山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

学号：202300130183	姓名： 宋浩宇	班级：23 级人工智能班
实验题目：2024 数据结构--数据/智能 实验 3 数组描述线性表		
实验学时：2	实验日期： 2024/9/25	

实验目的：

1. 掌握线性表结构、数组描述方法（顺序存储结构）、数组描述线性表的实现。
2. 掌握线性表应用。

软件开发工具：

1. visual studio code 2022（使用 C/C++、C/C++ Extension Pack、C/C++ Themes 插件）
2. mingw64 工具包

1. 实验内容

完成 2024 数据结构--数据/智能 实验 3 数组描述线性表中的题目 A 通讯录实验。

2. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）

我们首先创建一个结构体 PersonalData 来存储通讯录中结构化的数据。

我们由此来编写 arraylist 类。其中 PersonalData 的指针 datas 用于指向 PersonalData 的数组，count 用于计量数组的元素个数。我们需要完成增删查改的功能，我们以此来描述实现的方式，其中增操作是创建一个 count+1 大小的 PersonalData 数组，将原本数组的数据拷贝过去，再将原本 datas 指向的内存空间释放掉，再指向新的空间，并将 count 自增。这样就可以完成数组形式的增操作。删操作的实现方式与增操作类似，我们创建一个 count-1 大小的数组，将原本数组中除了要删除的元素以外的元素都拷贝过来，再将原本 datas 指向的内存空间释放掉，再让 datas 指向新的空间，并将 count 自减，这样就可以完成删的操作。查操作比较简单，只需遍历 datas 指向的数组，如果其中的某个 PersonalData 的 name 元素与要查找的相同，则输出 1，否则输出 0。而改的操作也很简单，只需对 datas 指向的数组的对应元素进行重新赋值即可。但是存在这么一个问题，删操作和改操作是需要要被操作的元素的下标的，因此我们在查操作的基础上在进行修改，获得新的查找函数，该函数会返回需要查找的元素的下标。至此该类内部可以进行正常运行，增删查改操作也都可以进行。题目要求我们再写一个找出同一个班级的学生，实现起来也很简单只需要遍历数组即可。另外，因为实验要求我们自己来实现所有需要的算法，因此再自己手写一个 strcmp 函数和 strcpy 函数，因为题目说所有的数据都是合法的，因此不需要考虑这两个函数操作 C 风格字符串时的异常，实际实现也是很简单的。至此，本题结束。

3. 测试结果（测试输入，测试输出）

测试输入为：

```
28
0 Evan 57298577609 1 65
0 WINNIE 37367348390 4 1
3 Evan
4 6
3 WINNIE
```

```
1 Evan
4 7
1 WINNIE
3 MARYAM
3 CAMERON
3 TZIVIA
0 OMAR 16447001130 6 55
4 8
4 2
3 JADEN
3 ELIZABETH
2 OMAR 1 79409905568
3 JOSHUA
2 OMAR 1 8978214817
1 OMAR
3 Azaan
3 MARIA
0 HANNAH 94060479192 5 98
3 HEIDY
1 HANNAH
0 Axel 92066832927 3 70
1 Axel
3 TIFFANY
```

输出结果为：

```
1
0
1
0
0
0
0
0
0
0
0
0
0
0
0
0
```

4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）

从结果来看，我们这个题目是成功解决了的，但实际上因为我们本身在设计时，是以解决这个问题为导向的，并没有编写 arraylist 的类模板，实际上将增删查改更改为抽象操作，即

不处理具体数据而是只进行声明的模板 T 的拷贝，以及引用的返回，再加上运算符==的重载，该代码可重构出 arraylist 类模板。

2024/9/25 补充，经过修改，成功将代码中的 arraylist 类写成了一个可用的类模板，其中大部分运算与原本代码逻辑一致，主要区别在于，为了解决本题还需要书写一个运算符重载，以及将 PersonalData 结构体改写为类，并单独书写修改其中元素的函数（详见附录）

5. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）

```
/*2024 数据结构--数据智能 实验 3 数组描述线性表 A  通讯录.cpp*/
#include<iostream>
using namespace std;

struct PersonalData
{
    char name[20];
    char telenum[20];
    char classnum[10];
    int house;
};
class arraylist
{
private:
    PersonalData* datas = nullptr;
    int count = 0;
    bool equal(const char* a, const char* b)
    {
        int count = 0;
        while (true)
        {
            if (a[count]=='\0' && b[count] == '\0')
            {
                return true;
            }
            if (a[count]!= b[count])
            {
                return false;
            }
            count++;
        }
    };
    void cpy(char* a, const char* b)
    {
        int count = 0;
        while (true)
        {
            a[count] = b[count];
```

```
39.             if (b[count] =='\0')
40.             {
41.                 return;
42.             }
43.         count++;
44.         }
45.     }
46. public:
47.     void insert();
48.     void disp();
49.     void del(int index);
50.     bool check(const char* name);
51.     void edit(int index, int num);
52.     int find(const char* name);
53.     void operation();
54.     void XOR(const char* num);
55. };
56. 
57. class Solution
58. {
59. public:
60.     void solute()
61.     {
62.         arraylist line;
63.         int n;
64.         cin >> n;
65.         for (int i = 0; i < n; i++)
66.         {
67.             line.operation();
68.         }
69.     }
70. };
71. 
72. void arraylist::insert()
73. {
74.     PersonalData newdata;
75.     cin >> newdata.name >> newdata.telenum >> newdata.classnum >> newdata.house;
76.     PersonalData* newlist = new PersonalData[count + 1];
77.     for (int i = 0;i < count;i++)
78.     {
79.         newlist[i] = datas[i];
80.     }
81.     count++;
82.     newlist[count-1] = newdata;
83.     delete [] datas;
```

```
84.         datas = newlist;
85.     }
86.
87.     void arraylist::disp()
88.     {
89.         for (int i = 0; i < count; i++)
90.         {
91.             cout << datas[i].name << " " << datas[i].telenum << " " << datas[i].classnum << " " << datas[i].house << endl;
92.         }
93.     }
94.
95.     void arraylist::del(int index)
96.     {
97.         PersonalData* newlist = new PersonalData[count - 1];
98.         for (int i = 0; i < count; i++)
99.         {
100.            if (i<index)
101.            {
102.                newlist[i] = datas[i];
103.            }
104.            if (i == index)
105.            {
106.                continue;
107.            }
108.            if (i>index)
109.            {
110.                newlist[i - 1] = datas[i];
111.            }
112.        }
113.        delete[] datas;
114.        count--;
115.        datas = newlist;
116.    }
117.
118.    bool arraylist::check(const char* name)
119.    {
120.        for (int i = 0; i < count; i++)
121.        {
122.            if (equal(datas[i].name , name))
123.            {
124.                cout << 1 << endl;
125.                return true;
126.            }
127.        }
128.        cout << 0 << endl;
```

```
129.     return false;
130. }
131. 
132. void arraylist::edit(int index,int num)
133. {
134.     if (num == 1)
135.     {
136.         char newdata[20];
137.         cin >> newdata;
138.         cpy(datas[index].telenum, newdata);
139.     }
140.     if (num == 2)
141.     {
142.         char newdata[10];
143.         cin >> newdata;
144.         cpy(datas[index].classnum, newdata);
145.     }
146.     if (num == 3)
147.     {
148.         int newdata;
149.         cin >> newdata;
150.         datas[index].house = newdata;
151.     }
152. }
153. 
154. int arraylist::find(const char* name)
155. {
156.     for (int i = 0; i < count; i++)
157.     {
158.         if (equal(datas[i].name,name))
159.         {
160.             return i;
161.         }
162.     }
163. 
164. }
165. 
166. void arraylist::XOR(const char* num)
167. {
168.     int ans = 0;
169.     // bool mark = 1;
