责任编辑　刘欣鑫　李　勇
装帧设计　墨创 MC CULTURE

数据结构
综合设计教程

数据结构综合设计教程

蔡茂蓉　杨春明◎主编

ISBN 978-7-5697-3374-7
9 787569 733747 >
定价：42.80元

SWUPG 西南大學出版社
国家一级出版社　全国百佳图书出版单位

普通高等学校一流专业建设
计算机科学与技术系列规划教材

数据结构
综合设计教程

蔡茂蓉　杨春明◎主编
孙　敏　曾立胜　李学俊◎副主编

SWUPG 西南大學出版社
国家一级出版社　全国百佳图书出版单位

普通高等学校一流专业建设
计算机科学与技术系列规划教材

数据结构综合设计教程

蔡茂蓉　杨春明◎主编
孙　敏　曾立胜　李学俊◎副主编

西南大学出版社
SWUPG
国家一级出版社　全国百佳图书出版单位

图书在版编目(CIP)数据

数据结构综合设计教程 / 蔡茂蓉 , 杨春明主编 . 重庆 : 西南大学出版社 , 2025. 10. -- ISBN 978-7-5697-3374-7

Ⅰ. TP311.12

中国国家版本馆 CIP 数据核字第 2025JP0141 号

数据结构综合设计教程

SHUJU JIEGOU ZONGHE SHEJI JIAOCHENG

蔡茂蓉　杨春明　主编

责任编辑：刘欣鑫　李　勇
责任校对：向文平
装帧设计：墨创文化
排　　版：怀恩文化
出版发行：西南大学出版社(原西南师范大学出版社)
地　址:重庆市北碚区天生路 2 号
邮　编:400715
印　　刷：崇阳文昌印务股份有限公司
成品尺寸：185 mm × 260 mm
印　　张：10
字　　数：211 千字
版　　次：2025 年 10 月　第 1 版
印　　次：2025 年 10 月　第 1 次印刷
书　　号：ISBN 978-7-5697-3374-7

定　　价：42.80 元

目 录

第 4 章　线性表、查找和排序的综合应用

线性表作为一种基本的数据结构，在多个领域有着广泛的应用。另外，在大部分涉及数据处理的程序中，查找和排序都是最常见的操作。本章通过一个案例，综合应用线性表、查找和排序等知识，实现一个简单的学生成绩管理系统。通过本章案例的学习，读者能够深入地了解线性结构、查找技术和排序算法，能够在程序设计中应用线性结构解决实际的编程问题。

4.1 案例简介

4.1.1 任务描述

设计一个有图形化用户界面的学生成绩管理系统，学生信息包括学号、姓名，数学、语文和英语三门课程的百分制成绩，基本功要求如下：

（1）插入学生信息。

（2）删除学生信息。

（3）修改学生信息。

（4）查找学生信息（按学号或姓名，重名情况也能处理）。

（5）遍历并输出所有学生信息。

（6）对学生成绩或其他属性进行排序（升序和降序）。

（7）统计学生成绩的平均分、最高分和最低分以及“优良中差”等级的人数。

（8）学生信息的文件读写。

4.1.2 实现思路

（1）在 Qt Creator 中创建一个工程，创建一个 QMainWindow（也可以是 QWidget）对象窗口，并把其作为程序的主界面，学生信息用 QTableWidget 表格控件展示。

（2）通过点击 QTableWidget 的列或者行标题单元格，选择一列或者一行数据。

双击列单元格（标题单元格）进行排序。

（3）单击一个单元格可以选择该单元格，双击一个单元格可以修改该单元格的值，实现数据的编辑功能。修改数据后，需要进行数据验证。例如：学号不能重复，成绩值不能为字母，应该在 [0,100] 区间等。

（4）在主窗体添加按钮（QPushButton），实现本案例的主要功能。点击按钮后，弹出一个 QDialog 或者 QWidget 模式窗口，进入子功能界面。包括添加、查找、排序、统计学生信息等功能，每个子功能界面有按钮实现返回主窗口功能。

（5）在学生信息管理系统中，学生数据以线性表为逻辑结构，线性表作为底层数据结构，用于存储和管理学生的各项信息。

（6）为了实现数据的长期保存，可以将学生信息保存到文本文件中。当用户需要保存数据时，系统会将线性表中的学生信息写入到指定的文件中；当用户需要读取数据时，可以从文件中读取数据并填充到线性表中，再展示到 QTableWidget 里。

4.2 知识要点

根据案例任务和实现思路，本案例实现涉及的数据结构是线性表，在线性表相关的操作中，需要对数据元素进行高效的存储、访问和处理，因此深入理解线性表及其存储结构、线性表的基本运算至关重要。在实现线性表相关功能时，需要用到线性表的常用基本运算包括线性表的建立、插入、删除、查找、排序等运算，下面先对线性表及相关知识进行简单介绍。

4.2.1 线性表知识解析

1. 线性表的定义

线性表是具有相同数据类型的 n（$n \geqslant 0$）个数据元素的有限序列，通常记为（a_1，a_2，…，a_i，a_{i+1}，…，a_n）。其中 n 是线性表的长度，当 $n = 0$ 时称为空表。a_1 是表头元素，它没有直接前驱；a_n 是表尾元素，它没有直接后继。对于非空线性表中的任意一个元素 a_i（$1 < i < n$），有且仅有一个直接前驱 a_{i-1} 和一个直接后继 a_{i+1}。

线性表是可以在任意位置进行插入和删除操作的线性结构，其抽象数据类型的定义如下：

```
ADT List
{
```

数据元素：

```
D={ai|1≤i≤n,n≥0，ai 为 ElemType 类型 } //ElemType 是自定义的类型标识符
```

元素关系：

$$R=\{<a_i,a_{i+1}>|a_i,a_{i+1} \in D,i=1,2,\cdots,n-1\}$$

2. 线性表的基本运算

（1）初始化线性表 InitList (&L)　创建一个空的线性表 L。

（2）销毁线性表 DestroyList (&L)　释放线性表 L 被占用的存储空间。

（3）求线性表的长度 ListLength (L)　返回线性表 L 中的元素个数。

（4）判断线性表是否为空 ListEmpty (L)　若线性表 L 为空，则返回真；否则返回假。

（5）取值 GetElem (L,i,&e)　从线性表 L 中取出第 i 个位置的元素，并将其值赋给 e。

（6）查找 LocateElem (L,e)　在线性表 L 中查找元素 e，若找到则返回其位置，否则返回 0。

（7）插入 ListInsert (&L,i,e)　在线性表 L 的第 i 个位置插入元素 e，插入成功返回 1，否则返回 0。

（8）删除 ListDelete (&L,i,&e)　从线性表 L 中删除第 i 个位置的元素，并将其值赋给 e，删除成功返回 1，否则返回 0。

（9）遍历线性表 TraverseList (L)　按照一定的顺序访问线性表 L 中的每个元素。

3. 线性表的存储结构

线性表常用的存储结构是顺序存储和链式存储，顺序存储的线性表称为顺序表，链式存储的线性表通常有单链表和双向链表。

（1）顺序表。

顺序表是用一组地址连续的存储单元依次存储线性表的数据元素，在 C 或 C++ 中，用申请数组的方法得到连续的存储空间。例如一个线性表 $s=(a_1,a_2,\cdots,a_i,\cdots,a_n)$，顺序表存储如图 4–1 所示，在顺序存储结构中，逻辑上相邻的数据元素物理上也相邻。

地址编号	0	1	…	i–1	i	…	n–2	n–
数据元素	a_1	a_2	…	a_i	a_{i+1}	…	a_{n-1}	a_n

图 4–1　顺序表存储示意图

（2）单链表和双向链表。

线性表的链式存储结构可用一组任意的存储单元（这组存储单元可以是连续的，也可以是不连续的）来存储线性表的数据元素。为了表示每个数据元素与其直接后继元素之间的逻辑关系，除了存储数据元素本身的信息之外，还需存储一个指示其直接后继元素的信息（直接后继元素的存储位置），这两部分信息组成数据元素的存储映像称为结点；如果一个结点中只有一个指针，这种结点构成的链表称为单链表。单链表分为带头结点的单链表和不带头结点的单链表，存储结构如图 4–2

和图 4–3 所示。有时候为了查找结点的后继以及前驱，也可以使用双向链表，双向链表的存储结构示意图如图 4–4 所示。

图 4–2　不带头结点的单链表

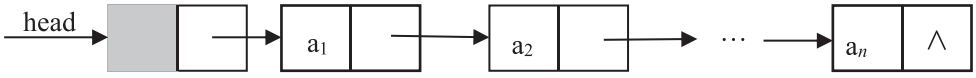

图 4–3　带头结点的单链表

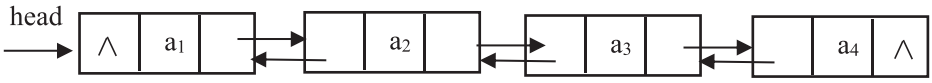

图 4–4　双向链表

4.2.2 用 list 实现学生信息表

在 Qt 中，线性表可以用 QList 容器类，QList 在 Qt 6 中实现为连续内存数组（类似 std::vector），适合随机访问。链表可以用 std::list，它的底层数据结构是一个双向链表，不适合随机访问，但是插入或删除效率高。

本案例中的数据信息在逻辑上是线性的，存储上可以选择顺序存储，也可以用链式存储。由于本案例任务中需要插入和删除学生信息，因此为了高效地插入和删除操作，把双向链表作为数据存储方式，因此选择使用 std::list 来实现本案例。表 4–1 是 list 类的常用方法。

表 4–1　std::list 类的常用方法

线性表功能函数	list 类对应的成员方法	成员方法说明
初始化线性表 InitList (&L)	list<Student> mystulist;	创建一个空的线性表
销毁线性表 DestroyList (&L)	clear()	移除列表中的所有元素，释放其占用的内存空间
求线性表的长度 ListLength (L)	size()	返回列表中元素的数量
判断线性表是否为空 ListEmpty (L)	empty()	检查列表中是否有元素，如果列表为空则返回 true，否则返回 false
插入 ListInsert (&L,i,e)	insert(iterator pos, const T& value)	在指定位置 pos 之前插入一个新元素，该元素的值为 value，返回指向新插入元素的迭代器
删除 ListDelete (&L,i,&e)	iterator erase(iterator pos);	移除 std::list 中 pos 所指向的元素

表 4–1 中的插入和删除操作，常用迭代器来实现，最初迭代器指针指向线性表第一个元素，然后后移，依次指向其他元素，直到结束。下面代码实现了 list 的基本操作，包括插入、删除、查找、访问、遍历。

```
#include <iostream>
#include <list>
using namespace std;
void printlist(list<int> mylist) // 遍历线性表
{
    for (const auto& elem : mylist) { std::cout << elem << " ";    }
}
int main()
{
    std::list<int> myList; // 创建一个空的 list
    myList.push_back(10); // 在列表末尾添加元素
    myList.push_front(5); // 在列表开头添加元素
    printlist(myList);// 输出 myList 中的数据信息
    myList.pop_front();// 删除第一个元素
    int pos,data;
    cout<<"input the pos and the insert data："<<endl;
    cin>>pos>>data;
    if(pos<1||pos>static_cast<int>(myList.size())+1)
        cout<<"insert pos error"<<endl;
    else
{
        auto it=myList.begin();
        std::advance(it,pos–1);// 迭代器移动到 pos–1 处
        myList.insert(it,data);
    }
    cout<<"input the pos to delete："<<endl;
    cin>>pos;
    if(pos<1||pos>static_cast<int>(myList.size()))
        cout<<"delete pos error"<<endl;
    else
{
        auto it=myList.begin();
        std::advance(it,pos–1);
        myList.erase(it);
        cout<<"after delete:"<<endl;
    }
    return 0;
}
```

