C++课程设计任务指导书

(2022/2023 学年第一学期)

指导教师：陈巧红、程宏伟

任务说明

本次课程设计每人独立完成任务 1 或任务 2 中的一项任务即可，自由选择。

课程设计综合评分由“设计报告 30%+程序演示 50%+过程性考核 20%”3 部分组成。各部分评分标准如下：

（1）设计报告考核目标与评价标准

课程目标	评分标准	
目标 1：根据任务需求，能够运用面向对象的程序设计思想给出合理的设计方案	内容完整，设计合理，制图规范，格式符合要求，有详细的分析和总结	优（90-100）
	内容较完整，设计较合理，制图规范，格式符合要求，有分析和总结	良（75-89）
	内容基本完整，设计基本合理，制图基本规范，格式基本符合要求，无分析和总结	中/及格（60-74）
	内容不完整，设计不合理，制图不规范，格式不符合要求，无分析和总结	不及格（0-59）

（2）程序演示考核目标与评价标准

课程目标	评分标准	
目标 2：能够搭建出符合设计需求的用户界面开发环境和数据存储环境，并用 C++语言实现	采用图形用户界面接收及展示数据，使用数据库存储数据，并能正确使用 C++语言设计编写符合设计需求的程序，且编码规范，功能完善	优（90-100）
	使用数据库存储数据，并能正确使用 C++语言设计编写符合设计需求的程序，且编码较规范，功能较完善	良（80-89）
	使用数据库存储数据，并能正确使用 C++语言设计编写符合设计需求的程序，且编码基本规范，功能基本完善	中（70-79）
	能正确使用 C++语言设计编写符合设计需求的程序，且编码规范性尚可，能完成基本功能	及格（60-69）
	不能根据设计需求编写符合 C++语法规则的程序，或编码不规范，功能不完善	不及格（0-59）

（3）过程性考核目标与评价标准

课程目标	评分标准	
目标 3：能合理规划从设计、实现、改进到完善的整个开发进度	能够合理规划从设计、实现、改进到完善的整个开发进度，很好地按照计划准时完成	优（90-100）
	能够较合理地规划从设计、实现、改进到完善的整个开发进度，较好地按照计划完成	良（75-89）
	能够基本合理地规划从设计、实现、改进到完善的整个开发进度，基本能够按照计划完成	中/及格（60-74）

	规划了从设计、实现、改进到完善的整个开发进度，但不能按照计划准时完成	不及格（0-59）

其中功能的完成度评价主要根据任务书中给出的阶梯式任务点的完成情况来评定。需要特别注意的是，除了第 1 个任务点，每个任务点的任务都包含了它前一个任务点的任务。例如第 3 个任务点为 80 分以上的任务，则必须同时完成第 1、第 2 和第 3 个任务点的任务，才能拿到功能完成度上 80 分以上的成绩。

任务 1

【题目】快件管理系统

【目的】

通过设计一个小型的快件管理系统，训练综合运用所学知识处理实际问题的能力，强化面向对象的程序设计理念，使自己的程序设计与调试水平有一个明显的提高。

【要求】

1、每个学生必须独立完成；

2、课程设计时间为 1 周分散进行；

3、设计语言采用 C++,程序设计方法必须采用面向对象的程序设计方法；

4、课程设计期间，无故缺席按旷课处理；缺席时间达四分之一以上者，未按规定上交实验报告的学生，其成绩按不及格处理。

【内容简介】

有一个快递服务代收点，现在需要你为这个服务代收点开发一个简单的快件管理系统，使收件人能够查询自己的快件情况，服务人员能够使用该系统管理该点代收的所有快件，并通知收件人取件，加快工作效率，提高服务质量。

【考核标准】

该系统为两种角色的用户提供服务，一种是代收点服务人员，一种是收件人。代收点服务人员根据账号、密码登录系统。收件人通过手机号和密码登录系统。

1、 收件人需注册才能登录该系统。代收点服务人员可将收件信息录入系统，收件信息包括快递单号、快递公司、收件人、收件人联系电话、收件人地址、邮编、寄件人、寄件人联系电话、寄件人地址、邮编、物品重量，系统可自动为该快件生成唯一的取件号。如收件人来取件，服务人员可根据手机号或者取件号查询到该快件并标记取件成功。收件人可以通过手机号查询自己在该代收点的快件的取件号以及是否收取的情况，成绩≥60；

2、 收件人注册信息和快件信息均保存在数据库中，其中，连接数据库所需信息（数据库服务 器地址、用户名、密码、数据库名）以文件形式存放，程序通过从文件中读取这些信息获得与数据库的连接。当快件信息录入有误时，代收点服务人员可以根据快递单号查找、删除和修改某个未取快件，还可以查询所有未取快件。成绩≥70；

3、 系统可根据数据库中存储的历史记录对收取件情况进行统计，根据服务人员的输入日期统计某天的收取件情况并显示，包括当天的收件量和取件量，各快递公司的收件量、取件量、未取件数量。成绩≥80；

