高级程序设计 2020 春

实验报告

指导老师：陈家骏 黄书剑 姓名：王晨渊, 学号：181220057

1 概念题

1.1 请说出 C++ 中同类对象共享数据的两种方式，并比较它们的优缺点。

一种方法是使用全局变量存储共享数据。优点在于使用自由，书写简便。缺点在于缺少对数据的保护，由于其不受类的访问控制的限制，任何类的对象都可以自由的访问或修改。

另一种方法是使用静态数据成员和静态成员函数。优点是在实现数据共享的同时，一方面使其收到类的访问控制的限制，提高数据的安全性，另一方面静态数据成员对该类的所有对象仅存在一个拷贝，节省了内存使用。缺点在于使用时要遵循繁琐的访问控制规范，代码不够简洁。

1.2 下面对静态数据成员的描述中，正确的是 (B)

A. 静态数据成员不可以通过类的对象调用
B. 静态数据成员可以在类体内进行初始化
C. 静态数据成员不能受 private(私有) 控制符的作用
D. 静态数据成员可以直接通过类名调用

1.3 已知类 A 是类 B 的友元，类 B 是类 C 的友元，则 (C)

A. 类 A 一定是类 C 的友元
B. 类 C 一定是类 A 的友元
C. 类 C 的成员函数可以访问类 B 的对象的任何成员
D. 类 A 的成员函数可以访问类 B 的对象的任何成员

1.4 简述 C++ 中的迪米特法则 (Law Of Demeter)，遵循迪米特法则设计的模块具有哪些优点？

一个类的成员函数除了能访问自身类结构的直接子结构（表层子结构）外，不能以任何方式依赖于任何其他类的结构；并且每个成员函数只应向某个有限集合中的对象发送信息。

对于类 C 中的任何成员函数 M，M 中能直接访问或引用的对象必须属于下述类之一：

1) 类 C 本身
2) 成员函数 M 的参数类
3)M 或 M 所调用的成员函数所创建的对象类
4) 全局对象所属的类

5) 类 C 的成员对象所属的类
好处在于可以降低模块间的耦合度和成员函数对环境的依赖性。

2 编程题

2.1 Array 类和 Matrix 类

```
1
2   class Array
3   {
4   public:
5       Array(int length, int column)
6           : len(length), col(column)
7       {
8           array = new double[length];
9       }
10      Array(int length, int column, double origin[], int origin_len)
11          : len(length), col(column)
12      {
13          array = new double[length];
14          for (int i = 0; i < len; i++)
15          {
16              if (i < origin_len)
17              {
18                  array[i] = origin[i];
19              }
20              else
21              {
22                  array[i] = 0;
23              }
24          }
25      }
26      inline double loc(int x, int y) const
27      {
28          return array[x * col + y];
29      }
30      inline double get(int x) const
31      {
32          return array[x];
```

```
33          }
34
35          inline int get_col() const
36          {
37              return col;
38          }
39          inline int get_row() const
40          {
41              return len / col;
42          }
43          inline int get_len() const
44          {
45              return len;
46          }
47          inline void swap()
48          {
49              for (int i = 0; i < len; i++)
50              {
51                  array[i] = -array[i];
52              }
53          }
54          inline void output() const
55          {
56              for (int i = 0; i < len; i++)
57              {
58                  if (i % col == col - 1)
59                  {
60                      cout << array[i] << ",\n";
61                  }
62                  else
63                  {
64                      cout << array[i] << ",";
65                  }
66              }
67          }
68          Array(const Array &a)
69          {
70              len = a.get_len();
71              col = a.get_col();
```

```
72              array = new double[len];
73              for (int i = 0; i < len; i++)
74              {
75                  array[i] = a.get(i);
76              }
77          }
78          ~Array()
79          {
80              delete[] array;
81          }
82
83      private:
84          int len;
85          int col;
86          double *array;
87      };
88
89      class Matrix
90      {
91      public:
92          Matrix(int this_row, int this_col, double origin[], int origin_len)
93              : row(this_row),
94                col(this_col),
95                arr(row * col, col, origin, origin_len)
96          {
97          }
98          Matrix(const Matrix &a)
99              : row(a.get_row()),
100               col(a.get_col()),
101               arr(a.arr)
102         {
103             arr.swap();
104         }
105         inline int get_col() const
106         {
107             return col;
108         }
109         inline int get_row() const
110         {
```

```
111             return row;
112         }
113         inline void output() const
114         {
115             arr.output();
116         }
117 
118     private:
119         int row, col;
120         Array arr;
121     };
```