4.3 代码实现

在 Qt Creator 中，新建一个名为 stuManager 的项目，项目类型为 Qt Widgets Application，Qt 项目结构如图 4–5 所示。

根据本案例设计要求，系统总体功能结构如图 4–6 所示，其中前面四个功能由四个窗体实现，后面三个功能在 mainwindow.cpp 文件中实现。

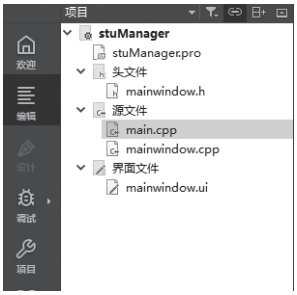

图 4–5　项目结构

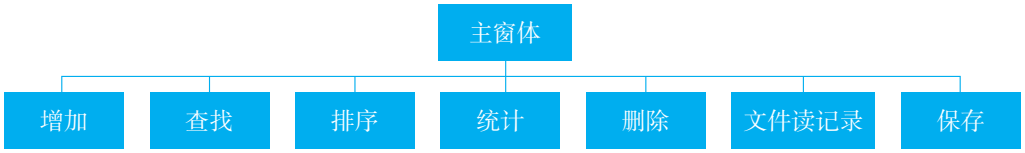

图 4–6　系统结构

创建项目文件后，再创建 4 个窗体。在项目资源管理器中，选中项目名称，点击右键，在弹出菜单中选择“添加新文件…”，弹出对话框如图 4–7 所示，左边选择“Qt”，然后在中间选择“Qt Widgets Designer Form”，单击右下角“选择…”按钮，在出现的“Qt Widgets Designer Form”的对话框中，可以选择 Dialog，也可以选择 Widget，如图 4–8 所示。后面设置好窗体文件的文件名即可。本案例创建了 addform.ui、querydialog.ui、sortdialog.ui 和 statform.ui 四个窗体文件，分别用来实现“添加”“查找”“排序”“统计”功能，如图 4–9 所示。

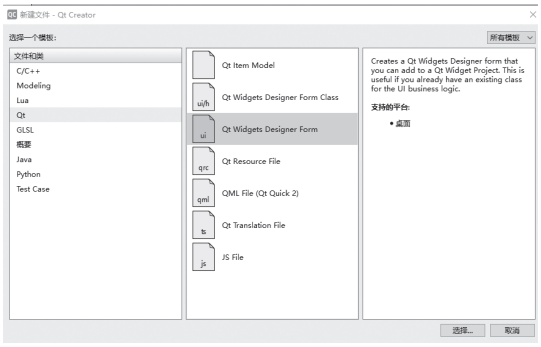

图 4–7　新建窗体文件

学生信息需要用一个类来实现，因此在项目中添加类。具体操作：在资源管理器中，选中项目名称，点击右键，在弹出菜单中选择“添加新文件…”，在弹出对话框中，左边选择“C/C++”，然后在中间选择“C++ Class”，设置类的名字为 Student，系统自动会创建 Student.h 文件和 Student.cpp 文件。另外，在本项目中需要一些全局变量和公共函数，因此添加“globaldata.h”头文件和对应的 cpp 文件。所有文件添加完后，在项目资源管理器所看到的文件如图 4–9 所示。

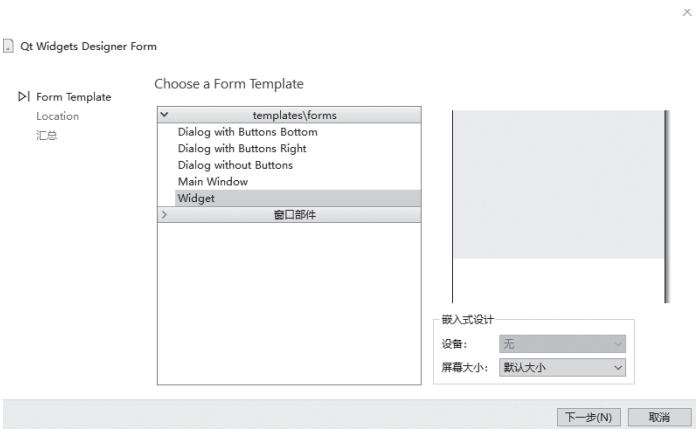

图 4–8　选择 Form 类型

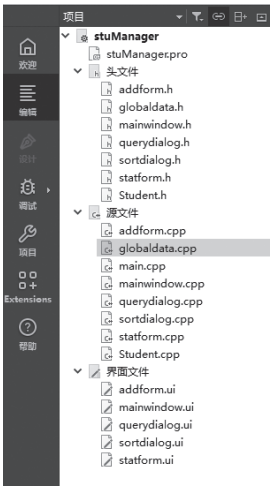

图 4–9　完整项目文件结构

4.3.1 全局变量和公共函数

在 globaldata.h 文件中定义了本案例实现过程中需要用到的全局变量，globaldata.h 文件中的完整代码如下：

```
#ifndef GLOBALDATA_H
#define GLOBALDATA_H
#include <QString>
#include <list>// 用于存储学生基本信息的双向链表
#include <QTableWidget>
#include "Student.h"
using namespace std;
// 定义全局变量
#define  MAXSIZE 100
extern QString titles[6];// 表格的标题，包括学号，姓名，语文成绩，数学成绩等
extern QString original;// 双击单元格操作时，单元格原始值
extern bool sendstuflag;
extern list<Student> mystulist;// 学生信息表
namespace ComFun {
    void xsetRowdata(QTableWidget *qtw,Student stu,int row);
    int findbysno(list<Student> stulist,QString sno);
```

```
}
#endif // GLOBALDATA_H
```

globaldata.cpp 文件中的完整代码如下：

```
#ifndef GLOBALDATA_CPP
#define GLOBALDATA_CPP
#include <QTableWidget>
#include <QTableWidgetItem>
#include "Student.h"
#include "globaldata.h"
bool sendstuflag; // 如果是从添加窗体返回，为 true, 如果是双击表格单元格后为 false
namespace ComFun {// 实现 globaldata.h 声明的命名空间中的函数
    void xsetRowdata(QTableWidget *qtw,Student stu,int row){
// 实现把 stu 内容放在表格 qtw 第 row 行
        QStringList dataList;
// 将一条学生信息复制到 dataList 中，便于在 table 中显示
        stu.copydatas2stringlist(dataList);
// 设置插入行
        qtw->insertRow(row);
        for (int col = 0; col < dataList.size(); ++col)
        {
            QTableWidgetItem *pItem = new QTableWidgetItem(dataList[col]);
            qtw->setItem(row, col, pItem);
        }
    }
    // 顺序查找的方式按学号在 list 对象中查找，返回下标
    int findbysno(list<Student> stulist,QString sno){
        list<Student>::iterator it;
        if(stulist.empty())
            return -1;
        int index=0;
        for(it=stulist.begin();it!=stulist.end();++it){// 顺序查找
            if(it->getSno()==sno)
                return index;
            index++;
        }
        return -1;
    }
}
#endif // GLOBALDATA_CPP
```

4.3.2 学生类的实现

前面已经在工程文件中添加了 Student.cpp 和 Student.h 文件，现分别叙述文件

中的代码。

在 Student.h 文件中定义学生类 Student，成员变量包括学号、姓名，语文、数学、英语以及平均分，成员函数主要是用于设置和获取私有变量的公共接口函数以及计算平均分等函数的声明，函数的实现放在 Student.cpp 中。student.h 中的完整代码如下：

```
#ifndef STUDENT_H
#define STUDENT_H
#include <QString>
class Student{
private:
    QString sno; // 学号
    QString sname; // 姓名
    double Chinese; // 语文成绩
    double Maths; // 数学成绩
    double English; // 英语成绩
    double Average; // 平均分
// 定义各个成员变量的 getter 和 setter 函数，为外界访问类的私有变量提供接口
public:
    QString getSno() const;
    void setSno(const QString &value);
    QString getSname() const;
    void setSname(const QString &value);
    double getChinese() const;
    void setChinese(double value);
    double getMaths() const;
    void setMaths(double value);
    double getEnglish() const;
    void setEnglish(double value);
    double getAverage() const;
    void setAverage(double value);
    void comAverage();
// 将一个学生信息的各字段内容复制到一个 QStringList 对象中
    void copydatas2stringlist(QStringList &strlist) const;
// 以下 5 个函数是自定义比较函数，方便排序使用
    static bool compareBysNo(const Student& p1, const Student& p2) {
        return p1.sno<p2.sno; // 根据学号比较大小
    }
    static bool compareBysName(const Student& p1, const Student& p2) {
        return p1.sname<p2.sname; // 根据姓名比较大小
    }
    static bool compareByChinese(const Student& p1, const Student& p2) {
        return p1.Chinese<p2.Chinese; // 根据语文成绩比较大小
    }
```

```
    static bool compareByMaths(const Student& p1, const Student& p2) {
        return p1.Maths<p2.Maths; // 根据数学成绩比较大小
    }
    static bool compareByEnglish(const Student& p1, const Student& p2) {
        return p1.English<p2.English; // 根据英语成绩比较大小
    }
    static bool compareByAverage(const Student& p1, const Student& p2) {
        return p1.Average<p2.Average; // 根据平均分比较大小
    }
};
#endif // STUDENT_H
```

在 Student.h 中定义了类，除了比较功能函数以外，其他的成员函数的功能需要在 Studen.cpp 中实现，主要是私有成员的设置与获取功能的实现，Studen.cpp 中的具体代码如下：

```
#include "Student.h"
#include <QString>
#include <QStringList>
QString Student::getSname() const{// 获取姓名
    return sname;
}
void Student::setSname(const QString &value){ // 设置姓名
    sname = value;
}
double Student::getChinese() const{// 获取语文成绩
    return Chinese;
}
void Student::setChinese(double value){// 设置语文成绩
    Chinese = value;
}
double Student::getMaths() const{// 获取数学成绩
    return Maths;
}
void Student::setMaths(double value){// 设置数学成绩
    Maths = value;
}
double Student::getEnglish() const{// 获取英文成绩
    return English;
}
void Student::setEnglish(double value){// 设置英语成绩
    English = value;
}
```

```
double Student::getAverage() const{// 获取平均分
    return Average;
}
void Student::setAverage(double value){// 设置平均分
    Average = value;
}
void Student::comAverage(){// 计算平均分
    Average=(Chinese+Maths+English)/3;
}
void Student::copydatas2stringlist(QStringList &strlist) const{
// 把学号 , 姓名 , 各科成绩都拷贝到 strlist 中
    strlist <<sno << sname<< QString::number(Chinese, 'f',2)
        << QString::number(Maths, 'f',2)<< QString::number(English, 'f',2)
            << QString::number(Average, 'f',2);
}
QString Student::getSno() const{// 获取学号
    return sno;
}
void Student::setSno(const QString &value){// 设置学号
    sno = value;
}
```