4、 取件号可以根据快件的重量、快递公司等信息实现自动编码。界面交互性好，功能设计全面、合理且有亮点。成绩≥90；

附加要求：

数据库管理系统自由选择，推荐使用 MySQL 数据库。对图形化 UI 接口有兴 趣的同学推荐选用 Qt 进行开发，不做强制性要求。

参考资料如下：

[1] 官网：https://www.qt.io/zh-cn/develop

[2] 教程：http://c.biancheng.net/qt/

[3] 教学视频：https://www.bilibili.com/video/BV1Jp4y167R9/?spm_id_from=333. 337.search-card.all.click

用面向对象的程序设计方法设计该系统。本系统涉及的基本对象包括代收员对象、收件人对象、快件对象、快件管理对象、系统界面对象等。实现对这些对象的合理抽象和封装，正确定义类之间的关系。界面合理，代码文件组织清晰，命名符合规范，代码注释清楚，课设报告书质量高。

任务 2

【题目】便利店管理系统

【目的】

通过设计一个小型的便利店管理系统，训练综合运用所学知识处理实际问题的能力，强化面向对象的程序设计理念，使自己的程序设计与调试水平有一个明显的提高。

【要求】

1、每个学生必须独立完成；

2、课程设计时间为 1 周分散进行；

3、设计语言采用 C++,程序设计方法必须采用面向对象的程序设计方法；

4、课程设计期间，无故缺席按旷课处理；缺席时间达三分之一以上者，未按规定上交实验报告的学生，其成绩按不及格处理。

【内容简介】

有一家社区便利店，现在需要你为这个便利店开发一个简单的便利店管理系统，方便顾客自己购买商品，并提供对便利店销售情况的统计和管理功能。

【考核标准】

该系统为两种角色的用户提供服务，一种是便利店管理员，一种是顾客。便利店管理员根据账号、密码登录系统，顾客通过手机号和密码登录系统。

1、顾客需注册才能登录该系统。顾客通过该系统购买商品，可实现多次购买一次性结账，商品信息包括商品编号、商品名称、商品价格等，商品分为普通商品和进口商品。购买后，系统能够打印订单信息，订单内容包括订单编号、商品名称、商品的价格、商品数量、折扣、总金额等。显示进口商品名称时，在名称后加<I>；管理员可以查看当天以及当月的所有订单，成绩≥60；

2、商品信息保存在数据库中，其中，连接数据库所需信息（数据库服务 器地址、用户名、密码、数据库名）以文件形式存放，程序通过从文件中读取这些信息获得与数据库的连接。系统能够实现管理员对商品信息的管理，包括对商品信息的增加、修改、删除、查找，成绩≥70；

3、所有顾客的订单均保存在数据库中。顾客可以查询自己购买的所有订单。能够根据商品名称查询所有包含该商品的订单，能够删除、取消某个订单，订单按照下单时间先后排序，成绩≥80；

4、系统可根据数据库中存储的历史记录对销售情况进行统计，根据管理员的输入日期统计某天的销售情况并显示（包括一共销售多少单，销售额是多少，各种商品的销售情况等）。界面交互性好，功能设计全面、合理且有亮点。成绩≥90.

附加要求：

数据库管理系统自由选择，推荐使用 MySQL 数据库。对图形化 UI 接口有兴 趣的同学推荐选用 Qt 进行开发，不做强制性要求。

参考资料如下：

[4] 官网：https://www.qt.io/zh-cn/develop

[5] 教程：http://c.biancheng.net/qt/

[6] 教学视频：https://www.bilibili.com/video/BV1Jp4y167R9/?spm_id_from=333. 337.search-card.all.click

