

计算机学院高级语言程序设计课程实验报告

实验题目：		学号：202300130183
日期：2024 年 3 月 4 日	班级：2023 级智能班	姓名：宋浩宇
Email：202300130183@mail.sdu.edu.cn		
<p>实验目的：</p> <ol style="list-style-type: none">1. 了解结点类的声明与实现，学习其使用方法。2. 了解链表类的声明与实现，学习其使用方法。3. 了解栈类的声明与实现，学习其使用方法。4. 了解并练习模板元编程。5. 掌握 Linux 系统下 C++环境的搭建和使用		
<p>实验软件和硬件环境：</p> <p>实验软件：Windows 11 家庭中文版(x64) Visual Studio 2022</p> <p>硬件环境：处理器：13th Gen Intel(R) Core(TM) i9-13980HX 2.20 GHz</p> <p>RAM 32.0 GB (31.6 GB 可用)</p>		
<p>实验步骤与内容：</p> <p>第 9 章 1. (1)</p>		

```
Microsoft Visual Studio × + - □ ×
在链表末端增加了1
在链表末端增加了2
在链表末端增加了3
0
1
2
3
0
2
3
2在链表中
1不在链表中
0
2
3
已将2改为1
0
1
3
2不在链表中
0
1
3

F:\Homework\高级语言程序设计作业\实验9\code\x
64\Debug\code.exe (进程 20592)已退出，代码为
0。
按任意键关闭此窗口 . . .|
```

链表的基本操作分为增删查改和输出，增加的实现方式为从头节点开始向后遍历，当链表的下一节点地址为 NULL 的时候将其设为新节点的地址；删除的实现为从头节点开始遍历，当节点的下一节点的数据为要删除的的值的时候，将该节点的下一节点设为原下一节点的下一节点，因为在构建链表的时候使用的是 new 申请来的内存，此处也可以添加 delete 操作；查找操作即为从头节点开始向后遍历，当节点的数据与要查找的数据相等时返回 true，当节点的下一节点的地址为 NULL 的时候返回 false；修改的实现为从头节点开始向后遍历，当节点的数据与要修改的值相等时，将其设置为要修改的值，结束函数的执行；输出的实现为，从头节点开始向后遍历，输出每个节点的值直到节点的下一节点的地址为 NULL 进行最后一次输出。

第9章 1. (2) 首先修正原 link.h 头文件中的几处 bug

```

//删除当前节点
template <class T> <T> 提供 IntelliSense 的示例模板参数
void LinkedList<T>::deleteCurrent() {
    Node<T>* temp = currPtr;
    prevPtr = currPtr->next; //bug1
    delete temp;
    size--;
}

```

删除节点的这个函数错误地将该注释行表达式左侧的对象写成了 currPtr

```

//在表头插入节点, 原头节点front之前插入, 插入之后原
template <class T> <T> 提供 IntelliSense 的示例模板参
void LinkedList<T>::insertFront(const T& item) {
    if (size == 0)
    {
        Node<T>* n = new Node<T>(item, currPtr);
        front = n; //front指向新的节点
        rear = n; //bug2
        size++;
        position++;
    }
    else
    {
        Node<T>* n = new Node<T>(item, currPtr);
        front = n; //front指向新的节点
        rear = n;
        size++;
        position++;
    }
}

```

```

//在链表尾节点之前插入节点
template <class T>
void LinkedList<T>::insertRear(const T& item) {
    if (size == 0)
    {
        Node<T>* temp = new Node<T>(item, NULL);
        front = temp;
        rear = temp; //bug3
        currPtr = temp;
        size++;
        return;
    }
    Node<T>* temp = front;
    while (temp->next != rear)
        temp = temp->next; //temp移至rear之前
    Node<T> *n = new Node<T>(item, rear);
    temp->next = n;
    size ++;
}