4.3.3 系统主界面

在 mainwindow.ui 设计界面中，拖入一个 QTableWidget 对象用来显示每个学生的成绩信息，另外拖入 7 个“QpushButton”按钮，分别用来实现“添加”“删除”“查找”“排序”“保存”“统计”“从文件中读取信息”等功能。点击按钮会触发按钮对象的 clicked 信号，调用相应的槽函数实现添加、删除等。主界面设计效果如图 4–10 所示，首次打开主界面时，因为没有数据，只有“加载信息”和“添加”命令按钮有效。

在此案例中，可以将学生信息表 mystulist 的数据随时读入到 QTableWidget 中，也可以将 QTableWidget 中的数据写入 mystulist。mystulist 是一个 list 类的对象，用来存储学生的信息。要在 QTableWidget 显示表头，先要设定 QTableWidget 列数，将表头字符串数组复制到一个 QStringList 对象中，然后才能设置表头。而与插入一行学生数据类似，先将数据复制到一个 QStringList 对象中，然后插入到指定行。对于学生信息的每个属性，都要生成一个 QTableWidgetItem 的对象，然后用 QTableWidget 的 setItem 方法将新创建的 QTableWidgetItem 对象设置到 QTableWidget 的指定行和列中。可以在主界面的 QTableWidget 上，通过双击单元格修改其内容。修改完内容后需要校验数据的有效性（成绩是否为数值，是否在 0~100 以内，学号是否重复等），可以在槽函数 on_tableWidget_itemChanged 中进行校验。在 on_tableWidget_itemDoubleClicked 中捕获，并处理单元格双击事件。

图 4–10　系统主界面

主界面对应的代码在 mainwindow.h 和 mainwind.cpp 中，其中 mainwindow.h 中的完整代码如下：

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <QTableWidget>
#include <QTableWidgetItem>
#include "addform.h"
#include "statform.h"
#include "querydialog.h"
#include "sortdialog.h"
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    void initui();
    void initvalue();
    void readlist2table();
    void savetable2list();
    int  findbyno(list<Student> stulist,QString stuno);
    void setBtnEnable(bool flag);
      bool modifyElementAtPosition(list<Student>& myList, size_t position,int col, QVariant
newValue);
private slots:
    void receiveStudent(Student stu);// 接受从 AddForm 传来的学生信息
```

```
    void on_btnaddstu_clicked(); // 添加
    void on_btnsavefile_clicked();// 保存
    void on_btnreadfile_clicked();// 读信息
    void on_btndelstu_clicked();// 删除
    void on_btnstat_clicked();// 统计
    void on_btnquery_clicked();// 查询
    void on_tableWidget_itemChanged(QTableWidgetItem *item);
    void on_tableWidget_itemDoubleClicked(QTableWidgetItem *item);
    void onEditFinished(const QModelIndex &index);
    void on_btnsort_clicked();
private:
    Ui::MainWindow *ui;
    AddForm  *addform;
    StatForm *statform;
    QueryDialog *querydialog;
    SortDialog *sortdialog;
signals:
    void sendstudentlist(list<Student> stulist);
};
#endif // MAINWINDOW_H
```

mainwindow.cpp 文件中的代码如下：

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "addform.h"
#include "globaldata.h"
#include "globaldata.cpp"
#include "Student.h"
#include <cstdlib>
#include <QDebug>
#include <QString>
#include <QFileDialog>
#include <QMessageBox>
#include <list>
#define countof(arr) (sizeof(arr) / sizeof(arr[0]))
QString titles[6];
list<Student> mystulist;// 用于存储学生信息的 list 对象
QString original;
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
```

```
    initvalue();// 调用初始化变量值函数
    initui();// 调用初始化 UI 函数
    addform=new AddForm(this); // 构建添加信息窗体实例
    statform=new StatForm(this);// 统计窗体
    querydialog=new QueryDialog(this);// 查询对话框
    sortdialog=new SortDialog(this);// 排序对话框
    connect(addform, &AddForm::sendStudent, this, &MainWindow::receiveStudent);
//sendStudent 为信号，receiveStudent 为槽函数，实现子窗体 (addform) 向主窗体传送添加的学生信息
    connect(this,&MainWindow::sendstudentlist,addform,&AddForm::getStulist);
// 主窗体将学生表传递给子窗体
    connect(this,&MainWindow::sendstudentlist,statform,&StatForm::getStulist);  connect(this,&MainWindow::sendstudentlist,querydialog,&QueryDialog::getStulist);
    connect(this,&MainWindow::sendstudentlist,sortdialog,&SortDialog::getStulist);
}
MainWindow::~MainWindow()
{
    delete ui;
}
void MainWindow::initui() // 初始化界面
{
    QStringList headerList;
    QString str;
    for(unsigned int i=0;i<countof(titles);i++){
// 把学生属性连接成一个 QStringList 对象，才能在 tableWidget 上显示列表头标题
        str=titles[i];
        headerList<<str;
    }
        // 设置表格列数为学生属性数
    ui->tableWidget->setColumnCount(headerList.size());
        // 设置列表头标题
    ui->tableWidget->setHorizontalHeaderLabels(headerList);
        // 设置可以通过单击列标题进行排序
    ui->tableWidget->setSortingEnabled(true);
    setBtnEnable(false); // 设置一些按钮不可用
    sendstuflag=false;
}
void MainWindow::initvalue()
{// 表头标题，方便一次性设置表头
    titles[0]=" 学号 ";
    titles[1]=" 姓名 ";
    titles[2]=" 语文 ";
    titles[3]=" 数学 ";
    titles[4]=" 英语 ";
    titles[5]=" 平均分 ";
```

```
}
void MainWindow::readlist2table()
{// 从学生信息表 mystulist 中读取学生信息到 tableWidget 中
    this->ui->tableWidget->clearContents();// 先清空表格内容
    list<Student>::iterator it; // 定义迭代指示器
    int index=0;
    if(!mystulist.empty()){// 遍历整个表
        for(it=mystulist.begin();it!=mystulist.end();++it){
            ComFun::xsetRowdata(ui->tableWidget,*it,index++);
        }
    }
// 设置表格数据行数
    ui->tableWidget->setRowCount(mystulist.size());
    if(mystulist.size()){
        // 如果学生信息表不为空，设置 " 排序 " " 统计 " 等按钮可用
        setBtnEnable(true);
    }
}
void MainWindow::savetable2list()// 将 tableWidget 中的学生信息保存到 mystulist 中
{
    int rows;
    rows=ui->tableWidget->rowCount();// 获取表格行数
    int i=0;
    Student stu;
    QString str;
    while(i<rows){ // 读取表格 i 行 0 列单元格数据到字符串 str, 需要 text() 函数转换
        str=ui->tableWidget->item(i,0)->text();
// 得到第 0 列数据，即学号
        if(str!=nullptr && !str.isEmpty())
            stu.setSno(str);
        str=ui->tableWidget->item(i,1)->text();// 第 1 列：姓名
        if(str!=nullptr && !str.isEmpty())
            stu.setSname(str);
        str=ui->tableWidget->item(i,2)->text();
        if(str!=nullptr && !str.isEmpty())
            stu.setChinese(str.toFloat());
        str=ui->tableWidget->item(i,3)->text();
        if(str!=nullptr && !str.isEmpty())
            stu.setMaths(str.toFloat());
        str=ui->tableWidget->item(i,4)->text();
        if(str!=nullptr && !str.isEmpty())
            stu.setEnglish(str.toFloat());
        str=ui->tableWidget->item(i,5)->text();
        if(str!=nullptr && !str.isEmpty())
```

```
                stu.setAverage(str.toFloat());
            i++;
            // 将一个学生信息附加到表最后
            mystulist.push_back(stu);
        }
}
void MainWindow::setBtnEnable(bool flag)
{// 设置一些按钮可用，包括："排序""统计""查询""删除""保存到文件"
    ui->btnsort->setEnabled(flag);
    ui->btnstat->setEnabled(flag);
    ui->btnquery->setEnabled(flag);
    ui->btndelstu->setEnabled(flag);
    ui->btnsavefile->setEnabled(flag);
}
bool MainWindow::modifyElementAtPosition(list<Student> &myList, size_t  position,int col,
QVariant newValue)
{// 修改 myList 学生信息表第 position 行，第 col 列的值为 newValue
    if (position >= myList.size()) {
        return false;
 }
    // 创建一个迭代器并将其移动到列表的起始位置
    auto it = myList.begin();
    // 将迭代器向前移动到指定位置
    std::advance(it, position);
    // 修改指定位置的元素
    Student stu=*it;// 暂时保存迭代器所指向数据
    switch (col){
        case 1:stu.setSname(newValue.toString());break;// 第 1 列，设置姓名
        case 2:stu.setChinese(newValue.toDouble());break; // 第 2 列，设置语文成绩
        case 3:stu.setMaths(newValue.toDouble());break;
        case 4:stu.setEnglish(newValue.toDouble());break;
        case 5:stu.setAverage(newValue.toDouble());break;
        default:
            stu.setSno(newValue.toString());// 第 0 列，设置学号
    }
        *it=stu;// 修改某个学生的信息后保存回 mylist 的指定行
    return true;
}
void MainWindow::receiveStudent(Student stu){// 槽函数，接受一个学生信息
    stu.comAverage();// 计算语数外成绩平均值
    int selectedRow = ui->tableWidget->currentRow();// 获取表格选中行
    if (selectedRow >= 0){
// 如果是选中某一行进行插入，学生信息插入到 list 表第 row 个位置
        list<Student>::iterator it;
        int i=0;
```

```
            it=mystulist.begin();
            while(i<selectedRow) {it++;i++;};// 计数和移动迭代器
            mystulist.insert(it,stu);
// 将接受到的一个学生信息保存到 mystulist 中指定位置
        }
        else          // 没有选中任何一行，添加学生信息在最后
            mystulist.push_back(stu);
        readlist2table();// 把 list 表信息复制到表格中显示
}
void MainWindow::on_btnaddstu_clicked()
{// 当按钮被点击时，调用 addform->exec() 来显示添加学生信息窗体 (addform)
        addform->exec();
}
void MainWindow::on_btnsavefile_clicked()
{// 关于文件操作详细内容，可以查看 "Qt 的文件操作 " 小节
// 打开保存对话框
        QString filename=QFileDialog::getSaveFileName(
            this,tr("Save File"),"",tr("Text Files (*.txt);;Data Files (*.dat);;All Files (*)"));
        if (filename.isEmpty())
            return;
        QFile file(filename); // 文件对象
        if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) {
            QMessageBox::warning(this, tr("Error"), tr("Cannot open file for writing"));
            return;
        }
        QTextStream out(&file);
        QString outputstr,str;
        int rows,cols; // 获取表格行数和列数
        rows=ui->tableWidget->rowCount();
        cols=ui->tableWidget->columnCount();
        for(int i=0;i<rows;i++){
            for(int j=0;j<cols;j++){// 获取表格每个单元格
                QTableWidgetItem *item = ui->tableWidget->item(i, j);
                if(item!=nullptr && !item->text().isEmpty())
                // 如果不为空，保存到字符串
                    str=item->text();
                outputstr.append(str+" ");// 添加到输出字符串
                qDebug()<<str;
            }
            outputstr.append("\n");
        }
        out<<outputstr; // 输出
        file.close();
}
void MainWindow::on_btnreadfile_clicked()
```

```
{// 从文件读内容（学生信息）到表格
        QString text,aline;
        QString fileName = QFileDialog::getOpenFileName(this, tr("Open File"), "",
                                tr("Text Files (*.txt);;All Files (*)"));

        if(ui->tableWidget->rowCount()){
            QMessageBox::information(this, tr("read from file"), tr(" 原表不为空 "));
                return;
        }
        mystulist.clear();// 清除 list 表
        if (fileName.isEmpty())
            return;
    QFile file(fileName);
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
            QMessageBox::warning(this, tr("Error"), tr("Cannot open file for reading"));
            return;
        }
        QTextStream in(&file); // 文件输入流
        QStringList qslist;
        int i=ui->tableWidget->rowCount();
        while(!in.atEnd()){// 当输入流没有到最尾端，设置表格在第 i 行进行插入
            ui->tableWidget->insertRow(i);
            aline=in.readLine();// 读取一行
            qslist=aline.split(" ");
        // 一行文本按空格分开，分别作为学生信息每个字段
            for(int j=0;j<qslist.length();j++){
                QTableWidgetItem *pItem = new QTableWidgetItem(qslist[j]);
// 每个字段信息显示在表格每个单元格内
                ui->tableWidget->setItem(i,j,pItem);
            }
            i++;
    }
    file.close();
    ui->tableWidget->update();
    if(i){ // 如果表格有数据，表格内容保存到 list 表中
        savetable2list();
        setBtnEnable(true); // 设置排序、统计、查询等按钮可用
    }
}
void MainWindow::on_btndelstu_clicked()// 删除按钮功能
{
        Student stu;
// 获取表格选中行号
```

```
        int selectedRow = ui->tableWidget->currentRow();
        if (selectedRow >= 0) // 如果选中某一行，则实现 list 表删除操作
        {
            list<Student>::iterator it;
            int i=0;
            it=mystulist.begin();
            while(i<selectedRow) {it++;i++;}
            mystulist.erase(it);// 调用 std::list 的 erase 函数删除指定记录
            ui->tableWidget->removeRow(selectedRow);// 在表格中删除指定行
        }
        if(!mystulist.size())// 如果 list 表为空
            setBtnEnable(false);
}
void MainWindow::on_btnstat_clicked()
{
// 这几个按钮程序类似，都是先用 emit 发送 " 传递学生 list 表 " 的信号，让对应窗体或对话框接受，然后执行对应窗体或对话框的 exec 函数，让其显示运行
    emit this->sendstudentlist(mystulist);
    statform->exec();
}
void MainWindow::on_btnquery_clicked()
{
    emit this->sendstudentlist(mystulist);
    querydialog->exec();
}
void MainWindow::on_btnsort_clicked()
{
    emit this->sendstudentlist(mystulist);
    sortdialog->exec();
}
void MainWindow::on_tableWidget_itemChanged(QTableWidgetItem *item)
{// 修改表格一个单元格后数据校验
    bool ok;
    double mark;
// 获取单元格列号和行号
    int col=item->column();
    int row=item->row();
    if(col>1){ // 成绩字段
        // 单元格数据转换为浮点型
        mark=item->text().toDouble(&ok);
        if(!ok){ // 单元格数据不是浮点型
            QMessageBox::warning(this, tr("Error"), tr(" 成绩格式不对 "));
// 恢复为修改前的数据，original 数据是在 itemDoubleClicked 事件后保存的原始值
            item->setText(original);
            return;
```

```
            }
            if(mark<0 || mark>100){
                QMessageBox::warning(this, tr("Error"), tr(" 成绩格式不对 "));
                item->setText(original);// 恢复编辑前的数据
                return;
            }
        }
        if(!sendstuflag &&col==0){
//col==0 表示修改的学号
// sendstuflag 为 true 表示表格数据为其他窗体传送而来
// 为 false 表示是双击单元格后修改数据
// 这里只对双击单元格后修改数据后的情况进行学号唯一性校验
// 下面是调用公共函数，按学号在 list 表进行顺序查找，查找失败返回 -1
            int index=ComFun::findbysno(mystulist,item->text());
            if(index!=-1){ //list 表中已存在 item->text() 的学号
                QMessageBox::warning(this, tr("Error"), tr(" 已存在此学号 "));
                item->setText(original); // 恢复编辑前的数据
                return;
            }
        }
// 修改学生信息表指定位置的数据
                modifyElementAtPosition(mystulist,row,col,item->text());
}
void MainWindow::on_tableWidget_itemDoubleClicked(QTableWidgetItem *item)
{// 双击表格一个单元格后保存原数据
        original=item->text();
        qDebug()<<"original="<<original;
//sendstuflag 为 false 表示是双击单元格后修改数据 , 要进行学号唯一性校验
        sendstuflag=false;
}
void MainWindow::onEditFinished(const QModelIndex &index)
{// 测试成绩，在单元格编辑后显示行号、列号和修改后数据
        // 获取编辑后的单元格数据
        QTableWidgetItem *item = ui->tableWidget->item(index.row(), index.column());
        if (item) {
            QString text = item->text();
            qDebug() << "Cell (" << index.row() << ", " << index.column() << ") edited. New
value: " << text;
        }
}
```