用面向对象的程序设计方法设计该系统。本系统涉及的基本对象有订单对象、订单管理对象、商品对象、进口商品对象、购物篮对象、商店对象。实现对这些

对象的合理抽象和封装，正确定义类之间的关系。界面合理，代码文件组织清晰，命名符合规范，代码注释清楚，课设报告书质量高。

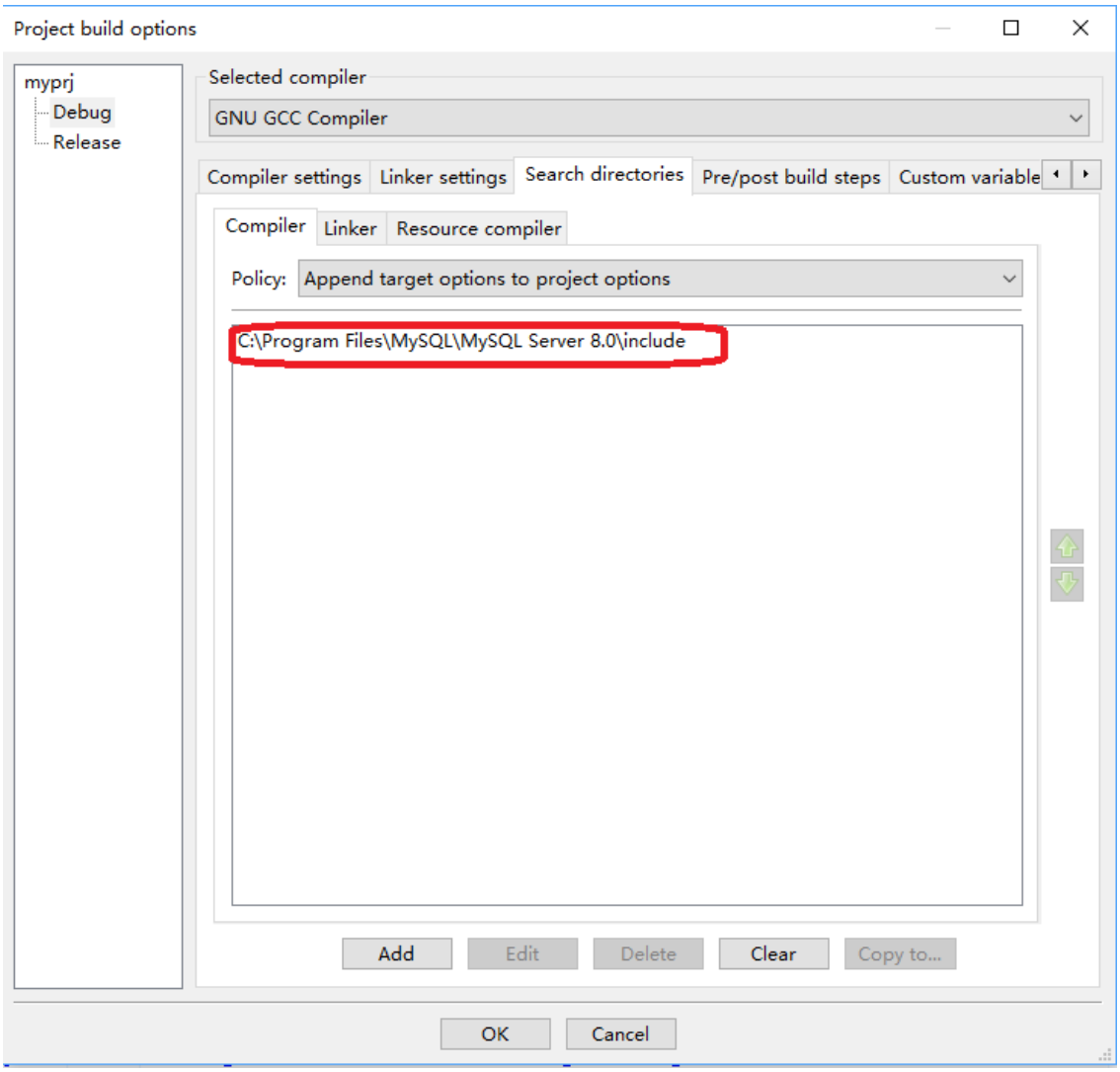
如何编写C++程序实现对MySQL数据库的访问

注意：此文档适用数据库版本为MySQL 8.0

1. 使用Codeblocks连接MySQL

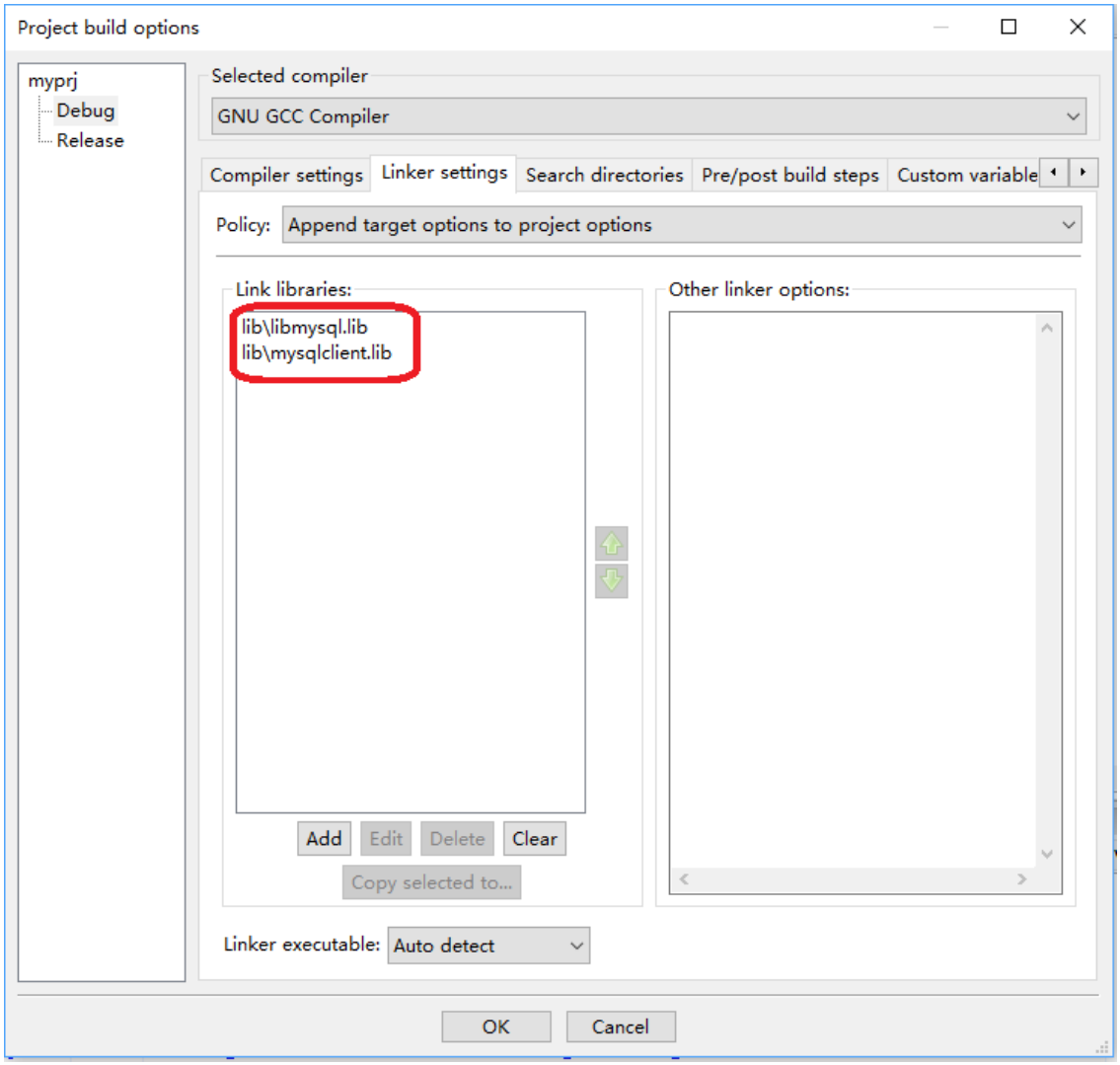
- **引入mysql.h头文件**

  选择Project->Build Options菜单项，在Search directories属性页中设置MySQL的C++接口头文件mysql.h的路径，**mysql.h的默认路径在MySQL的安装路径下的include文件夹下。**



- **为项目添加MySQL C链接库**

  选择Project->Build Options菜单项，在Linker settings属性页中设置MySQL的C++链接库libmysql.lib和mysqlclient.lib，**libmysql.lib和mysqlclient.lib在MySQL的安装路径下的lib文件夹下**，可以拷贝出来放到项目目录项然后再设置。