```

```

//在当前节点之前插入节点
template <class T>
void LinkedList<T>::insertAt(const T& item) {
    Node<T>* temp = new Node<T>(item, currPtr);
    prevPtr->next = temp;
    prevPtr = temp;
    if (size == 0)
    {
        front = temp; //bug4
        rear = temp;
    }
    size ++;
    position ++;
}

```

```

//在当前节点之后插入节点
template <class T> <T> 提供 IntelliSense 的示例模板参数
void LinkedList<T>::insertAfter(const T& item) {
    if (size == 0)
    {
        Node<T>* temp = new Node<T>(item, NULL);
        front = temp;
        rear = temp;
        currPtr = temp; //bug5
        size++;
        position = 1;
        return;
    }
    if (position == size)
    {
        Node<T>* temp = new Node<T>(item, NULL);
        temp->next = currPtr->next;
        currPtr->next = temp;
        rear = temp;
        size++;
        return;
    }
    Node<T>* temp = new Node<T>(item, NULL);
    temp->next = currPtr->next;
    currPtr->next = temp;
    size ++;
}

```

以上几个 bug 的共性是没有考虑链表为空时节点的创建，其中在 bug5 中，原本的函数也没有考虑在最后一个节点后边添加节点时需要刷新尾节点的问题。

补充定义了所需的函数：

```

template<class T>
void LinkedList<T>::addAfter(LinkedList<T>& list)
{
    int cnt = 0;
    Node<T>* tem = list.front;
    while (cnt<list.size)
    {
        currPtr = rear;
        position=size;
        insertAfter(tem->getData());
        tem = tem->next;
        cnt++;
    }
    currPtr = prevPtr->next;
}

```

用于链接 AB 两个链表

```
template<class T>
void LinkedList<T>::display()
{
    if (size == 0)
    {
        cout << "链表为空";
        return;
    }
    Node<T>* tem = front;
    int cnt = 1;
    while (cnt<=size)
    {
        cout << tem->getData() << endl;
        tem = tem->next;
        cnt++;
    }
}
```