4.3.4 加载信息

在主界面上，单击“加载信息”按钮，打开文件对话框，选择事先准备好的文本文件 student.txt，文件中保存了学生的基本信息，如图 4-11 所示。系统会把

student.txt 中保存的所有学生的信息加载到主界面的 QTableWidget 对象中。实现加载功能的代码见 mainwindow.cpp 中的 on_btnreadfile_clicked() 函数，加载后的结果如图 4–12 所示。

图 4–11　文件中的内容

图 4–12　加载信息显示结果

4.3.5 添加学生数据

在本案例中，窗体之间的数据传递是实现系统功能的关键，它保证窗体间信息交互时高效且流畅，让子窗体操作能实时反馈到主窗体，反之亦然。主窗体是系统核心，展示学生信息全貌并提供主要操作入口。子窗体处理特定功能，如添加、查询学生信息等。二者数据传递围绕学生信息的添加、修改等操作展开，通过信号与槽机制实现。

当用户在图 4–10 所示主窗体中单击“添加”按钮时，主窗体调用相应的函数打开添加学生信息的子窗体，如图 4–13 所示。

在子窗体中，用户可以输入要添加学生的各项信息，包括学号、姓名，语文、数学、英语成绩。子窗体就像一个信息收集站，将用户输入的这些信息整合起来，创建一个对象来存储这些信息。

图 4–13　添加信息

子窗体在将学生信息传递给主窗体之前，会进行数据验证，检查学号是否已经存在于系统中（避免重复录入），同时检查各科成绩是否在合理的范围（0~100 分）内。如果检测到学号重复或者成绩不合理，子窗体就会弹出相应的提示框告知用户，让用户重新输入。如果数据验证通过，子窗体就会通过信号机制发送一个信号，这个信号就像是一个“包裹”，里面装着新创建的 Student 类的对象；子窗体将这个“包裹”发送出去，等待主窗体接收。主窗体预先设置好了一个槽函数来接收

子窗体发送的信号，当主窗体接收到信号后，就会打开这个“包裹”，取出里面的 Student 对象，然后主窗体根据当前表格的选中行情况，将新添加的学生信息插入 mystulist 对象中。最后，主窗体更新表格的显示，将新学生的信息展示在表格中。

与 addform.cpp 文件中的完整代码

添加学生信息的主要代码在 addform.h 和 addform.cpp 文件中，文件的完整代码分别如下：

```
#ifndef CHILDFORM_H
#define CHILDFORM_H
#include <QWidget>
#include <QDialog>
#include "globaldata.h"
namespace Ui {
class AddForm;
}
class AddForm : public QDialog
{
    Q_OBJECT
public:
    explicit AddForm(QWidget *parent = nullptr);
    ~AddForm();
    void initui();
    int checkmark(double mark);
    void getStulist(list<Student>  stulist);
private slots:
    void on_pushButton_clicked();
private:
    Ui::AddForm *ui;
    list<Student> stulist;
signals:
    void sendStudent(Student stu);
};
#endif // CHILDFORM_H
```

```
#include "addform.h"
#include "ui_addform.h"
#include "globaldata.h"
#include <QMessageBox>
AddForm::AddForm(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::AddForm)
{
    ui->setupUi(this);
```

```
    initui();
}
AddForm::~AddForm()
{
    delete ui;
}
void AddForm::initui()// 初始化添加界面上的输入文本框均为空
{
    ui->txtsNo->setPlainText("");
    ui->txtsName->setPlainText("");
    ui->txtChinese->setPlainText("");
    ui->txtsMaths->setPlainText("");
    ui->txtEnglish->setPlainText("");
}
int AddForm::checkmark(double mark)// 检测输入成绩是否合法
{
    if(mark<0 || mark>100)
        return 0;
    else
        return 1;
}
void AddForm::getStulist(list<Student> stulist) // 槽函数，接受 list 表
{
    this->stulist=stulist;
}
void AddForm::on_pushButton_clicked()// 提交表单
{
    Student stu;
    int index;
    QString sno;
    sendstuflag=true; // sendstuflag 为 true 是主窗体表格数据为其他窗体传送而来
      sno=ui->txtsNo->toPlainText();// 获取学号字符串；调用公共函数，按学号在 list 表
进行顺序查找，查找失败返回 -1
    index=ComFun::findbysno(mystulist,sno);
    if(index!=-1){
        QMessageBox::warning(this, tr("ADD FORM"), tr(" 已存在此学号 "));
        ui->txtsNo->setPlainText("");
        return;
    }
    else{// 学生信息对象设置学号姓名等字段值
        stu.setSno(sno);
        stu.setSname(ui->txtsName->toPlainText());
        stu.setChinese((ui->txtChinese->toPlainText()).toFloat());
        stu.setMaths((ui->txtsMaths->toPlainText()).toFloat());
        stu.setEnglish((ui->txtEnglish->toPlainText()).toFloat());
```

```
            // 检验语数外成绩
            int x=checkmark(stu.getChinese());
            int y=checkmark(stu.getMaths());
            int z=checkmark(stu.getEnglish());
            if(!x ||!y || !z){
                QMessageBox::information(nullptr, " 出错 ", " 成绩有误，请检查 ");
            }else{
                emit this->sendStudent(stu); // 发送信号，传送一个学生信息
                this->close();
            }
        }
    }
```

4.3.6 删除信息

在学生信息管理系统中，删除功能可让用户移除不需要的学生信息。在主窗体操作时，若要删除学生信息，需先在 QTableWidget 表格中选中目标行，再单击“删除”按钮，启动删除流程。系统会先获取选中行号，若未选中任何行，删除操作终止；若有选中行，系统记录行号，后续据此定位并删除对应学生信息。