170.     for (int i = 0; i < count; i++)
171.     {
172.         if (equal(datas[i].classnum,num))
173.         {
174.             ans = ans ^ datas[i].house;
```

```
        }
    }
    cout << ans << endl;
}
void arraylist::operation()
{
    int mark;
    cin >> mark;
    if (mark==0)
    {
        insert();
    }

    if (mark == 1)
    {
        char name[20];
        cin >> name;
        del(find(name));
    }
    if (mark==2)
    {
        char name[20];
        cin >> name;
        int a;
        cin >> a;
        edit(find(name), a);
    }
    if (mark==3)
    {
        char name[20];
        cin >> name;
        check(name);
    }
    if (mark==4)
    {
        char classnum[10];
        cin >> classnum;
        XOR(classnum);
    }
}

int main()
{
    Solution ans;
    ans.solute();
}
```

```cpp
/*2024 数据结构--数据智能 实验 3 数组描述线性表 A  通讯录.cpp*/
/*类模板版本*/
#include<iostream>
using namespace std;


class PersonalData
{
private:
    bool strequal(const char* a, const char* b) const
    {
        int count = 0;
        while (true)
        {
            if (a[count]=='\0' && b[count] == '\0')
            {
                return true;
            }
            if (a[count]!= b[count])
            {
                return false;
            }
            count++;
        }
    };
    void strcpy(char* a, const char* b)
    {
        int count = 0;
        while (true)
        {
            a[count] = b[count];
            if (b[count] =='\0')
            {
                return;
            }
        count++;
        }
    }


public:

    char name[20];
```

```
46.         char telenum[20];
47.         char classnum[10];
48.         int house;
49.         bool operator==(const PersonalData& compared) const;
50.         PersonalData(const PersonalData& copyed);
51.         PersonalData(int mark);
52.         PersonalData() {};
53.         PersonalData(const char* name) { strcpy(this->name, name);strcpy(this->classn
    um, "\0"); strcpy(this->telenum, "\0"); this->house = 0; };
54.         void edit();
55.         int getHouse() { return this->house; };
56.         bool checkClassnum(const char* target) { return this->strequal(this->classnum
    , target); };
57.         void disp() { cout << name << endl; }
58.     };
59.
60.     bool PersonalData::operator==(const PersonalData& compared) const
61.     {
62.         return this->strequal(this->name, compared.name);
63.     }
64.
65.     PersonalData::PersonalData(const PersonalData& copyed)
66.     {
67.         this->strcpy(this->name, copyed.name);
68.         this->strcpy(this->classnum, copyed.classnum);
69.         this->strcpy(this->telenum, copyed.telenum);
70.         this->house = copyed.house;
71.     }
72.
73.     PersonalData::PersonalData(int mark)
74.     {
75.         cin >> this->name;
76.         cin >> this->telenum;
77.         cin >> this->classnum;
78.         cin >> this->house;
79.     }
80.
81.     void PersonalData::edit()
82.     {
83.         int operation;
84.         cin >> operation;
85.         if (operation == 1)
86.         {
87.             char tele[20];
88.             cin >> tele;
89.             this->strcpy(this->telenum, tele);
```

```
90.         }
91.         if (operation == 2)
92.         {
93.             char clas[10];
94.             cin >> clas;
95.             this->strcpy(this->classnum, clas);
96.         }
97.         if (operation == 3)
98.         {
99.             int new_house;
100.            cin >> new_house;
101.            this->house = new_house;
102.        }
103.    }
104.
105.    template<class T>
106.    class arraylist
107.    {
108.    private:
109.        T* datas;
110.        int count;
111.        void copy(T* copyed, T* to);
112.
113.    public:
114.        T& operator[](const unsigned int subscript) { return datas[subscript]; };
115.        int find_first(const T& target);
116.        void push_back(const T& element);
117.        void erase(const T& target);