用于输出链表里的值

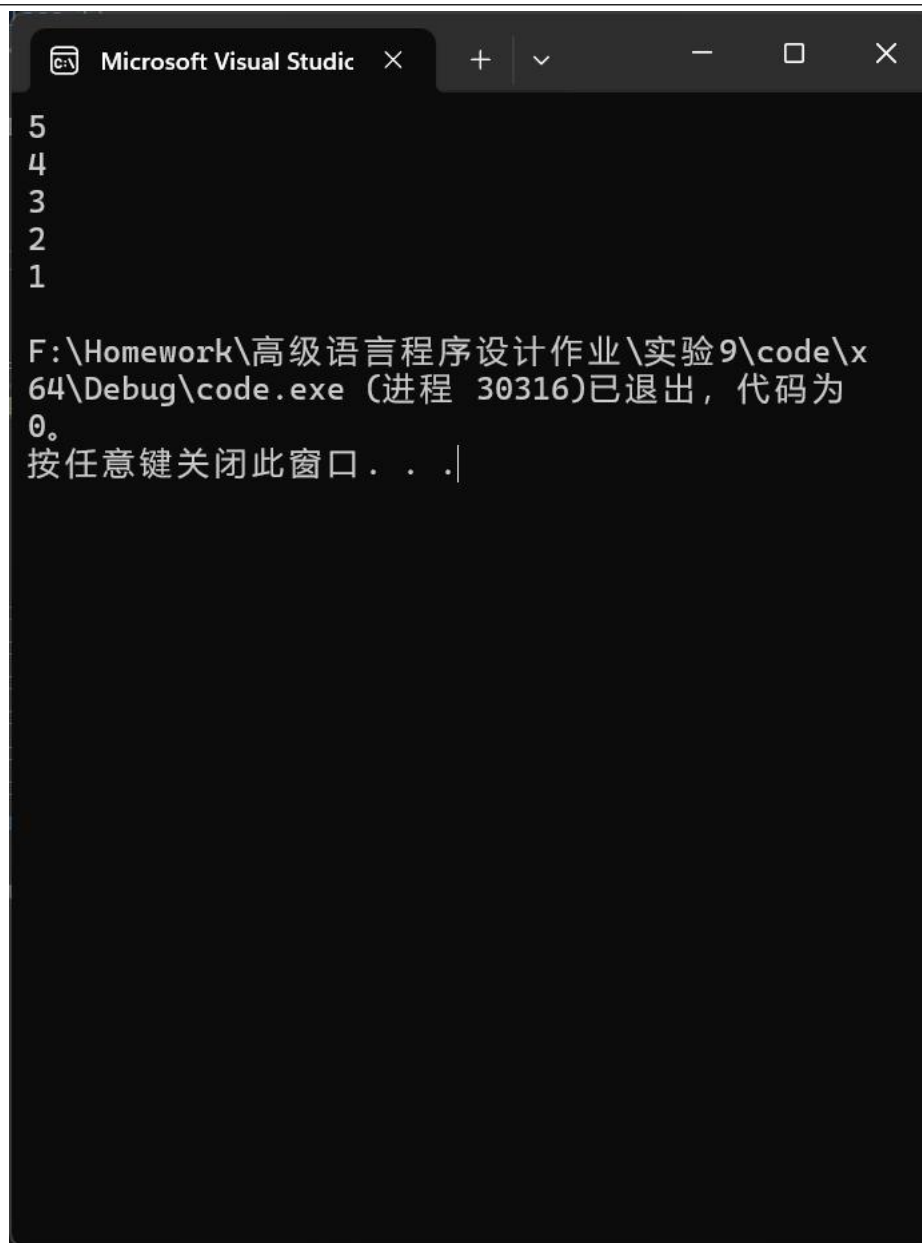
因为考虑到编译器会自动对两个链表对象 AB 的内存进行释放，而原本的代码没有使用智能指针，为了防止内存释放出现错误，链接两个链表实际上是把一个链表内的数据依次加到另一个链表的末端。

运行结果如下：

```
Microsoft Visual Studio × + - □ ×
A链表
1
5
4
3
2
B链表
6
10
9
8
7
AB连接后
1
5
4
3
2
6
10
9
8
7

F:\Homework\高级语言程序设计作业\实验9\code\x
64\Debug\code.exe (进程 39336)已退出, 代码为
0。
按任意键关闭此窗口...
```

第9章 1. (3) 输出结果如下:

A screenshot of the Microsoft Visual Studio IDE. The top toolbar shows icons for adding, removing, and zooming tabs. The main editor area has a dark background with white text. On the left side of the editor, the numbers 5, 4, 3, 2, and 1 are listed vertically, representing a stack. Below these numbers, a message is displayed: 'F:\Homework\高级语言程序设计作业\实验9\code\x64\Debug\code.exe (进程 30316)已退出, 代码为 0。' followed by '按任意键关闭此窗口. . .|'.

```
5
4
3
2
1

F:\Homework\高级语言程序设计作业\实验9\code\x64\Debug\code.exe (进程 30316)已退出, 代码为 0。
按任意键关闭此窗口. . .|
```

在编写 queue 类使用了 c++ 中的 vector 容器, 使用 vector 的 `push_back()` 和 `pop_back()` 方法可以很容易的实现栈的功能, 如图所示:


```

template<class T>
class Queue
{
private:
    vector<T> datas;
    int size = 0;
public:
    void insert(T data);
    T get();
};

template<class T>
void Queue<T>::insert(T data)
{
    datas.push_back(data);
    size++;
}

template<class T>
T Queue<T>::get()
{
    T tem = datas[--size];
    datas.pop_back();
    return tem;
}