完成删除操作后，系统会检查 mystulist 中是否还存在其他学生信息。如果为空，这意味着系统中已经没有学生信息。此时，系统会禁用与学生信息操作相关的一些功能按钮，例如“排序”“统计”“查询”等，避免用户进行无效操作。删除功能代码见 mainwindow.cpp 中的 on_btndelstu_clicked 函数。

4.3.7 修改功能的实现

数据修改功能主要是通过双击元格实现的。主要程序在 mainwindow.cpp 中的 on_tableWidget_itemChanged 函数中。这个函数的参数是 QTableWidgetItem 类型的指针 *item，用以指示双击的单元格对象。可以通过 item 对象获取单元格所在的行号和列号。如果点击的单元格不是第 1 列（学号列），就只进行数据格式的检查。如果是学号列，首先检查变量 sendstuflag 的值，因为从其他窗体（主要是增加数据窗体）传送数据到主窗体也会触发 itemChanged 事件，这时 sendstuflag 为 true。而双击单元格时，sendstuflag 为 false，这时就要检查修改后的学号是否已经存在，如果存在，报错，恢复学号值为原始值。最后，如果正常修改，还要更新内存中 mystulist 中对应位置的学生信息值。

4.3.8 查询功能的实现

该功能主要由查询子窗体实现，查询界面如图 4–14 所示，允许用户根据学号或姓名进行查找，并将符合条件的学生信息显示在 QTableWidget 对象中。该程序主要使用顺序查找算法。

图 4–14　查询功能界面

查询时，用户先在下拉框中选查询条件，再在输入框中输入具体查询值。单击“查询”按钮后，系统会清空 TableWidget 中的信息避免旧数据干扰，然后根据所选条件调用对应查询函数。

若选“学号”，调用 querybyNo 函数，该函数遍历学生信息，该信息是 list<Student> stulist，将学号与查询值比对，找到匹配项就把学生信息显示在查询窗口的表格中；若未找到，弹出“没有数据”警告框。若选“姓名”，调用 querybyName 函数，它同样遍历 stulist，找出姓名匹配的学生信息并显示在表格。因可能有重名，函数会遍历以显示所有匹配结果；若未找到，也弹出“没有数据”警告框。

查询功能代码主要在 querydialog.h 和 querydialog.cpp 中，其中 querydialog.h 的完整代码如下：

```
#ifndef QUERYDIALOG_H
#define QUERYDIALOG_H
#include <QDialog>
#include <list>
#include "globaldata.h"
using namespace std;
namespace Ui {
class QueryDialog;
}
class QueryDialog : public QDialog
{
    Q_OBJECT
public:
    explicit QueryDialog(QWidget *parent = nullptr);
    ~QueryDialog();
    void getStulist(list<Student> stulist);
    void initui();
    void initvalues();
    void querybyNo(list<Student>  stulist,QString queryvalue);
    void querybyName(list<Student> stulist,QString queryvalue);
```

```
private slots:
    void on_btnReturn_clicked();
    void on_btnQuery_clicked();
private:
    Ui::QueryDialog *ui;
    list<Student> stulist;
};
#endif // QUERYDIALOG_H
```

querydialog.cpp 文件中的完整代码如下：

```
#include "querydialog.h"
#include "ui_querydialog.h"
#include "globaldata.h"
#include <QMessageBox>
#include <QDebug>
#define countof(arr) (sizeof(arr) / sizeof(arr[0]))
QueryDialog::QueryDialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::QueryDialog)
{
    ui->setupUi(this);
    initvalues();// 初始化一些变量的值
    initui();// 初始化界面
}
QueryDialog::~QueryDialog()
{
    delete ui;
}
void QueryDialog::getStulist(list<Student> stulist)
{// 获取从主窗体传来的学生信息表对象
    this->stulist=stulist;
}
void QueryDialog::initui()
{
//comboBox 设置查询选项
    ui->comboBox->addItem(" 学号 ");
    ui->comboBox->addItem(" 姓名 ");
    QStringList headerList;
    QString str;
// 把字段名设置到表头
    for(unsigned int i=0;i<countof(titles);i++){
        str=titles[i];
        headerList<<str;
    }
```

```
// 设置列数
    ui->tableWidget->setColumnCount(headerList.size());
// 设置表头
    ui->tableWidget->setHorizontalHeaderLabels(headerList);
// 设置可以通过点击列标题进行排序
    ui->tableWidget->setSortingEnabled(true);
}
void QueryDialog::initvalues()
{
        titles[0]=" 学号 ";
        titles[1]=" 姓名 ";
        titles[2]=" 语文 ";
        titles[3]=" 数学 ";
        titles[4]=" 英语 ";
        titles[5]=" 平均分 ";
}
void QueryDialog::querybyNo(list<Student>  stulist, QString queryvalue)
{// 查询窗体的按学号进行查询，与公共程序类似
    int index=0;
    list<Student>::iterator it;
    ui->tableWidget->clearContents();// 先清除表格的内容
    for(it=stulist.begin();it!=stulist.end();++it){
        index++;
        if(it->getSno()==queryvalue)// 如果找到对应学号的记录
            break;
    }
    if(index>=static_cast<int>(stulist.size()))
        QMessageBox::warning(this, tr("Notice!"), tr(" 没有数据 "));
    else// 查询到数据后，显示在表格
        ComFun::xsetRowdata(ui->tableWidget,*it,0);
}
void QueryDialog::querybyName(list<Student> stulist, QString queryvalue)
{// 通过姓名查找
    int count=0;
    Student students[MAXSIZE]; // 查询结果集合
    list<Student>::iterator it;
    ui->tableWidget->clearContents();// 清除表格内容
    for(it=stulist.begin();it!=stulist.end();++it){
        if(it->getSname()==queryvalue){
// 按姓名查询到记录，则将结果加入到结果集合中
            students[count]=*it;
// 添加到表格中

            ComFun::xsetRowdata(ui->tableWidget,*it,count++);
        }
    }
```

```
        if(!count){// 没有找到对应记录
            QMessageBox::warning(this, tr("Notice!"), tr(" 没有数据 "));
        }
        else{
            ui->tableWidget->setRowCount(count);
        }
}
void QueryDialog::on_btnReturn_clicked()
{// 关闭此对话框
    this->close();
}
void QueryDialog::on_btnQuery_clicked()
{// 查询按钮
    int queryCategory=ui->comboBox->currentIndex();
// 根据不同类别进行不同查询
    if(queryCategory==0)// 按学号查询
        querybyNo(stulist,ui->lineEdit->text());
    else// 按姓名查询
        querybyName(stulist,ui->lineEdit->text());
}
```

4.3.9 排序

排序功能允许用户按照不同的字段（如学号、姓名、各科成绩等）对学生信息进行升序或降序排列。该功能主要由排序子窗体实现，用户可以在该窗体中选择排序字段和排序方式（升序或降序），系统根据用户的选择对存储学生信息的线性表进行排序，并将排序结果更新显示在表格中，排序窗口如图 4-15 所示。

设置排序字段选项：在下拉框中添加可供选择的排序字段。设置排序方式选项：默认选中“升序”排序方式，同时也提供“降序”选项。用户可以根据需要切换排序方式。

当用户单击排序子窗体中的“排序”按钮时，系统获取用户选择的排序字段和排序方式。从下拉框中获取用户选择的排序字段，用单选按钮选择排序方式（升序或降序）。调用排序函数：根据用户的选择，调用标准库排序函数 Sort 进行排序操作。std::sort 是 C++ 标准库中用于对元素进行排序的函数，位于 <algorithm> 头文件中。它采用了一种混合排序算法，通常是快速排序和插入排序结合的算法，能够在大多数情况下提供接近 O(nlog n) 的时间复杂度。在这里，std::sort 函数会根据排序字段的不同，采用不同的比较规则对 list 表中的学生信息进行排序，因此需要通过自定义比较函数，将其传递给 std::sort 函数，从而实现不同字段的排序。默认排序方法是升序，实现降序排序只需将升序结果反转。

排序完成后，需要更新线性表和表格的显示，以反映排序后的结果。

图 4-15　排序窗口

排序功能代码主要在 sortdialog.h 和 sortdialog.cpp 中，其中 sortdialog.h 的完整代码如下：

```
#ifndef SORTDIALOG_H
#define SORTDIALOG_H
#include <QDialog>
#include <list>
// #include <QList>
#include "globaldata.h"
using namespace  std;
namespace Ui {
class SortDialog;
}
class SortDialog : public QDialog
{
    Q_OBJECT
public:
    explicit SortDialog(QWidget *parent = nullptr);
    void getStulist(list<Student> stulist);
    ~SortDialog();
    void initui();
    void readlist2table();
    void sort(int index,int mode);
private slots:
    void on_btnReturn_clicked();
    void on_btnsort_clicked();
    void on_btnreaddata_clicked();
private:
    Ui::SortDialog *ui;
    list<Student> stulist;
```

```
};
#endif // SORTDIALOG_H
```

sortdialog.cpp 文件中的完整代码如下：

```
#include "sortdialog.h"
#include "ui_sortdialog.h"
#include "globaldata.h"
#include <QMessageBox>
#include <QDebug>
#include <list>
using namespace std;
#define countof(arr) (sizeof(arr) / sizeof(arr[0]))
SortDialog::SortDialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::SortDialog)
{
    ui->setupUi(this);
    initui();
}
SortDialog::~SortDialog()
{
    delete ui;
}
void SortDialog::initui()
{// 默认排序方式是升序
    ui->rbascend->setChecked(true);
    QStringList headerList;
    QString str;
    for(unsigned int i=0;i<countof(titles);i++){
// 学生信息不同字段都添加到 comboBox 中，可以实现不同字段排序
        ui->comboBox->addItem(titles[i]);
        str=titles[i];
        headerList<<str;
    }
// 设置表格列数
    ui->tableWidget->setColumnCount(headerList.size());
// 设置表头
    ui->tableWidget->setHorizontalHeaderLabels(headerList);
// 设置不能通过点击列标题进行排序，本程序手动排序
    ui->tableWidget->setSortingEnabled(false);
    ui->btnsort->setEnabled(false);
}
void SortDialog::readlist2table()
```

```
{// 从学生信息表 stulist 中读取数据到表格中
    if(stulist.size()){// 如果 stulist 不为空
        this->ui->tableWidget->clearContents();// 先清除表格内容
        list<Student>::iterator it;
        int index=0;
        if(!stulist.empty()){
            for(it=stulist.begin();it!=stulist.end();++it){
                ComFun::xsetRowdata(ui->tableWidget,*it,index++);
            }
        }
                    // 设置表格行数为学生表记录数
        ui->tableWidget->setRowCount(stulist.size());
    }
}
void SortDialog::sort(int index, int mode)
{// 自定义排序函数
    switch (index) {// 根据不同字段进行不同方式排序
        case 0:stulist.sort(Student::compareBysNo);break;
        case 1:stulist.sort(Student::compareBysName);break;
        case 2:stulist.sort(Student::compareByChinese);break;
        case 3:stulist.sort(Student::compareByMaths);break;
        case 4:stulist.sort(Student::compareByEnglish);break;
        default:
            stulist.sort(Student::compareByAverage);
    }
    if(mode) // 如果是降序，将升序结果翻转
        stulist.reverse();
    ui->tableWidget->clearContents();// 清除表格内容
    readlist2table();//list 表数据显示到表格
}
void SortDialog::getStulist(list<Student> stulist)
{// 设置当前对话框中的 stulist 为主窗体传来的学生信息表 stulist
    this->stulist=stulist;
}
void SortDialog::on_btnReturn_clicked()
{
    this->close();
}
void SortDialog::on_btnsort_clicked()
{
// 获取排序字段序号
    int index=ui->comboBox->currentIndex();
    int mode=0;
    if(ui->rbdescend->isChecked())
        mode=1;
```

```
        sort(index,mode);// 调用自定义排序函数，根据不同类别和方式排序
}
void SortDialog::on_btnreaddata_clicked()
{// 读取数据按钮
        readlist2table();// 读取 list 数据到表格
        if(ui->tableWidget->columnCount())// 如果表格有记录
                ui->btnsort->setEnabled(true); // 则可以排序
        ui->btnreaddata->setEnabled(false); // 不能再从列表读取数据
}
```

