

《(空间)数据结构》

实验报告

课程代码(FGEE004512)

姓 名：于沛豪

学 号：2023113911

指导教师：杨骏

目 录

第一部分 实验分析	1
1.1 实验目的	1
第二部分 实验流程	1
2.1 实验内容	1
2.2 可视化处理	2
2.3 Widget.h 头文件编写	3
2.4 Widget.cpp 源文件编写	4
第三部分 实验总结	6
3.1 实验成果	6
3.1.1 顺序表操作	6
3.1.2 链表操作	7
3.2 实验简述	9
3.2.1 关于可视化	9
3.3 实验小结	9
参考文献:	9

第一部分 实验分析

1.1 实验目的

- (1) 通过实验过程了解并掌握线性表的顺序存储结构的定义及顺序表中的各种基本操作。
- (2) 通过实验过程了解并掌握链表的顺序存储结构的定义及顺序表中的各种基本操作。
- (3) 认识线性表并且会利用线性表的两种存储结构解决简单问题。

1.1 操作环境

本实验及该学期的剩余实验及学习均使用开源软件体系完成

- 编译环境：MiniGW QT 6.8.3
- 可视化窗口展示：QT Creator
- 本次实验工程已上传至 Github 供参考：[B1AnKAlpha/SpaceDataStructure](https://github.com/B1AnKAlpha/SpaceDataStructure)



第二部分 实验流程

2.1 实验内容

- ✧ 完成对顺序表的删除，插入，查找，输出等操作。
- ✧ 完成对链表的插入，查找，返回等操作。
- ✧ 认识线性表并且会利用线性表的两种存储结构解决简单问题。
- ✧ 对线性表中的代码进行学习与掌握。

2.2 可视化处理

本次实验采用 QtCreator 创建 Qmake 项目，通过槽函数以及信号来实现按钮单击事件读写数据（类似于 C#语言中的事件与委托）。

窗体布局设计如下所示：



图 1 窗体布局参考

本次实验选择 Qtcreator 的 Widget 项目进行开发，其优点是适合该种小型项目，并且不宜在代码编译时窗口发生意料之外的形变。

通过窗口创建的对象如下所示：

对象	类
Widget	QWidget
tabWidget	QTabWidget
tab	QWidget
label	QLabel
label_10	QLabel
label_2	QLabel
label_3	QLabel
label_4	QLabel
label_5	QLabel
label_6	QLabel
label_7	QLabel
label_8	QLabel
label_9	QLabel
lineEdit	QLineEdit
lineEdit_2	QLineEdit
lineEdit_3	QLineEdit
lineEdit_4	QLineEdit
lineEdit_5	QLineEdit
lineEdit_6	QLineEdit
lineEdit_7	QLineEdit
lineEdit_8	QLineEdit
pushButton	QPushButton
pushButton_2	QPushButton
pushButton_3	QPushButton
pushButton_4	QPushButton
tab_2	QWidget

图 2 窗体对象与类

2.3 Widget.h 头文件编写

关于链表、线性表等基本定义（Typedef）经测试无法通过窗体加载时的析构函数来定义，故选择在头文件定义，以链表为例，头文件如下所示：

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE
typedef double ElemType;
typedef struct LNode {
    ElemType data;
    struct LNode *next;
} LNode; // 结点结构

class Widget : public QWidget
{
    Q_OBJECT
public:
    Widget(QWidget *parent = nullptr);
    ~Widget()
private slots:
    void on_pushButton_clicked(); // 建立链表
    void on_pushButton_2_clicked(); // 插入元素
    void on_pushButton_3_clicked(); // 删除元素
    void on_pushButton_4_clicked(); // 查找元素
private:
    Ui::Widget *ui;
    LNode *L; // 链表头指针
    LNode* creat_L(); // 初始化链表
    void insert_L(int i, ElemType e); // 插入元素
    ElemType delete_L(int i); // 删除元素
    int locat_L(ElemType e); // 查找元素
    void out_L(); // 输出链表
};
#endif // WIDGET_H
```

注：为支持读取普通测量实验所需数据，Typedef Datatypy 处改为 double。在头文件中将链表作为成员来申明，使代码更加模块化，具有规范性。

2.4 Widget.cpp 源文件编写

使用 C++ 来完成窗体应用这一强面向对象式编程相比于 C# 等语言着实难度不小。在窗体创建时的构造函数，我进行了初始化链表的操作：