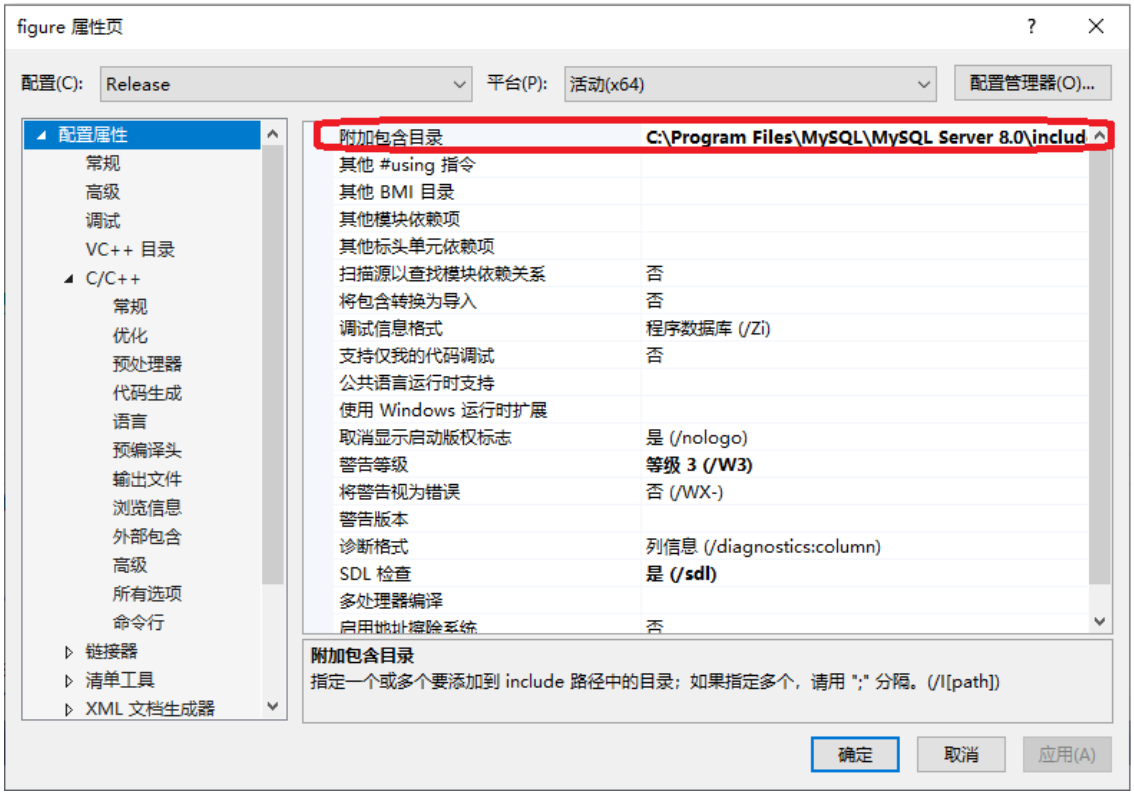
- **导入动态链接库**

  将MySQL的动态链接库libmysql.dll放到项目可执行程序的运行目录下，**libmysql.dll在MySQL的安装路径下的lib文件夹下**

2. 使用Visual Studio连接MySQL

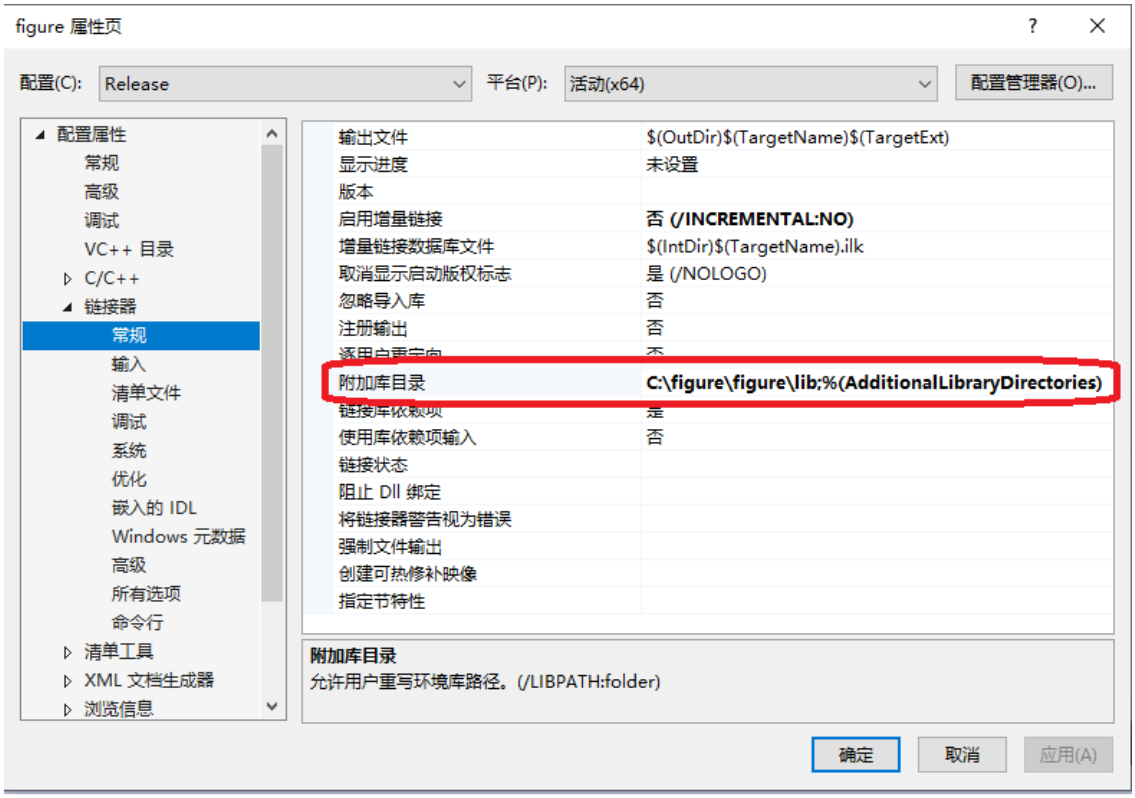
- **引入mysql.h头文件**

  选择项目—> 属性页—> 配置属性 —> C/C++ —> 常规 —>附加包含目录，设置MySQL的C++接口头文件mysql.h的路径，mysql.h的默认路径在MySQL的安装路径下的include文件夹下。