4.3.10 统计功能的实现

统计窗体类的初始化函数，对统计功能相关的界面元素进行初始化。这里主要是对科目和统计类别的下拉框进行初始化，将科目（语文、数学、英语）和统计类别（平均成绩、最高分、最低分、差、及格、中、良、优）添加到对应的下拉框中。当用户单击“统计”按钮时，首先获取用户选择的科目索引和统计类别索引，然后根据选择的科目，将学生信息列表 stulist 中对应科目的成绩存储到数组 datas 中。收集完数据后，调用函数进行具体的统计计算，根据传入的统计类别索引对数据数组 datas 进行不同的统计操作，如计算平均值、最大值、最小值、各分数段的人数等。最后，将统计计算的结果转换为字符串，显示在 plainTextEdit 控件中，统计功能界面如图 4–16 所示。

图 4–16　统计功能界面

统计功能实现主要在 statform.h 和 statform.cpp 文件中，其中 statform.h 中的完整代码如下：

```
#ifndef STATFORM_H
#define STATFORM_H
#include <QDialog>
#include <list>
#include "globaldata.h"
using namespace  std;
```

```
namespace Ui {
class StatForm;
}
class StatForm : public QDialog
{
    Q_OBJECT
public:
    explicit StatForm(QWidget *parent = nullptr);
    ~StatForm();
    void getStulist(list<Student>  stulist);
    void initui();
    double stat(double datas[],int n,int cate);
private slots:
    void on_btnBack_clicked();
    void on_btnStat_clicked();
private:
    Ui::StatForm *ui;
    list<Student> stulist;
};
#endif // STATFORM_H
```

statform.cpp 中的完整代码如下：

```
#include "statform.h"
#include "ui_statform.h"
StatForm::StatForm(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::StatForm)
{
    ui->setupUi(this);
    initui();
}
StatForm::~StatForm()
{
    delete ui;
}
void StatForm::getStulist(list<Student> stulist)
{// 设置当前窗体的 stulist 对象为主窗体传来的 stulist
    this->stulist=stulist;
}
void StatForm::initui()// 初始化 UI
{
    QString subjects[3]={" 语文 "," 数学 "," 英语 "};
    QString category[8]={" 平均成绩 "," 最高分 "," 最低分 "," 差 "," 及格 "," 中 "," 良 "," 优 "};
```

```
    for(int i=0;i<3;i++)
        this->ui->cbSubject->addItem(subjects[i]);
    for(int i=0;i<8;i++)
        ui->cbCategory->addItem(category[i]);
}
double StatForm::stat(double datas[], int n, int cate) // 统计函数
{
// 要统计的数据先放入 datas 数组中，然后根据统计类别 cate 进行统计
    if(!cate){ // 求平均分
        int sum=0;
        for(int i=0;i<n;i++)
            sum+=datas[i];
        return sum/n;
    }
    if(cate==1){ // 求最高分
        int max=-1;
        for(int i=0;i<n;i++)
            if(datas[i]>max)
                max=datas[i];
        return max;
    }
    if(cate==2){ // 求最低分
        int min=101;
        for(int i=0;i<n;i++)
            if(datas[i]<min)
                min=datas[i];
            return min;
    }
    if(cate==3){ // 统计不及格人数
        int count=0;
        for(int i=0;i<n;i++)
            if(datas[i]<60)
                count++;
        return count;
    }
    if(cate==4){ // 统计 60~70 之间的人数
        int count=0;
        for(int i=0;i<n;i++)
            if(datas[i]<70 && datas[i]>=60)
                count++;
        return count;
    }
    if(cate==5){
        int count=0;
```

```
            for(int i=0;i<n;i++)
                if(datas[i]<80 && datas[i]>=70)
                    count++;
            return count;
        }
        if(cate==6){
            int count=0;
            for(int i=0;i<n;i++)
                if(datas[i]<90 && datas[i]>=80)
                    count++;
            return count;
        }else{
            int count=0;
            for(int i=0;i<n;i++)
                if(datas[i]>=90)
                    count++;
            return count;
        }
}
void StatForm::on_btnBack_clicked()
{// 关闭此窗体
    this->close();
}
void StatForm::on_btnStat_clicked()
{// 统计按钮，获取统计科目序号和统计类别
    int subjectindex=ui->cbSubject->currentIndex();
    int categoryindex=ui->cbCategory->currentIndex();
    double datas[100];
    list<Student>::iterator it;
    int i=0;
    if(subjectindex==0){ // 科目序号是 0，则统计语文成绩
        for(it=stulist.begin();it!=stulist.end();++it)
            datas[i++]=it->getChinese();
    }
    if(subjectindex==1){// 统计数学成绩
        for(it=stulist.begin();it!=stulist.end();++it)
            datas[i++]=it->getMaths();
    }
    if(subjectindex==2){ // 统计英语成绩
        for(it=stulist.begin();it!=stulist.end();++it)
            datas[i++]=it->getEnglish();
    }
// 调用统计函数 stat 进行统计
    double result=stat(datas,stulist.size(),categoryindex);
// 统计结果显示在 plainTextEdit 中
```

```
    this->ui->plainTextEdit->setPlainText(QString::number(result));
}
```

4.3.11 保存

本案例程序中，执行添加、删除、修改等功能，改变的是 mystulist 中的信息，可以通过单击主界面上的“保存”按钮，将 mystulist 中的信息保存到一个文本文件中。实现代码见 mainwind.cpp 中的 on_btnsavefile_clicked() 函数。

练习

1. 图书销售管理系统

（1）任务简介

书店图书销售管理系统旨在助力书店实现图书信息的高效管理。一方面，它能对书店内的所有图书，如书名、作者、出版社、库存数量、进货价格、销售价格等信息进行数字化录入与存储。另一方面，系统提供便捷的查询功能，支持通过关键词快速定位所需图书。在日常运营中，帮助书店工作人员实时掌握库存动态，及时安排补货，避免缺货情况发生。同时，借助系统生成的销售报表，书店管理者能分析销售趋势，为采购决策提供数据支撑，提升书店整体运营效率。

（2）设计任务

每一类图书所涉及的信息有图书的 ISBN 号、名称、作者、出版社、价格、库存数量。系统能实现的操作和功能如下：

图书信息管理　实现图书信息的录入、修改、删除和查询功能，支持按书名、作者、出版社等关键词进行查询。

库存管理　实时更新图书库存数量，设置库存预警值，当库存低于预警值时自动提醒工作人员补货。

销售管理　记录图书销售信息，包括销售时间、销售数量、销售金额等，生成销售报表，支持按时间段、图书类别等进行统计分析。

2. 汽车租赁管理系统

（1）任务简介

简单的汽车租赁管理软件面向汽车租赁企业，旨在简化租赁流程，提升运营效率。软件支持录入车辆基本信息，如车型、车牌号、车架号、租赁价格等，方便快速检索与管理。租赁业务环节，能登记客户资料，跟踪订单进度，自动计算租赁费用，并支持多种支付方式。在日常运营中，系统可实时监控车辆状态，对即将到期的车辆租赁业务进行提醒。此外，还能生成运营报表，助力管理者分析车辆的使用

频率、营收状况，以便调整经营策略，实现降本增效。

（2）设计任务

车辆档案创建 录入每辆车的详细信息，包括品牌、型号、车牌号、车架号、购置时间、颜色、座位数、日租金等。

车辆状态监控 实时更新车辆状态，如可出租、已出租、维修中、已报废等，方便员工快速了解车辆可用性。

租赁订单创建 根据客户需求，创建租赁订单，包括租赁车辆、租赁时间、归还时间、租金计算等。

订单跟踪与管理 实时跟踪订单状态，如待确认、已确认、已完成、已取消等，及时处理异常情况。

租金结算 根据租赁时长、租金标准等自动计算租金，支持多种支付方式，如现金、银行卡、在线支付等。

3. 商品库存管理系统

（1）任务简介

商品库存管理系统旨在高效管理商品库存信息。借助线性表存储商品编号、名称、数量、价格等数据。系统可实现商品的入库、出库操作，能实时更新库存数量。支持按多种条件查询商品，还能生成库存报表，为企业合理规划库存、降低成本提供有力支持。

（2）设计任务

商品信息管理 实现商品信息的添加功能，包括商品编号、名称、规格、单价、供应商等必要信息。实现商品信息的修改和删除操作，确保数据的准确性和及时性。支持按商品编号、名称等关键字进行商品信息的查询功能。

库存操作管理 实现商品入库功能，记录入库数量、入库日期、经手人等信息，并更新库存数量。实现商品出库功能，记录出库数量、出库日期、领用部门等信息，同时更新库存数量。对库存数量进行实时监控，当库存数量低于预设的安全库存时，系统发出预警提示。

库存查询与统计 支持按商品编号、名称、入库日期、出库日期等条件进行库存信息的查询。提供库存统计功能，能够统计某一时间段内的入库总量、出库总量、库存余额等信息。生成库存报表，如库存清单、出入库统计报表等，报表可导出为常见格式（如 Excel）。

第 5 章　栈和队列

栈和队列都是操作受限的线性结构，它们广泛应用于各种程序设计中。栈的操作特点是先进后出；队列的操作特点是先进先出。在编写程序的过程中，如果所处理数据满足上述特点，即可使用栈或队列。本章综合使用栈和队列，采用算符优先算法，示例如何设计一个可以计算四则混合算术表达式的计算器。通过本章案例的学习，读者深入了解栈和队列结构，能够在程序设计中应用栈以及队列解决实际的编程问题。

5.1 案例简介

5.1.1 任务描述

计算器是日常生活中经常使用到的计算工具，本章任务是设计一个带括号的算术表达式计算器。为了简单，假设表达式中只有小括号，并且能够进行“+”“-”“*”“/”基本运算。在满足基本运算的同时，要求计算器要有图形化界面。计算器设计完成后，运行效果如图 5-1 所示。用户输入的表达式显示在计算器界面顶部的空白显示区域，单击“=”按钮后计算结果会显示在计算器上。