```
Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
{
    ui->setupUi(this);
    L = creat_L(); // 初始化链表
}
```

同理在结束窗体的析构函数时，我进行了删除链表的操作：

```
Widget::~~Widget()
{
    delete ui;
    // 释放链表内存
    LNode *p = L;
    while (p != nullptr) {
        LNode *temp = p;
        p = p->next;
        free(temp);
    }
}
```

其大部分代码与示范代码类似，但有些需要改变，例如读取数据时，采用空格区分数据并加入到链表中，其中区分数据需要利用 Qt 自带的函数

```
// 获取链表元素个数
QString countStr = ui->lineEdit->text();
int count = countStr.toInt();

// 获取输入的元素列表
QString elementsStr = ui->lineEdit_2->text();
QStringList elementsList = elementsStr.split(" ", Qt::SkipEmptyParts);
ui->lineEdit_3->setText(elementsList[0]);
```

注：QString 为 Qt 自带的数据格式，若需进行计算或处理需转换为 String 或 double 等类型。

也可利用 `iostream`，即流（<<或>>）的特点来读取空格为分割的数据。
如下所示：

```
QString a = ui->lineEdit->text();
QString n = ui->lineEdit_2->text();

double p[MAX] = {};
stringstream ss(n.toStdString());
std::string t;
std::vector<std::string> res;