```

第 9 章（4）输出结果为

```
Microsoft Visual Studio × + ▾ − □ ×
0 1 2 3 4 5 6 7 8 9
9 8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
8
F:\Homework\高级语言程序设计作业\实验9\code\x
64\Debug\code.exe (进程 38004)已退出, 代码为
0。
按任意键关闭此窗口. . .|
```

此处三个数组原数组均为 0123456789，分别使用三种不同的排序方式来排序

```

int main()
{
    Array<int> a(10);
    Array<int> b(10);
    Array<int> c(10);

    for (int i = 0; i < 10; i++)
    {
        a[i] = i;
        b[i] = i;
        c[i] = i;
    }

    a.show();
    cout << endl;
    a.BubbleSort();
    a.show();
    cout << endl;
    b.insertSort();
    b.show();
    cout << endl;
    c.selectSort();
    c.show();
    cout << endl;
    cout << c.seqSearch(1) << endl;
    return 0;
}

```

最后一个输出则是 c 数组中 1 的下标。
 分别解释三个算法（以从大到小为例）：

```

template<class T>
void Array<T>::insertSort() //直接插入排序(从大到小)
{
    for (int i = 0; i < size; i++)
    {
        T tem = list[i];
        for (int j = i - 1; j >= 0; j--)
        {
            if (list[j] < tem) {
                list[j + 1] = list[j];
            }
            if (list[j] >= tem) {
                list[j + 1] = tem;
                break;
            }
            if (j == 0)
            {
                list[j] = tem;
            }
        }
    }
}

```

直接插入排序，过程如下：将前 n 个数字视为一个数组，考虑第 $n+1$ 个数字，先用 `tem` 存储下这个数字，然后向前依次比较，如果比它小则将对应该数字向后移动一位，如果大于等于它则将 `tem` 赋给这个数字的下一位， n 的取值范围是 1 到 `size`；这样就可以在保证每次计算时前 n 个数字组成的数组的有序性。

```

template<class T>
void Array<T>::selectSort() //选择排序(从大到小)
{
    for (int i = 0; i < size; i++)
    {
        T max = list[i];
        T tem = list[i];
        T mark = i;
        for (int j = i; j < size; j++)
        {
            if (list[j] > max)
            {
                max = list[j];
                mark = j;
            }
        }
        list[i] = max;
        list[mark] = tem;
    }
}

```

选择排序，过程如下：将前 n 个数字视为一个数组，考虑第 n 位数字，从第 $n+1$ 为向后遍历，

找出最大的数字和第 n 个数字比较，比他大就交换位置，比他小就不做处理， n 的取值为 1 到 $size$ ，这样可以保证每次挑出来的数字都是当前可选数字中的最大值。