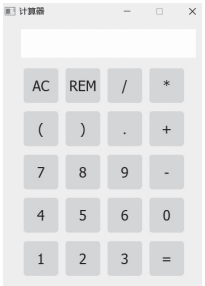

图 5-1　计算器主界面

5.1.2 实现思路

（1）在 Qt Creator 实现一个 Qwidget 对象窗口，窗口最上面布局一个 QLabel 对

象，用来显示表达式以及计算结果。主界面上布局 20 个 QPushButton 按钮，用来显示计算器的按键内容，设计效果如图 5–1 所示。

（2）设置一个 Qt 的 Queue 类的对象 queue，当单击计算器上的某个按钮时，按钮上的信息显示在 QLabel 对象中，同时将按钮上的字符入队列 queue 保存。

（3）单击“=”按钮，用算符优先算法计算队列 queue 中存储的算术表达式，并将计算的结果显示在计算机界面上的 QLable 对象中。

（4）在录入算术表达式的同时，检测表达式是否有语法错误，如果不符合录入规则，点击按钮，计算器不会给出任何回应。比如输入 8.3.3 这种情况，在录入到第二个小数点时，程序没有任何回应，录入不了。

（5）算术表达式计算采用算符优先算法，利用两个栈同时进行，也可以先将中缀表达式转换为后缀表达式，然后再计算。本案例采用两个栈同时进行的方法实现。

（6）计算器的输入是通过在主界面点击按钮实现的，不允许键盘直接输入。

5.2 知识要点

根据案例任务和实现思路，需要使用基于优先级的算符优先算法实现表达式求值，要用到栈结构。计算器程序输入的算术表达式是中缀表达式，在进行表达式的求值时，要从表达式的开头处理到结尾，因此可以先将录入的表达式存储到一个队列中。基于上述分析，本案例的实现要用到栈和队列两种数据结构，先将这两种结构以及需要用到的算符优先算法简单介绍一下。

5.2.1 栈知识解析

1. 栈的定义

栈是限定在表的一端进行插入和删除操作的线性结构，通常插入和删除的这端被称为栈顶，另一端称为栈底。当栈中没有数据元素时称为空栈。栈的插入操作通常称为进栈或入栈，栈的删除操作通常称为退栈或出栈。

2. 栈的基本运算

- 初始化栈 InitStack(s)　构造一个空栈 s。
- 销毁栈 ClearStack(s)　释放栈 s 被占用的存储空间。
- 求栈的长度 StackLength(s)　返回栈 s 中的元素个数。
- 判断栈是否为空 StackEmpty(s)　若栈 s 为空，则返回真，否则返回假。
- 进栈 Push(s,e)　入栈，将元素 e 插入到栈 s 中作为栈顶元素。
- 出栈 Pop(s,e)　出栈，将栈顶元素赋给 e。
- 取栈顶元素 GetTop(s,e)　返回当前的栈顶元素，并将其值赋给 e。

• 显示栈中元素 DispStack(s)　按从栈顶到栈底的顺序显示栈中所有元素。

在 Qt 框架里，内置了栈类 QStack，对标准模板库（STL）风格的栈数据结构予以实现。本案例直接用 QStack 栈来实现计算器，主要用入栈、出栈、取栈顶元素和判断栈是否为空几个栈的基本运算，以整型栈为例简单介绍如下，详细的知识请参考相关书籍。

若要使用 QStack 类，代码里需要包含相应的头文件：

#include <QStack>;

声明栈对象 stack：QStack<DataType> stack;

QStack()　构建一个空栈。

void push(const T &value)　把元素 value 压入栈顶。

T pop()　移除并返回栈顶元素。若栈为空，该行为未定义。

T &top()　返回对栈顶元素的引用。

const T &top() const　返回对栈顶元素的常量引用。

bool isEmpty() const　若栈为空则返回 true，反之则返回 false。

int size() const　返回栈中元素的数量。

3. 基于算符优先算法的表达式求值

计算器上录入的算术表达式是中缀表达式，由操作数、操作符和界限符组成。在本案例中，限定操作数为双精度浮点数，操作符为算术运算符“+”“–”“*”“/”，界限符为小括号，表达式结束符为“=”。可以先将中缀表达式转换为后缀表达式，然后再计算，也可以只用一个操作数和一个操作符栈，直接完成计算。本章采用后者完成算术表达式的计算。

本章把操作符和界限符统称为操作符，表达式计算基于算符优先算法，算法之间的优先级关系如表 5–1 所示。

表 5–1　算法之间的优先级关系

	M_2						
M_1	+	–	*	/	(	)	=
+	>	>	<	<	<	>	>
–	>	>	<	<	<	>	>
*	>	>	>	>	<	>	>
/	>	>	>	>	<	>	>
(	<	<	<	<	<	=	N
)	>	>	>	>	=	>	>
=	<	<	<	<	<	N	<

算法之间的优先级只能是大于、小于或者等于关系。具体算法描述如下：

• 初始化一个操作数栈 OPND，初始化一个操作符栈 OPTR，并入栈一个“=”表示求值开始。

• 依次读入表达式中的字符，如果是操作数，压入操作数栈 OPND；如果是操作符，则先用操作符栈的栈顶元素 M_1 和当前处理的字符 M_2 比较优先级大小。如果 M_1 的优先级小于 M_2，M_2 入操作符栈，接着处理表达式中的下一个字符；如果 M_1 的优先级大于 M_2，将操作数栈出栈两次，依次把栈顶元素保存在变量 a 和 b 中，再把操作符栈的栈顶元素出栈保存在变量 op 中，计算 b op a 的操作，并将结果压入操作数栈，然后继续处理当前字符 M_2。如果 M_1 的优先级和 M_2 的优先级相等，则是括号与括号相遇，只需要将操作符栈出栈一次，消去括号即可。

• 操作符栈中的栈顶元素是等号，当前处理字符 M_2 是等号的情况下，算术表达式计算结束，返回操作数栈的栈顶元素即为运算结果。

以 3*（3–2）–5/2 为例，利用两个栈，采用算符优先算法实现求值的过程。其中 Push() 是入栈函数，Pop() 表示出栈，OP() 表示一次计算，GetTop() 表示取栈顶元素。具体过程如表 5–2 所示，算法的程序实现见 5.3 节。

表 5–2　表达式 3*（3–2）–5/2 的计算过程

步骤	处理字符	操　作	操作符栈	操作数栈
1	3*（3–2）–5/2	Push(OPTR,′ =′)	=	
2	3*(3–2)–5/2=	Push(OPND,3)	=	3
3	*(3–2)–5/2=	Push(OPTR,′ *′)	=*	3
4	(3–2)–5/2=	Push(OPTR,′ (′)	=*(	3
5	3–2)–5/2=	Push(OPND,3)	=*(	3 3
6	–2)–5/2=	Push(OPTR,′ –′)	=*(–	3 3
7	2)–5/2=	Push(OPND,2)	=*(–	3 3 2
8	)–5/2=	Pop(OPND,a) Pop(OPND,b) Pop(OPTR,r) Push(OPND,OP(b op a))	=*(	3 1
9	)–5/2=	Pop(OPTR,op)	=*	3 1
10	–5/2=	Pop(OPND,a) Pop(OPND,b) Pop(OPTR,op) Push(OPND,OP(b op a))	=	3
11	–5/2=	Push(OPTR, –)	=–	3
12	5/2=	Push(OPND,5)	=–	3 5
13	/2=	Push(OPTR,′ /′)	=–/	3 5

续表

步骤	处理字符	操　作	操作符栈	操作数栈
14	2=	Push(OPND,2)	=–/	3 5 2
15	=	Pop(OPND,a) Pop(OPND,b) Pop(OPTR,op) Push(OPND,OP(b op a))	=–	3 2.5
16	=	Pop(OPND,a) Pop(OPND,b) Pop(OPTR,op) Push(OPND,OP(b op a))	=	0.5
17	=	Pop(OPTR,′ =′) return GetTop(OPND)		0.5

5.2.2 队列知识解析

存储算术表达式时，可以用 Qt 的 QString 类来实现，也可以用队列来实现，本章案例用队列实现。

1. 队列的基本概念

队列是一种运算受限的线性结构，限制仅允许在表的一端进行插入，而在表的另一端进行删除，它的操作特点是先进先出。插入的一段称为队头，删除的一段称为队尾，当队列中没有数据元素时称为空队列。

2. 队列的基本运算

InitQueue(&q)　初始化队列。

ClearQueue(&q)　销毁队列。

QueueEmpty(q)　判断队列是否为空。

QueueFull(q)　判断队列是否为满。

enQueue(q,e)　入队列。

deQueue(q,e)　出队列。

Qt 内置了 QQueue 模板类，用于实现队列数据结构。计算器的实现主要用声明队列对象、入队列、出队列、取队尾（队头）元素以及计算队列是否为空等操作，以整型队列为例简单介绍如下，详细的知识请参考相关书籍。

若要在代码中使用 QQueue 类，需要包含相应的头文件：#include <QQueue>；

声明队列对象 stack：QStack<DataType> queue；

QQueue()　创建一个空的队列。

void enqueue(const T &value)　将元素 value 添加到队列的尾部。

T dequeue()　移除并返回队列的头部元素。如果队列为空，该行为未定义。

T &head()　返回对队列头部元素的引用。

const T &head() const　返回对队列头部元素的常量引用。

T &tail()　返回对队列尾部元素的引用。

const T &tail() const　返回对队列尾部元素的常量引用。

bool isEmpty() const　若队列为空则返回 true，否则返回 false。

int size() const　返回队列中元素的数量。

5.3 实现代码

在 Qt Creator 中创建一个 widgets Application 工程，在其 ui 页面中设计的计算器界面如图 5-1 所示。

widget.cpp 文件中的包含文件如下：

```
#include "widget.h" // 自动生成的
#include "ui_widget.h" // 自动生成的
#include <QStack> // 包含 QStack 类
#include <QQueue> // 包含 QQueue 类
#include <QString> // 包含 QString 类
```

先将本章案例中所有可能出现的操作符用 change 函数数字化，即可在 priority 数组中读取出两个字符之间的优先级大小关系。各功能的实现代码如下。

5.3.1 优先级的实现

```
char priority[7][7]={// 顺序为 +-*/()=
        {'>','>','<','<','<','>','>'},
        {'>','>','<','<','<','>','>'},
        {'>','>','>','>','<','>','>'},
        {'>','>','>','>','<','>','>'},
        {'<','<','<','<','<','=',' '},
        {'>','>','>','>','N','>','>'},
        {'<','<','<','<','<','N','='}};
int Widget::change(char optr) // 给操作符编号
{
        switch(optr)
        {
        case '+': return 0;
        case '-': return 1;
        case '*': return 2;
        case '/': return 3;
        case '(': return 4;
        case ')': return 5;
```

```
        case '=': return 6;
        }
        return –1;
}
```

5.3.2 操作符数字化

```
int Widget::chJuage(char x) // 判别字符 x 是操作数还是操作符，是操作数返回 –1，
                                是操作符返回 0，其他的字符返回 1
{
        if(x>='0'&&x<='9')
                return –1;
        else if(x=='+'||x=='–'||x=='*'||x=='/')
                return 0;
        else
                return 1;
}
```

