基础实验：实现文件读写与向量

姓名：黎行健　　学号：2024201585

2025 年 10 月 22 日

1 问题分析

文件的读写要用到文件 IO 流，向量的功能实现则可以利用普通顺序表的动态分配内存以及基础操作完成

2 数据结构设计与实现

实现 Myarray 类，其对象即为向量

2.1 ADT 设计

- 数据对象：Myarray 类，内含 size（向量长度）arr（数组，向量元素存储空间），数组 arr 内每个元素都是 int 类型

```
class Myarray{
private:
    int size;
    int* arr;
};
```

- 数据关系：arr 是顺序的，每个元素除了头元素和尾元素都有唯一的前驱和后继元素，头元素有唯一的后继元素但没有前驱元素，尾元素有唯一的前驱元素但没有后继元素
- 基本操作：

```
setsize(x): 设置size
getsize(): 获取size
initial_arr(p): 初始化向量
show(): 打印向量
push_back(x): 在向量末尾添加元素
pop_back(): 在向量末尾弹出元素
insert(pos,x): 在向量pos位处插入元素
erase(pos): 删除pos位置的元素
update(pos,x): 修改pos位置的元素
save(filename,length): 保存向量数据到指定文件里面
```

2.2 ADT 的物理实现

```
class Myarray{
public:
    Myarray();
    void setsize(int size);
    void getsize();
    void initial_arr(int* p);
    void show();
    void push_back(int x);
    void pop_back();
    void insert(int pos,int x);
    void erase(int pos);
    void update(int pos,int x);
    void save(char* filename,int length);
    ~Myarray();
private:
    int size;
    int* arr;
};
```

3 算法设计与实现

3.1 算法设计及实现

整个流程，在选择文件输入之后，我选择了循环结构，支持用户不断进行对向量的操作，直到用户发出 exit 指令

3.1.1 文件写入写出

根据不同类型，进行不同的读和写操作其中，读入文件的时候用了额外的数组

- 时间复杂度分析：$O(n)$
- 空间复杂度分析：$O(n)$

这里只展示输出文件 save 的实现，先判断文件类型，再对应输出

```
void save(char *filename, int length)
{
    if (this->arr == NULL)
    {
        cout << "Error!!! You have not initialized your array yet
           !" << endl;
```

```
        return;
    }
    char root[10] = "./output/";
    char *fullpath = new char[9 + length + 1];
    for (int i = 0; i < 9; i++)
    {
        fullpath[i] = root[i];
    }
    for (int i = 9; i < 9 + length; i++)
    {
        fullpath[i] = filename[i - 9];
    }
    fullpath[9 + length] = '\0';
    if (filename[length - 1] == 't')
    {
        ofstream fout(fullpath);
        fout << this->size << endl;
        for (int i = 0; i < this->size; i++)
        {
            fout << this->arr[i];
            if (i < this->size - 1)
                fout << " ";
        }
        fout.close();
    }
    else
    {
        ofstream fout(fullpath, ios::binary);
        fout.write((char *)&this->size, sizeof(int));
        fout.write("\n", sizeof(char));
        for (int i = 0; i < this->size; i++)
        {
            fout.write((char *)&this->arr[i], sizeof(int));
            if (i < this->size - 1)
                fout.write(" ", sizeof(char));
        }
        fout.close();
    }
    delete[] fullpath;
}
```

3.1.2 输入处理

共 9 种输入格式，分别对应题目要求的八个操作和退出，利用 funcname、op1、op2 三个字符数组存储对应的函数名、参数 1 和参数 2

- 时间复杂度分析：$O(n)$
- 空间复杂度分析：$O(n)$

```
        char instruction[1000];
        cout << "请输入: " << endl;
        if(loop_count==0) cin.ignore();
        cin.getline(instruction, 1000);
        // cout<<instruction<<endl;
        char func_name[1000] = {0}, op1[1000] = {0}, op2[1000] = {0};
        int id1 = 0, id2 = 0, id3 = 0;
        int face_blank = 0;
        for (int i = 0; instruction[i] != '\0'; i++)
        {
            if (instruction[i] == ' ')
            {
                face_blank++;
                continue;
            }
            switch (face_blank)
            {
            case 0:
            {
                func_name[id1] = instruction[i];
                id1++;
                break;
            }
            case 1:
            {
                op1[id2] = instruction[i];
                id2++;
                break;
            }
            case 2:
            {
                op2[id3] = instruction[i];
                id3++;
                break;
            }
```

```
            }
        }
```

3.1.3 函数名匹配

利用自定义 cmp 函数，返回对应 case 值，进入对应 switch 分支，执行对应函数

- 时间复杂度分析：$O(n)$
- 空间复杂度分析：$O(n)$

实现函数匹配

```
bool cmp(char* string1,char* string2,int length)
{
    for(int i=0;i<length;i++)
    {
        if(string1[i]!=string2[i]) return false;
    }
    return true;
}

int matching_mode(char *func_name, int size)
{
    if(size==4)
    {
        char func1[5]="size";
        char func2[5]="show";
        char func3[5]="save";
        char func4[5]="exit";
        if(cmp(func_name,func1,4)) return 1;
        else if(cmp(func_name,func2,4)) return 2;
        else if(cmp(func_name,func3,4)) return 8;
        else if(cmp(func_name,func4,4)) return 9;
        else return -1;
    }
    else if(size==5)
    {
        char func[6]="erase";
        if(cmp(func_name,func,5)) return 6;
        else return -1;
    }
    else if(size==6)
    {
```

```
        char func1[7]="insert";
        char func2[7]="update";
        if(cmp(func_name,func1,6)) return 5;
        else if(cmp(func_name,func2,6)) return 7;
        else return -1;
    }
    else if(size==8)
    {
        char func[9]="pop_back";
        if(cmp(func_name,func,8)) return 4;
        else return -1;
    }
    else if(size==9)
    {
        char func[10]="push_back";
        if(cmp(func_name,func,9)) return 3;
        else return -1;
    }
    else
    {
        return -1;
    }
}
```