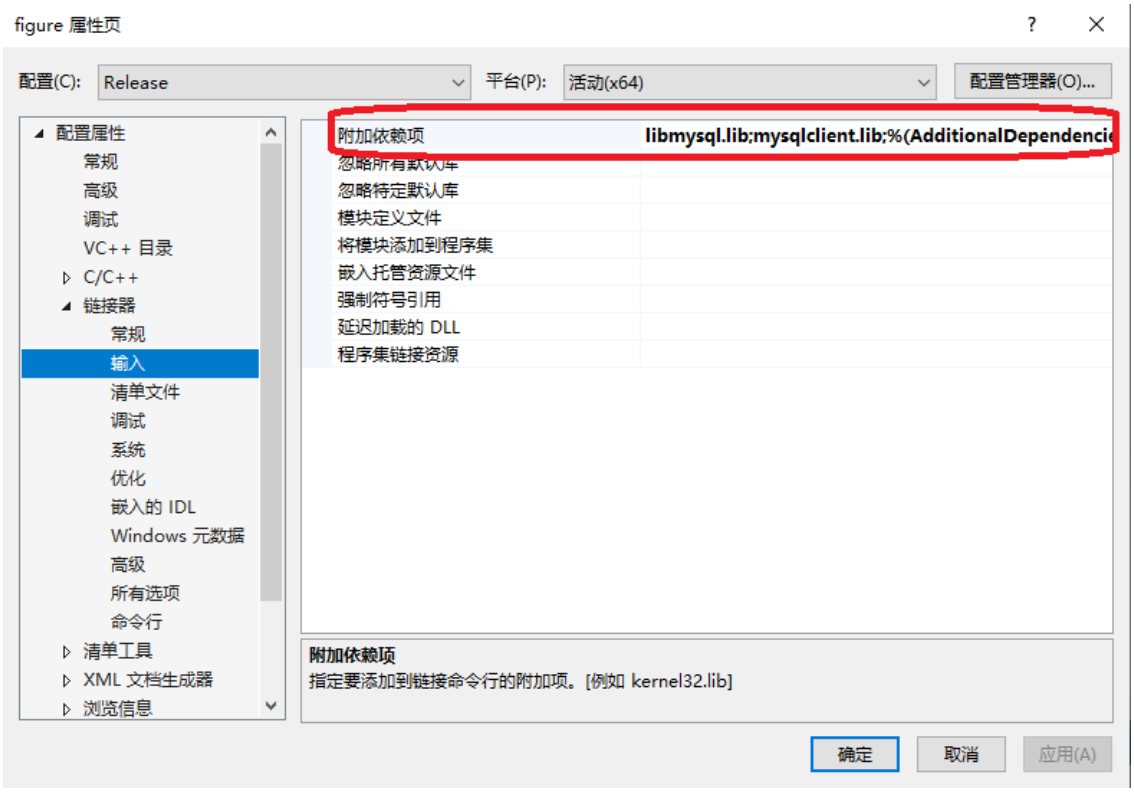


- **为项目添加MySQL C链接库**

选择项目—> 属性页—> 配置属性 —> 链接器 —> 常规 —> 附加库目录，设置MySQL的C++链接库libmysql.lib和mysqlclient所在目录，libmysql.lib和mysqlclient在MySQL的安装路径下的lib文件夹下，可以拷贝出来放到项目目录项然后在设置。



选择项目—> 属性页—> 配置属性 —> 链接器 —> 输入 —> 附加依赖项，设置MySQL的C++链接库libmysql.lib和mysqlclient。



- **导入动态链接库**

  将MySQL的动态链接库libmysql.dll放到项目可执行程序的运行目录下，libmysql.dll在MySQL的安装路径下的lib文件夹下。

  注意：如果在上述设置后出现找不到libssl-1_1-x64.dll”的错误信息，请参考博文https://blog.csdn.net/tianya_lu/article/details/115674442

3. MySQL的C程序接口

- **官方技术文档**

  https://dev.mysql.com/doc/c-api/8.0/en/c-api-function-reference.html

  基础接口文档

  https://dev.mysql.com/doc/c-api/8.0/en/c-api-basic-interface.html

- **基本程序框架**

  1. 通过调用mysql_init（）初始化连接处理程序，并通过调用连接建立函数如mysql_real_connect（）连接到服务器；
  2. 调用mysql_query（）发送SQL语句，调用mysql_fetch_row（）获得结果集后处理其结果；
  3. 通过调用mysql_close（）关闭与MySQL服务器的连接；

- **基本函数用法**

  - 初始化适用于MySQL的数据库连接对象

```
MYSQL *mysql_init(MYSQL *mysql)
```

如果参数mysql是空指针，则函数分配、初始化并返回一个新对象；否则，将初始化对象并返回对象的地址。如果内存不足，无法分配新对象，则返回NULL。

MYSQL结构体用于表示一个数据库连接的处理程序，不要试图复制MYSQL结构。

用法：

```
MYSQL my_connection;
mysql_init(&my_connection);
```

- 建立MySQL连接

```
MYSQL *
mysql_real_connect(MYSQL *mysql,
                   const char *host,
                   const char *user,
                   const char *passwd,
                   const char *db,
                   unsigned int port,
                   const char *unix_socket,
                   unsigned long client_flag)
```

host：数据库服务器地址；user：数据库连接用户名； passwd：数据库连接密码；db：数据库名；port：端口号；unix_socket：指定要使用的套接字或命名管道；client_flag：通常为0，但也可以通过设置启用某些功能，具体见技术文档。

如果连接失败，则返回NULL，否则连接成功，返回数据库连接。

用法：

```
const char* host = "localhost";
const char* username = "root";
const char* password = "123456";
const char* database = "ecommerce";
mysql_real_connect(&connection, host, username, password, database, 3306,
nullptr, 0);
```

- 执行SQL命令

```
int mysql_query(MYSQL *mysql,
                const char *stmt_str)
```

执行以'\0'结尾的字符串stmt_str，通常情况下，stmt_str由单个SQL语句组成。

如果查询成功，mysql_query返回0，否则查询失败，返回错误代码如下：

CR_COMMANDS_OUT_OF_SYNC：命令执行顺序不正确

CR_SERVER_GONE_ERROR：MySQL服务器失效

CR_SERVER_LOST：与服务器的连接在查询过程中丢失

CR_UNKNOWN_ERROR：未知错误

用法：

```
char* sql = "select * from t_user";
mysql_query(&my_connection, sql);
```

- 获取查询结果

```
MYSQL_RES *mysql_store_result(MYSQL *msql)
```

函数返回指向包含结果的MYSQL_RES的指针。如果语句未返回结果集或发生错误，则为NULL。要确定是否发生错误，请检查mysql_error( )是否返回非空字符串，mysql_errno( )是否返回非零，或者mysql_field_count( )是否返回零。