while (ss >> t) {
    res.push_back(t);
}

int m = 0;
for (auto s : res) {
    p[m] = stod(s);
    m++;
}

for (int i = 0; i < a.toInt(); i++) {
    L->elem[i] = p[i];
    L->Last++;
}
```

其余大部分代码相同，总实验一完整项目见 [B1AnKAlpha/SpaceDataStructure](https://github.com/B1AnKAlpha/SpaceDataStructure)

第三部分 实验总结

3.1 实验成果

3.1.1 顺序表操作

- 顺序表的建立



- 顺序表的删除



- 顺序表的查询

数据结构实验一

顺序表的基本操作 链表的基本操作

请输入你要创建的顺序表元素个数 4

请输入元素(以空格分界) 9.395 499.419 500 创建

输出 499.545 499.395 499.419 500

请输入要删除的元素 499.419 请输入要查询的元素 499.395 请输入要插入的元素 位置 插入值

删除 查询 插入

输出 1

2023113911 于沛豪 使用Qt Creator开发

- 顺序表的插入

数据结构实验一

顺序表的基本操作 链表的基本操作

请输入你要创建的顺序表元素个数 4

请输入元素(以空格分界) 9.395 499.419 500 创建

输出 499.545 499.395 499.419 500

请输入要删除的元素 499.419 请输入要查询的元素 499.395 请输入要插入的元素 位置 1 插入值 111.11

删除 查询 插入

输出 499.545 111.11 499.395 500

2023113911 于沛豪 使用Qt Creator开发

3. 1. 2 链表操作

- 链表的建立

空间数据结构实验一

链表的基本操作 顺序表的基本操作

请输入你要创建的链表元素个数

请输入元素(以空格分界)

输出

请输入要删除的元素

请输入要查询的元素

请输入要插入的元素

位置 插入值

输出

2023113911 于沛豪 使用Qt Creator开发

- 链表的删除

空间数据结构实验一

链表的基本操作 顺序表的基本操作

请输入你要创建的链表元素个数

请输入元素(以空格分界)

输出

请输入要删除的元素

请输入要查询的元素

请输入要插入的元素

位置 插入值

输出

2023113911 于沛豪 使用Qt Creator开发

- 链表的插入

空间数据结构实验一

链表的基本操作 顺序表的基本操作

请输入你要创建的链表元素个数

请输入元素(以空格分界)

输出

请输入要删除的元素

请输入要查询的元素

请输入要插入的元素

位置 插入值

输出

2023113911 于沛豪 使用Qt Creator开发

3.2 实验简述

3.2.1 关于可视化

本次实验采用 QtCreator 创建 Qmake 项目，通过槽函数以及信号来实现按钮单击事件读写数据（类似于 C#语言中的事件与委托）。选择 Qtcreator 的 Widget 项目进行开发，其优点是适合该种小型项目，并且不宜在代码编译时窗口发生意料之外的形变。

关于链表、线性表等基本定义（Typedef）经测试无法通过窗体加载时的析构函数来定义，故选择在头文件定义，在窗体创建时的构造函数，我进行了初始化链表的操作，同理在结束窗体的析构函数时，我进行了删除链表的操作。为支持读取普通测量实验所需数据，Typedef Datatypy 处改为 double。在头文件中将链表作为成员来申明，使代码更加模块化，具有规范性。

3.3 实验小结

- 在本次实验中，我深入了解了 C 语言可视化的整个操作流程，并利用 Git 进行版本管理，了解了目前现代化的编程流程
- 本次实验我完成对顺序表的删除，插入，查找，输出等操作。完成对链表的插入，查找，返回等操作。认识了线性表并且会利用线性表的两种存储结构解决简单问题。
- 在课外，也自学了 C 语言利用 QtCreator 进行可视化的基本流程。这不仅对我课内的一些实验或作业有帮助，也为我今后编程打下了基础。我也加深了对 C 语言面向对象编程的基本流程的理解。

参考文献：

- [1]. 王远飞, 何洪林. 空间数据分析方法[M]. 科学出版社, 2007.
- [2]. 陈雪梅, 韩洁琼. C 语言可视化编程环境的设计与实现[J]. 武汉理工大学学报: 信息与管理工程版, 2010, 32(4): 561-564.
- [3] 严蔚敏, 吴伟民. 数据结构: C 语言版[M]. 清华大学出版社有限公司, 1997.

附录 设计实验

约瑟夫环算法实现：

使用 Node 结构实现单向循环链表。构建 n 个人的循环链表。从 s 号开始，每次报数 m 个人后，删除当前节点。

```
#include <iostream>
using namespace std;

// 单向循环链表节点
struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

// 约瑟夫环（链表实现）
void josephus_linkedlist(int n, int m, int s) {
    Node *head = new Node(1), *prev = head;
    for (int i = 2; i <= n; ++i) { // 创建循环链表
        prev->next = new Node(i);
        prev = prev->next;
    }
    prev->next = head; // 尾节点指向头，形成循环链表

    // 找到起始节点
    Node *cur = head, *pre = prev;
    for (int i = 1; i < s; ++i) {
        pre = cur;
        cur = cur->next;
    }

    // 开始报数
    while (cur->next != cur) { // 直到只剩最后一个人
        for (int i = 1; i < m; ++i) { // 报数 m-1 次
            pre = cur;
            cur = cur->next;
        }

        cout << cur->data << " "; // 输出出局者
        pre->next = cur->next; // 删除当前节点
        delete cur; // 释放内存
        cur = pre->next; // 移动到下一个人
    }

    cout << cur->data << endl; // 输出最后剩下的那个人
}
```

```
    delete cur; // 释放内存
}

int main() {
    int n, m, s;
    cout << "输入总人数 n, 报数值 m, 起始位置 s: ";
    cin >> n >> m >> s;
    josephus_linkedlist(n, m, s);
    return 0;
}
```