5.3.3 显示表达式

```
void Widget::display() // 在计算器上显示算术表达式
{
        QString axep="";
        for(QString value:queue) // 从头到尾读出队列中的对象连接成字符串对象
        {
                axep=axep+value;
        }
        ui–>label_exp–>setText(axep);
}
```

5.3.4 计算器上各个按钮的实现代码

1. AC 按钮

```
void Widget::on_btnAC_clicked()// 清除
{
        queue.clear();
        ui–>label_exp–>setText( "" );
}
```

2. REM 按钮

单击“REM”按钮，将队尾删除，并重新显示表达式。

```
void Widget::on_btnREM_clicked()// 清除表达式最后一位
{
    queue.removeOne(queue.last()); // 删除队列最后一个数据
    display();
}
```

3. “+”“–”“*”“/”按钮

因为这四个按钮的代码处理原理一样，所以只给出“+”号按钮的实现代码。

```
void Widget::on_btnAdd_clicked() // 加号按钮
{
    if(queue.size()==0) //+ 号不能是第一个表达式的第一个字符
        return;
    if(queue.back()==”.”)// 小数点后不能输入 + – * 或 /
        return;
    QString input=ui–>btnAdd–>text();
    if(queue.size()!=0)
    {
        QString aexp=queue.back(); // 队尾
        if(chJuage(aexp.back().toLatin1())==0)  // 队尾是 + – * 或 /
        {
            queue.removeOne(queue.last()); // 删除队列最后一个数据
        }
        queue.enqueue(input);// 入队列
        display();
    }
}
```

4. 小数点

录入小数点的时候，需要考虑表达式中小数点语法是否合法，如下情况小数点输入不进去：

表达式的第一个字符不能是小数点；

小数点前面一个字符必须是数字；

不能出现如 3.3.3 这样的情况。

```
void Widget::on_btnPoint_clicked() // 小数点
{
    if(queue.size()==0) // 第一个字符不能是小数点
```

```
        return;
    QString aexp=queue.back(); // 获取前面已经录入的最后一个字符
    if(chJuage(aexp.back().toLatin1())!=-1) // 最后一个不是数字，不能跟小数点
        return;
    if(queue.back()==".") // 小数点后不能是小数点
        return;
    QString input=ui->btnPoint->text();
    int i=queue.size()-1;
    while(i>=0)
    {

        if(chJuage(queue[i].back().toLatin1())==0)// 要输入小数点前面出现过的
                                                  操作符
        {
            queue.enqueue(input);
            display();
            return;
        }
        if(queue[i]==".") // 在想要输入小数点的前面只要出现过小数点
        {
            return;
        }
        else
        i--;
    }
    if(i==-1)
    {
        queue.enqueue(input);
        display();
    }
}
```

5. 数字 0~9 按钮

数字 0~9 按钮的处理方法一样，所以只给出数字 0 的代码。

```
void Widget::on_btn0_clicked()
{
    if(queue.size()!=0&&queue.back()==')') // 右括号后不能直接输入操作数
```

```
            return;
        QString input=ui->btn0->text();
        queue.enqueue(input);
        display();
}
```

6. 左括号

```
void Widget::on_btnleft_clicked()
{
        if(queue.size()!=0&&queue.back()==')')
            return;   // 右括号后面不能直接输入左括号
        QString input=ui->btnleft->text();
        queue.enqueue(input);
        display();
}
```

7. 右括号

```
void Widget::on_btnright_clicked()
{
        if(queue.size()==0) // 第一个字符不能是右括号
            return;
        QString axep=queue.back();
        if(chJuage(axep.back().toLatin1())==0) // 左括号后面不能直接输入右括号
            return;
        QString input=ui->btnright->text();
        queue.enqueue(input);
        display();
}
```

8. 等号

```
void Widget::on_btnEquals_clicked()  // 等号
{
        if(queue.size()==0) // 表达式为空
            return;
        QString input=ui->btnEquals->text(); // 获取等号
            queue.enqueue(input);// 等号入队列
            double result=count();// 调用计算表达式的函数
        if(result==INT_MAX)
            ui->label_exp->setText(“表达式有错！”);
```

```
        else
        {

            ui->label_exp->setText(QString::number(result));
// 在主界面的 QLabel 标签中显示计算结果
        }
}
```

5.3.5 表达式求值计算

```
        QStack<char> charStack;// 字符栈
        charStack.push('=');// 运算符栈中先放入一个 =
        QStack<double> digStack;// 数据栈
        double op1,op2;// 存放取出用于计算的浮点数
        char chtop;// 当前字符
        QString tempdata;
        while(!queue.isEmpty())
        {
            QString aexp=queue.head();// 取队头信息
             if((aexp.back().toLatin1()>='0'&&aexp.back().toLatin1()<='9')||aexp.back().
toLatin1()=='.')
            {
                tempdata.push_back(aexp.back());
                // 是数字或者是小数点，先放入 tempdata 字符串对象
                queue.dequeue();
            }
            else if(aexp.back()=='='&&charStack.top()=='=') // 等号与等号相遇
            {
                if(digStack.size()!=0) // 正常计算结束，返回计算结果
                    return digStack.pop();
                else
                    return INT_MAX; // 异常结束，返回无穷大
            }
            else // 操作符
            {
                if(tempdata.size()!=0)
                 // 将 tempdata 对象中的字符串转为浮点数并入操作数栈
                {
                digStack.push(tempdata.toDouble());
                tempdata.clear();
                }
                chtop=charStack.top();// 操作符栈的栈顶操作符
                char proc=priority[change(chtop)][change(aexp.back().toLatin1())];
```

```
                // 比较栈顶字符与当前处理字符的优先级
                    switch (proc)
                    {
                    case '<':// 如果 xt 的优先级小于 x 的优先级
                        charStack.push(aexp.back().toLatin1());//ch 入运算符栈
                        chtop=aexp.back().toLatin1();// 更新栈顶字符
                        queue.dequeue();// 出队列
                    break;
                case '>':
                    {
                    // 例如需要计算 4+= 这种错误表达式
                    if(digStack.size()==1)
                        return INT_MAX;
                    char op=charStack.pop();
                    op1=digStack.pop();// 获取第一个操作数
                    op2=digStack.pop();// 获取第二个操作数
                    switch(op) // 分别计算
                    {
                        case '+':digStack.push(op2+op1);break;
                        case '-':digStack.push(op2-op1);break;
                        case '*':digStack.push(op2*op1);break;
                        case '/':
                        if(op1==0)
                            return INT_MAX;
                        else
                            digStack.push(op2/op1);
                    }
                        queue.enqueue(aexp.back());
                    break;
                    }
                case '=':// 如果 x1 的优先级等于 x2 的优先级
                    charStack.pop();
                    queue.dequeue();// 出队列
                    break;
                default:// 其他情况是转换失败而结束
                    return INT_MAX;
                    break;
                }
            }
        }
        return digStack.top();
    }
```

5.4 运行结果

运行计算器，输入多组测试数据，运行结果如表 5–3 所示。

表 5–3　计算器运行结果

已输入	当前输入	输　出
无输入	.、+、–、*、/、)	无输出
4+	–	4–
86.7/(7–7)	=	表达式错误!
()	=	表达式错误!
(4+4.5))+6/2	=	表达式错误!
((4+4.5)+6/2	=	表达式错误!
5.5*(8–6.9)–9.6/8+9.9	=	14.75
3*(3–2)–5/2	=	0.5
88+9.8*(85.3–6)	=	865.14

练习

1. 音乐播放器

（1）任务简介

音乐播放器作为人们日常生活中常用的多媒体应用之一，不仅需要具备播放音乐的基本功能，还需要良好的用户交互界面来提升用户体验。本题目要求学生开发一个带有图形化界面的音乐播放器，旨在让学生综合运用数据结构与 C++ 编程知识解决实际问题，同时深入理解如何通过合理的数据组织和算法设计来实现高效的音乐播放和管理功能。

（2）设计任务

音乐库管理模块　支持音乐文件的添加、删除、修改功能。能够从本地文件夹中导入音乐文件到音乐库，并在 UI 界面上以列表形式展示音乐库中的音乐信息，包括显示歌曲缩略图（若文件包含）、歌曲名称、歌手等信息。

播放控制模块　具备基本的播放、暂停、停止、上一曲、下一曲播放控制功能，可通过 UI 界面上的按钮或键盘快捷键进行操作。

搜索功能模块　提供搜索功能，用户可以根据歌曲名称、歌手名称等信息在音乐库中进行搜索，搜索结果能够快速准确地在 UI 界面上呈现，并定位到对应的歌

曲条目，方便用户直接播放或进行其他操作。

歌词显示模块（选作） 对于带有歌词的音乐文件（常见的如 MP3 格式且内嵌歌词的文件），能够在播放时同步显示歌词。歌词显示的位置与节奏要与音乐播放进度相匹配，并且支持用户对歌词的字体大小、颜色、显示位置等样式进行简单的设置，以提升歌词的可读性。

2. 贪吃蛇游戏

（1）任务简介

贪吃蛇游戏是一款经典的休闲益智游戏，以简单的规则和趣味性深受玩家喜爱。在游戏中，玩家通过控制一条不断移动的贪吃蛇，使其在场景中寻找食物，让自己不断长大，同时要避免撞到场景中的墙壁或自身身体，否则游戏结束。本设计要求学生运用所学的数据结构知识来构建贪吃蛇游戏的底层逻辑，包括游戏的初始化、画面更新、贪吃蛇的移动控制、食物生成与碰撞检测等，从而深入理解数据结构在游戏开发中的应用，提高编程能力和算法设计能力，同时也能锻炼学生的逻辑思维和问题解决能力。

（2）设计任务

游戏场景绘制 设计一个二维的游戏场景，可以是自定义的图形界面，用于展示贪吃蛇、食物和墙壁等元素，自定义场景的大小和边界。

游戏控制 控制蛇在游戏场景中移动，吃到食物后的蛇身体长大（或者自定义规则），碰到墙壁游戏结束。

3. 火车票订票系统

（1）任务简介

火车票订票系统是一个成熟的商业软件。本题要求学生综合运用所学的数据结构知识，构建一个模拟的火车票订票系统，涉及车站信息管理、车次信息管理、车票预订与查询、余票管理等多个模块，以深入理解数据结构在实际问题中的应用，培养编程能力、算法设计、分析能力，以及解决复杂实际问题的能力，同时提升学生的团队协作和项目管理能力。

（2）设计任务

车票预订 设计一个车票预订模块，当用户输入出发站、目的站、出行日期等信息后，系统能够根据已有的车次信息和余票情况，为用户推荐合适的车次。如果用户选择了某个车次，系统需要检查该车次的余票数量，若有余票，则为用户分配座位（座位可以用数字或字母加数字等形式表示），并更新该车次的余票信息，同时记录用户的购票信息（包括用户名、身份证号、购票时间等）。

车票查询 实现车票查询功能，用户可以根据购票凭证（如订单号或身份证号）查询自己已购买的车票信息，包括车次、座位号、出发时间和到达时间等。

退票处理 当用户因某种原因需要退票时，系统能够根据用户提供的购票凭证（如订单号或身份证号）找到相应的车票信息。