MYSQL_RES定义如下：

用法：

```
MYSQL_RES res_ptr = mysql_store_result(&my_connection);
```

- 取结果列数

```
unsigned int mysql_num_fields(MYSQL_RES *result)
```

函数返回结果集中的列数。

用法：

```
int column = mysql_num_fields(res_ptr);
```

- 取结果行数

```
uint64_t mysql_num_rows(MYSQL_RES *result)
```

函数返回结果集中的行数

用法：

```
int row = mysql_num_rows(res_ptr);
```

- 按行返回查询信息

```
MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)
```

函数返回结果集的下一行。MYSQL_ROW是一个字符串数组，每一个元素是对应的属性值，下标从0开始，可以通过遍历每一个元素读取结果集中的每一行的每一列的属性值。

MYSQL_ROW定义如下：

```
typedef char **MYSQL_ROW;
```

用法：

```
MYSQL_ROW row;
unsigned int num_fields = mysql_num_fields(res_ptr);
while ((row = mysql_fetch_row(res_ptr)))
{
   for(unsigned int i = 0; i < num_fields; i++)
   {
       printf("[%s] ", row[i] ? row[i] : "NULL");
   }
   printf("\n");
}
```

- 获取影响结果的行数

```
uint64_t mysql_affected_rows(MYSQL *mysql)
```

如果是UPDATE、DELETE或INSERT语句，则返回上一条语句更改、删除或插入的行数，对于SELECT语句，mysql_infected_rows（）的工作方式与mysql_ num_rows（）类似。

用法：

```
uint64_t num_rows = mysql_affected_rows(&connection);
```

- 获取表字段名

```
MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *result)
```

函数返回包含表字段信息的MYSQL_FIELD类型的数组，字段信息包括字段的名称（name）、类型和大小。可通过不断

MYSQL_FIELD定义如下：

```
typedef struct MYSQL_FIELD {
  char *name;                 /* Name of column */
  char *org_name;             /* Original column name, if an alias */
  char *table;                /* Table of column if column was a field */
  char *org_table;            /* Org table name, if table was an alias */
  char *db;                   /* Database for table */
  char *catalog;              /* Catalog for table */
  char *def;                  /* Default value (set by mysql_list_fields) */
  unsigned long length;       /* Width of column (create length) */
  unsigned long max_length; /* Max width for selected set */
  unsigned int name_length;
  unsigned int org_name_length;
  unsigned int table_length;
  unsigned int org_table_length;
  unsigned int db_length;
  unsigned int catalog_length;
  unsigned int def_length;
  unsigned int flags;         /* Div flags */
  unsigned int decimals;      /* Number of decimals in field */
  unsigned int charsetnr;     /* Character set */
  enum enum_field_types type; /* Type of field. See mysql_com.h for types */
  void *extension;
} MYSQL_FIELD;
```

用法：

```
MYSQL_FIELD *fields;
unsigned int num_fields = mysql_num_fields(res_ptr);
fields = mysql_fetch_fields(res_ptr);
for(unsigned int i = 0; i < num_fields; i++)
{
   printf("Field %u is %s\n", i, fields[i].name);
}
```

- 获取MySQL API函数调用错误信息

```
const char *mysql_error(MYSQL *mysql)
```

对于mysql指定的连接，mysql_error（）返回一个以null结尾的字符串，其中包含最近调用的失败API函数的错误消息。如果函数没有失败，mysql_error（）的返回值可能是前一个错误，也可能是表示没有错误的空字符串。

用法：

```
char *error = mysql_error(&connection));
```

- 关闭连接

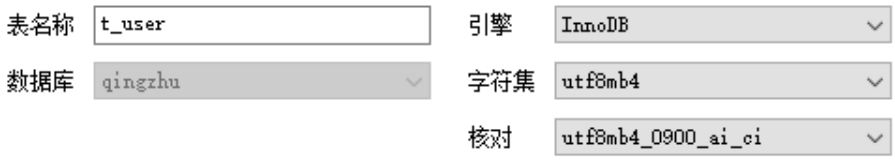
```
void mysql_close(MYSQL *mysql)
```

关闭打开的数据库连接。关闭后，mysql将不可用。

用法：

```
mysql_close(&connection);
```

- **操作实例**
  - **表结构**



  - **数据库连接类**

```
#include "mysql.h"
class DbConnection{
public:
    DbConnection(const char* host, const char* username, const char*
password, const char* database); //创建与数据库的连接
    ~DbConnection();  //关闭与数据库的连接
    //向数据库发送select命令，查询成功，返回结果集，失败，抛出异常
    MYSQL_RES executeQuery(const char* sql);
     //向数据库发送insert、updatedelete或命令，执行成功，返回插入、更新或删除的记录数，
失败，抛出异常
    int executeSQL(const char* sql);
private:
    MYSQL connection;  //MySQL的数据库连接
};
```

```
DbConnection::DbConnection(const char* host, const char* username, const
char* password, const char* database)
{
     //初始化MySQL连接对象
     mysql_init(&connection);
     //通过设定数据库连接主机地址host、端口号3306，用户名username、密码passowrd和数据
库名database等，建立与MySQL数据库的连接，连接失败抛出异常
     if(NULL == mysql_real_connect(&connection, host, username, password,
database, 3306, nullptr, 0))
         throw DbExcepiton(mysql_error(&connection));
    //设置查询编码为gbk，以支持中文
    mysql_query(&connection, "set names gbk");
}
DbConnection::~DbConnection(){
    mysql_close(&connection);
}
MYSQL_RES  DbConnection::executeQuery(const char* sql)
{
    //执行select命令
    int res = mysql_query(&connection, sql);

    if(res) //执行失败
        throw DbExcepiton(mysql_error(&connection));
    else
    {
        //获取结果集
        MYSQL_RES* res_ptr = mysql_store_result(&connection);
        //如果获取成功
        if(res_ptr)
            return *res_ptr;
        else throw DbExcepiton(mysql_error(&connection));
    }
}
 int DbConnection::executeSQL(const char* sql)
 {
    //执行insert、updatedelete或命令
    int res = mysql_query(&connection, sql);

    if(res) //执行失败
        throw DbExcepiton(mysql_error(&connection));
    else //执行成功，返回产生影响的记录数
    {
        return mysql_affected_rows(&connection);
    }
 }
```