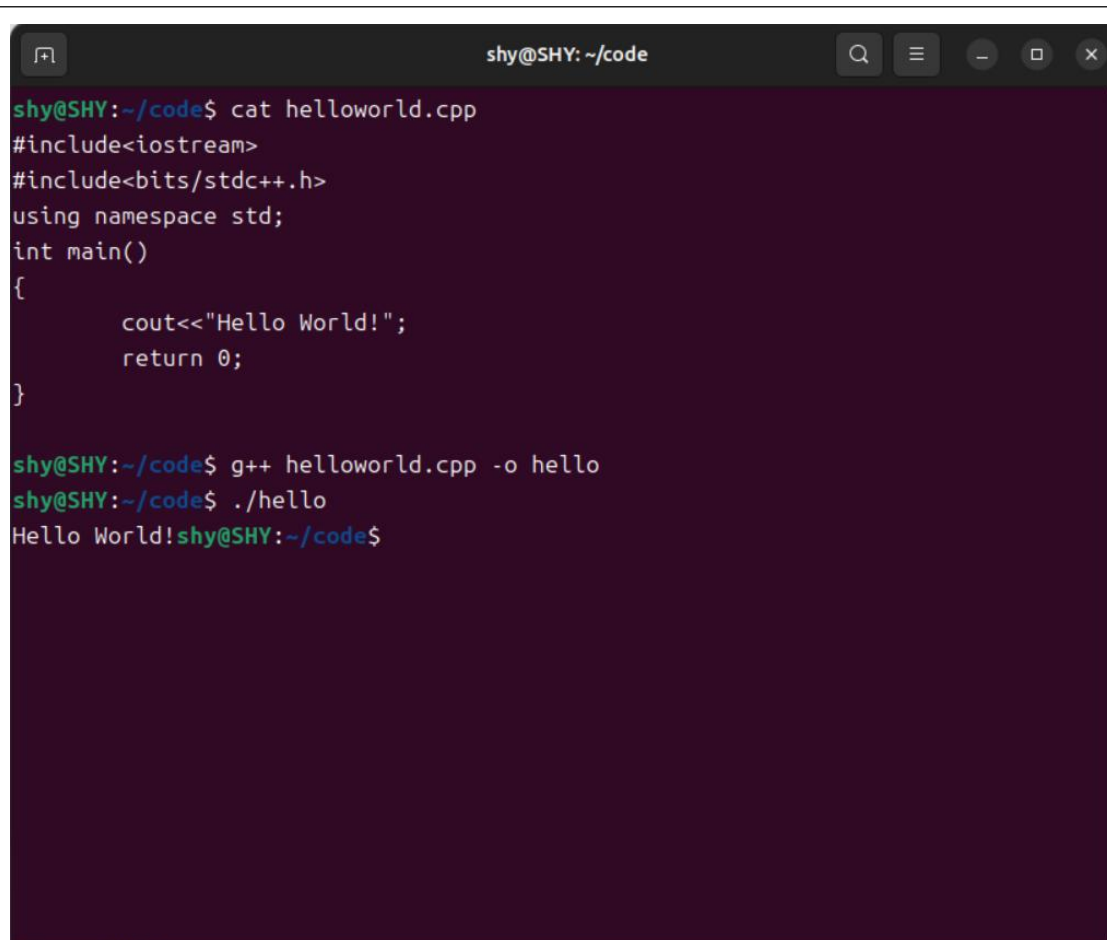
```
template<class T>
void Array<T>::BubbleSort() //冒泡排序(从大到小)
{
    for (int i = 0; i < size; i++)
    {
        T tem;
        for (int j = 0; j < size - 1 - i; j++)
        {
            if (list[j] < list[j + 1])
            {
                tem = list[j + 1];
                list[j + 1] = list[j];
                list[j] = tem;
            }
        }
    }
}
```

冒泡排序，过程如下：从数组的第一位开始向后遍历，如果第 i 位的数字小于第 $i+1$ 位的数字，则交换两个数字的位置，这样可以使每次遍历数组都挑选出一个最小值放在数组末尾，遍历 n 次就选出了 n 个前 $(size-n)$ 个最小值，以此类推执行 $size$ 次后数组有序。

第 9 章 4. 模板元编程是建立在 C++ 模板之上的一种高级技术，它利用模板的通用性和编译时计算的能力，实现在编译时生成代码逻辑和执行计算。模板元编程是一种强大的编程技术，可以在一定程度上提高程序的性能和可维护性，但也需要深入理解 C++ 模板和编译器的工作原理。

第 9 章 5. 利用了类模板的主要知识，其中还牵扯了大量的前几章的知识和类模板知识混合在一起。

Linux 下 C++ 环境配置：
运行 helloworld 程序

A terminal window with a dark background and light-colored text. The window title is 'shy@SHY: ~/code'. The terminal shows the following sequence of commands and output:
1. Command: `cat helloworld.cpp`
Output: `#include<iostream>
#include<bits/stdc++.h>
using namespace std;
int main()
{
 cout<<"Hello World!";
 return 0;
}`
2. Command: `g++ helloworld.cpp -o hello`
3. Command: `./hello`
Output: `Hello World!`
The prompt `shy@SHY:~/code$` is visible at the end of the output line.

运行以前实验的代码：