118.        bool is_in(const T& target);
119.        int size() { return count; };
120.        arraylist();
121.    };
122.
123.    template<class T>
124.    void arraylist<T>::copy(T* copyed, T* to)
125.    {
126.        for (size_t i = 0; i < count; i++)
127.        {
128.            to[i] = copyed[i];
129.        }
130.    }
131.
132.    template<class T>
133.    void arraylist<T>::push_back(const T& element)
134.    {
135.        T* new_datas = new T[count + 1];
```

```
136.     copy(datas, new_datas);
137.     new_datas[count] = element;
138.     count++;
139.     delete[] datas;
140.     datas = new_datas;
141. }
142. 
143. template<class T>
144. int arraylist<T>::find_first(const T& target)
145. {
146.     for (size_t i = 0; i < count; i++)
147.     {
148.         if (target == datas[i])
149.         {
150.             return i;
151.         }
152.     }
153.     return -1;
154. }
155. 
156. template<class T>
157. void arraylist<T>::erase(const T& target)
158. {
159.     int index = find_first(target);
160. 
161.     T* new_datas = new T[count - 1];
162. 
163.     for (size_t i = 0; i < count; i++)
164.     {
165.         if (i < index)
166.         {
167.             new_datas[i] = datas[i];
168.         }
169.         if (i == index)
170.         {
171.             continue;
172.         }
173.         if (i > index)
174.         {
175.             new_datas[i - 1] = datas[i];
176.         }
177.     }
178. 
179.     delete[] datas;
180. 
181.     count--;
```

```
182.
183.     datas = new_datas;
184. }
185.
186. template<class T>
187. bool arraylist<T>::is_in(const T& target)
188. {
189.     int index = this->find_first(target);
190.
191.     if (index != -1)
192.     {
193.         return true;
194.     }
195.     if (index == -1)
196.     {
197.         return false;
198.     }
199.     return false;
200. }
201.
202. template<class T>
203. arraylist<T>::arraylist()
204. {
205.     datas = nullptr;
206.     count = 0;
207. }
208.
209. class Solution
210. {
211. public:
212.     void solute();
213.     void test();
214. };
215.
216. void Solution::solute()
217. {
218.     int n;
219.     cin >> n;
220.     int operation;
221.     arraylist<PersonalData> data;
222.
223.     for (size_t i = 0; i < n; i++)
224.     {
225.         cin >> operation;
226.         if (operation == 0)
227.         {
```

```
228.                PersonalData newdata(1);
229.                data.push_back(newdata);
230.            }
231.
232.            if (operation == 1)
233.            {
234.                char name[20];
235.                cin >> name;
236.                data.erase(name);
237.            }
238.            if (operation == 2)
239.            {
240.                char name[20];
241.                cin >> name;
242.                data[data.find_first(name)].edit();
243.            }
244.            if (operation == 3)
245.            {
246.                char name[20];
247.                cin >> name;
248.                cout << data.is_in(name);
249.            }
250.            if (operation == 4)
251.            {
252.                int ans = 0;
253.                char target[10];
254.                cin >> target;
255.                for (size_t i = 0; i < data.size(); i++)
256.                {
257.                    if (data[i].checkClassnum(target))
258.                    {
259.                        ans = ans ^ data[i].getHouse();
260.                    }
261.                }
262.                cout << ans;
263.            }
264.        }
265.
266.    }
267.
268.    void Solution::test()
269.    {
270.        arraylist<int> list;
271.        list.push_back(1);
272.        list.push_back(1);
273.        list.push_back(1);
```

```
    for (int i = 0;i < list.size();i++)
    {
        cout << list[i];
    }
    list.erase(1);
    for (int i = 0;i < list.size();i++)
    {
        cout << list[i];
    }
}


int main()
{
    Solution ans;
    ans.solute();
    // ans.test();
    // system("pause");
}
```