2.2 String 类及其拼接函数

```
1     class String
2     {
3     public:
4         String(char* s)
5         {
6             if(s)
7             {
8                 len = strlen(s);
9                 str = new char[len+1];
10                strcpy(str,s);
11            }
12            else
13            {
14                len = 0;
15                str = NULL;
16            }
17        }
18        String(const String& s)
19        {
20            len = s.get_len();
21            set(s.get_str());
22        }
23        int get_len() const
24        {
25            return len;
```

```
26          }
27          char* get_str() const
28          {
29              return str;
30          }
31          bool is_NULL() const
32          {
33              return len==0;
34          }
35
36          void set(char* s)
37          {
38              if(s)
39              {
40                  delete[] str;
41              }
42              int s_len = strlen(s);
43              str = new char[s_len+1];
44              strcpy(str,s);
45          }
46          void clear()
47          {
48              delete[] str;
49              str = NULL;
50          }
51
52          static int mystrcat(const String& str1, const String& str2, String& str3)
53          {
54              if(str1.is_NULL()&&str2.is_NULL())
55              {
56                  cerr<<"Both are NULL\n";
57                  return 0;
58              }
59              else if(str1.is_NULL())
60              {
61                  str3.set(str2.get_str());
62                  return strlen(str2.get_str());
63              }
64              else if(str2.is_NULL())
```

```
65              {
66                  str3.set(str1.get_str());
67                  return strlen(str1.get_str());
68              }
69              else
70              {
71                  int new_s_len = str1.get_len()+str2.get_len();
72                  char* new_s = new char[new_s_len+1];
73                  strcpy(new_s,str1.get_str());
74                  strcpy(&new_s[str1.get_len()],str2.get_str());
75                  str3.set(new_s);
76                  delete new_s;
77                  return strlen(new_s);
78              }
79          }
80          void output()
81          {
82              cout<<str<<endl;
83          }
84          ~String()
85          {
86              delete[] str;
87              str = NULL;
88              len = 0;
89          }
90      private:
91          char* str;
92          int len;
93      };
```

2.3 Single 类

规定只能通过 Single::creat_obj() 和 Single::del_obj() 两个函数来创建或消亡 Single 类对象。

```
1           class Single
2           {
3           public:
4               static Single* create_obj();
5               static void del_obj();
6               Single()
```

```
7           {
8               count_object++;
9
10          }
11          ~Single()
12          {
13              count_object--;
14          }
15
16      private:
17          static int get_count_obj()
18          {
19              return count_object;
20          }
21          static int count_object;
22          static Single* the_only;
23      };
24      int Single::count_object = 0;
25      Single* Single::the_only = NULL;
26      Single* Single::create_obj()
27      {
28          if(Single::get_count_obj())
29          {
30              cerr<<"Object Creation Failed!"<<endl;
31          }
32          else
33          {
34              Single::the_only = new Single;
35              Single::count_object++;
36
37          }
38          return Single::the_only;
39
40      }
41      void Single::del_obj()
42      {
43          if(Single::get_count_obj())
44          {
45              Single::count_object--;
```

```
46                  delete Single::the_only;
47                  Single::the_only = NULL;
48              }
49              else
50              {
51                  cerr<<"There is no object!"<<endl;
52              }
53          }
```