- **数据库异常类**

```
#include<exception>
using namespace std;
class DbExcepiton: public exception
{
public:
    DbExcepiton(const char* msg):err_msg(msg){}
    const char* what() const throw(){return err_msg;}
private:
    const char* err_msg;
};
```

- **select操作**

```
#include<iostream>
#include"mysql.h"
#include"DbConnection.h"
#include"DbException.h"
using std::cout;
using std::endl;
int main()
{
     const char* host = "localhost";
     const char* username = "root";
     const char* password = "123456";
     const char* database = "encommerce";
     try{
         DbConnection db(host, username, password, database);
         //执行select命令，查询t_user表中的所有记录
         char *sql;
         sql = "select * from t_user";
         MYSQL_RES res = db.executeQuery(sql);

         //输出字段名
         unsigned int num_fields = mysql_num_fields(&res);
         MYSQL_FIELD* fields = mysql_fetch_fields(&res);
         for(unsigned int i = 0; i < num_fields; i++){
            cout.width(20); cout.setf(ios::left);
            cout << fields[i].name;
         };
         cout << endl;

         //输出查询出的结果集中的每条记录中的数据
         MYSQL_ROW row;
         while ((row = mysql_fetch_row(&res)))
         {

           for(unsigned int i = 0; i < num_fields; i++)
           {
               cout.width(20); cout.setf(ios::left);
               cout << row[i] ? row[i] : "NULL";
           }
           cout << endl;
         }
     }catch(DbExcepiton exp){
```

```
        cout << exp.what() << endl;
    }
    return 0;
}
```

- **insert操作**

```
#include<iostream>
#include"DbConnection.h"
#include"DbException.h"
using std::cout;
using std::endl;
int main()
{
    const char* host = "localhost";
    const char* username = "root";
    const char* password = "123456";
    const char* database = "encommerce";
    try{
        DbConnection db(host, username, password, database);
        string sql;
        string loginName, password;
        loginName = "\'zstu002\'";
        password = "\'111111\'";
        //构建insert语句,向t_user表中插入login_name字段的值为zstu002,password字段
值为    111111的数据
        sql = "insert into t_user (login_name, password) values (";
        sql = sql + loginName + "," + password + ")";
        //执行insert命令，返回成功插入的记录个数
        int num_add = db.executeSQL(sql.c_str());
        cout << num_add << " users is inserted!" << endl;
    }catch(DbExcepiton exp){
        cout << exp.what() << endl;
    }
    return 0;
}
```

- **delete操作**

```
#include<iostream>
#include"DbConnection.h"
#include"DbException.h"
using std::cout;
using std::endl;
int main()
{
    const char* host = "localhost";
    const char* username = "root";
    const char* password = "123456";
    const char* database = "encommerce";
    try{
        DbConnection db(host, username, password, database);
        string sql;
        int userId = 5;
```

```
        //构建delete语句，删除t_user表中user_id值为5的记录
        sql = "delete from t_user where ";
        sql = sql + "user_id = " + to_string(userId);
        //执行delete命令，返回成功删除的记录个数
        int num_add = db.executeSQL(sql.c_str());
        if(num_add != 0)
           cout << num_add << " users is deleted!" << endl;
        else cout << "The deleted user does not exist" << endl;
    }catch(DbExcepiton exp){
       cout << exp.what() << endl;
    }
    return 0;
}
```

- **update操作**

```
#include<iostream>
#include"DbConnection.h"
#include"DbException.h"
using std::cout;
using std::endl;
int main()
{
    const char* host = "localhost";
    const char* username = "root";
    const char* password = "123456";
    const char* database = "encommerce";
    try{
        DbConnection db(host, username, password, database);
        string sql;
        string password = "\'123456\'";
        int userId = 6;
        //构建update语句，将t_user表中user_id值为6的password的属性值修改为123456
        sql = "update t_user set ";
        sql = sql + "password = " + password;
        sql = sql + "where user_id = " + to_string(userId);
        //执行update命令，返回成功更新的记录个数
        int num_add = db.executeSQL(sql.c_str());
        if(num_add != 0)
           cout << num_add << " users is updated!" << endl;
        else cout << "The updated user does not exist" << endl;
    }catch(DbExcepiton exp){
       cout << exp.what() << endl;
    }
    return 0;
}
```