3.1.4 向量初始化

将外部读入的数组存入向量

- 时间复杂度分析：$O(n)$
- 空间复杂度分析：$O(n)$

3.1.5 打印向量

- 时间复杂度分析：$O(n)$
- 空间复杂度分析：$O(n)$

3.1.6 向量尾部进行 push

需要重新分配动态内存

- 时间复杂度分析：$O(1)$
- 空间复杂度分析：$O(n)$

```
void push_back_x(int x)
{
    if (this->arr == NULL)
    {
        cout << "Error!!! You have not initialized your array yet
           !" << endl;
        return;
    }
    int *newarr = new int[this->size + 1];
    for (int i = 0; i < this->size; i++)
    {
        newarr[i] = this->arr[i];
    }
    newarr[this->size] = x;
    this->size++;
    delete[] this->arr;
    this->arr = newarr;
    this->show();
}
```

3.1.7 插入数据进向量某个位置

使得 pos 位置后所有元素都向后移动一位

- 时间复杂度分析: $O(n)$
- 空间复杂度分析: $O(n)$

```
void insert(int pos, int x)
{
    if (this->arr == NULL)
    {
        cout << "Error!!! You have not initialized your array yet
           !" << endl;
        return;
    }
    if(pos<=0||pos>this->size+1)
    {
        cout<<"Invalid index!!! please try another..."<<endl;
        return ;
    }
    int *newarr = new int[this->size + 1];
    for (int i = 0; i < pos - 1; i++)
```

```
        {
            newarr[i] = this->arr[i];
        }
        for (int i = this->size; i > pos - 1; i--)
        {
            newarr[i] = this->arr[i - 1];
        }
        newarr[pos - 1] = x;
        this->size++;
        delete[] this->arr;
        this->arr = newarr;
        this->show();
    }
```

3.1.8 向量尾部 pop

- 时间复杂度分析: $O(1)$
- 空间复杂度分析: $O(1)$

3.1.9 向量内部删除元素

pos 位置之后元素都要向前移动一位

- 时间复杂度分析: $O(n)$
- 空间复杂度分析: $O(1)$

```
    void erase(int pos)
    {
        if (this->arr == NULL)
        {
            cout << "Error!!! You have not initialized your array yet
               !" << endl;
            return;
        }
        if(pos<=0||pos>this->size)
        {
            cout<<"Invalid index!!! please try another..."<<endl;
            return ;
        }
        for (int i = pos; i < this->size; i++)
        {
            this->arr[i - 1] = this->arr[i];
```

```
        }
        this->size--;
        this->show();
    }
```

3.1.10 更新 pos 元素

- 时间复杂度分析：$O(1)$
- 空间复杂度分析：$O(1)$

4 实验环境

- 硬件环境：Macos M1 Pro、内存 16GB
- 软件环境：操作系统：Ubuntu 22.04 IDE：vscode language：cpp

5 实验结果与分析

在每次输入之后，都能够正确的完成对应的操作，
展示图片如下：

图 1: show vector

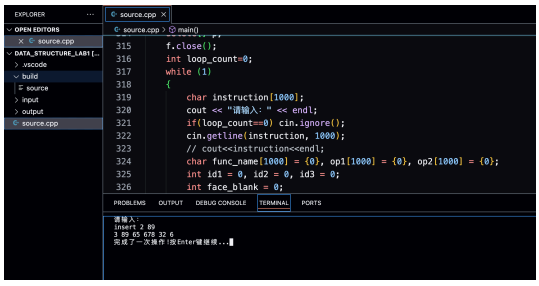

图 2: insert

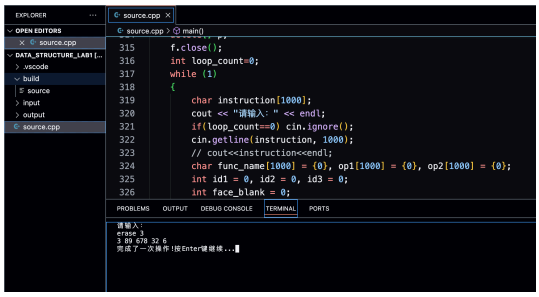

图 3: erase

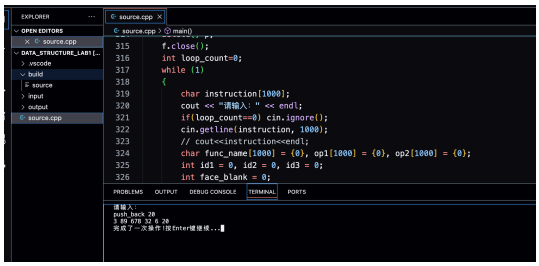

图 4: push$_b ack$

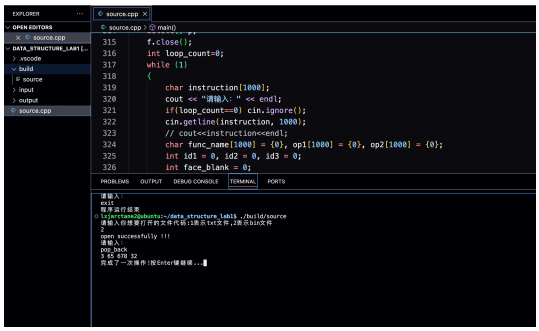

图 5: pop$_b ack$

图 6: save file