  printf("Unable to allocate environment\n");
  exit(255);
}

rc = SQLSetEnvAttr( henv,
                        SQL_ATTR_ODBC_VERSION,
                        (SQLPOINTER)SQL_OV_ODBC3,
                        SQL_IS_INTEGER );

rc = SQLConnect (hdbc, (SQLCHAR*)uid, SQL_NTS, (SQLCHAR*)pwd, SQL_NTS);
if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO))
{
  printf("SQLConnect: Failed...\n");
  ODBC_error (henv, hdbc, SQL_NULL_HSTMT);
  exit(255);  /* Exit with failure */
}

rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt);
if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO)) {
  printf ("Unable to Allocate a SQLHANDLE:\n");
  ODBC_error (henv, hdbc, hstmt);
  EnvClose (henv, hdbc);
  exit (255);
}

rc = SQLGetInfo(hdbc, SQL_DRIVER_VER, (SQLPOINTER)ver, sizeof(ver), &strLen);
if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO)) {
  printf ("SQLGetInfo has Failed. RC=%d\n", rc);
  ODBC_error (henv, hdbc, hstmt);
}

printf("Driver version: %s\n", ver);
```

```
  printf("Calling SQLGetTypeInfo...\n");
  rc = SQLGetTypeInfo ((SQLHSTMT)hstmt, SQL_ALL_TYPES) ;
  if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO)) {
    printf ("SQLGetTypeInfo has Failed. RC=%d\n", rc);
    ODBC_error (henv, hdbc, hstmt);
    EnvClose (henv, hdbc);
    exit (255);
  }

  rc = SQLBindCol (hstmt, 1, SQL_C_CHAR,
        &dataStruct[0].charCol1[0],
        (SDWORD)sizeof(dataStruct[0].charCol1),
        (SQLLEN*)&dataStruct[0].length1);
  if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO)) {
    printf ("SQLBindCol(1) has Failed. RC=%d\n", rc);
    ODBC_error (henv, hdbc, hstmt);
    EnvClose (henv, hdbc);
    exit (255);
  }

  rc = SQLBindCol (hstmt, 4, SQL_C_CHAR,
        &dataStruct2[0].charCol1[0],
        (SDWORD)sizeof(dataStruct2[0].charCol1),
        (SQLLEN*)&dataStruct2[0].length1);
  if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO)) {
    printf ("SQLBindCol(4) has Failed. RC=%d\n", rc);
    ODBC_error (henv, hdbc, hstmt);
    EnvClose (henv, hdbc);
    exit (255);
  }

  rc = SQLBindCol (hstmt, 5, SQL_C_CHAR,
        &dataStruct3[0].charCol1[0],
        (SDWORD)sizeof(dataStruct3[0].charCol1),
        (SQLLEN*)&dataStruct3[0].length1);
  if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO)) {
    printf ("SQLBindCol(5) has Failed. RC=%d\n", rc);
    ODBC_error (henv, hdbc, hstmt);
    EnvClose (henv, hdbc);
    exit (255);
  }

// Set to bogus
dataStruct[0].length1 = -1;
dataStruct2[0].length1 = -1;
dataStruct3[0].length1 = -1;

int count = 0;

  printf("Fetching result set...\n");
  while (SQLFetch (hstmt) == SQL_SUCCESS)
  {
    char* ptr;

    if (count == 0)
    {
      printf("LITERAL PREFIX\n");
      printf("indicator(d):    %d\n", dataStruct2[0].length1);
      printf("indicator(lld):  %lld\n", dataStruct2[0].length1);
      printf("indicator(hex):  ");
```

```
      /* assign to char so we can dump out each byte */
      ptr = (char*)&dataStruct2[0].length1;

      for (i=0; i < 8; i++)
      {
        printf("%x", ptr[i]);
      }
      printf("\n");
      printf("%-32s\n\n", dataStruct2[0].length1 == SQL_NULL_DATA ? (UCHAR*)"NULL"
: dataStruct2[0].charCol1);

      printf("LITERAL SUFFIX\n");
      printf("indicator(d):    %d\n", dataStruct3[0].length1);
      printf("indicator(lld):  %lld\n", dataStruct3[0].length1);
      printf("indicator(hex):  ");

      /* assign to char so we can dump out each byte */
      ptr = (char*)&dataStruct3[0].length1;

      for (i=0; i < 8; i++)
      {
        printf("%x", ptr[i]);
      }
      printf("\n");
      printf("%-32s\n\n", dataStruct3[0].length1 == SQL_NULL_DATA ? (UCHAR*)"NULL"
: dataStruct3[0].charCol1);
    }

    printf("indicator(d):    %d\n", dataStruct[0].length1);
    printf("indicator(lld):  %lld\n", dataStruct[0].length1);
    printf("indicator(hex):  ");

    /* assign to char so we can dump out each byte */
    ptr = (char*)&dataStruct[0].length1;

    for (i=0; i < 8; i++)
    {
      printf("%x", ptr[i]);
    }
    printf("\n");
    printf("%-32s\n\n", dataStruct[0].length1 == SQL_NULL_DATA ? (UCHAR*)"NULL" :
    dataStruct[0].charCol1);


    dataStruct[0].length1 = -1;
    dataStruct2[0].length1 = -1;
    dataStruct3[0].length1 = -1;

    count++;

  }
  if (rc == SQL_NO_DATA_FOUND) {
    printf ("SQLFetch returns: SQL_NO_DATA_FOUND\n");
    goto end;
  }
  if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO)) {
    printf ("SQLFetch has Failed. RC=%d\n", rc);
    ODBC_error (henv, hdbc, hstmt);
   goto end;
  }
```

```
/*
** Free Bind Buffers
*/
end:
  rc = SQLFreeStmt (hstmt, SQL_UNBIND);
  EnvClose(henv, hdbc);
}
```