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 简述 C++ 中类属的概念。

一个程序实体能对多种类型的数据进行操作或描述的特性称为类属

1.2 C++ 提供了哪两种实现类属函数的机制? 简述它们的缺点。

1 采用 void* 类型的参数。比较麻烦，需要大量的指针操作。容易出错，编译程序无法进行类型检查。**2** 函数模板。模板复用会产生相同的实例，需要额外处理。

1.3 简述 C++ 中参数化多态的概念及作用。

概念：一段带有类型参数的代码，给该参数提高不同类型就能得到多个不同的代码，即一段代码有多种解释。作用：实现源代码复用。

2 编程题

2.1 Phone

```
#include<iostream>
using namespace std;
template <typename T>
void f(T){ cout<<"f(T)"<<endl; }
template <typename T>
void f(const T*){ cout<<"f(const T*)"<< endl; }
template <typename T>
void g(T){ cout<<"g(T)"<<endl; }
template <typename T>
void g(T*){ cout<<"g(T*)"<<endl; }
int main(){
    int a=1;
```

```
13          int* b=&a;
14          const int c=0;
15          const int*d=&c;
16          f(a);//f(T) int
17          f(b);//f(T) int*
18          f(c);//f(T) const int
19          f(d);//f(const T*) int
20          g(a);//g(T) int
21          g(b);//g(T*) int
22          g(c);//g(T) const int
23          g(d);//g(T*) const int
24      }
```

2.2 链表

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4   template <class T>
5   class Node
6   {
7   public:
8       Node(){next=NULL;}
9       T& val(){return value;}
10      Node*& pnext(){return next;}
11  private:
12      T value;
13      Node *next;
14  };
15
16  template<class T>
17  class List
18  {
19  public:
20      List()
21      {
22          head=NULL;
23          tail=NULL;
24      }
```

```
25          ~List()
26          {
27              Node<T> *tmp =head;
28              if(!tmp)
29              {
30                  Node<T> *tmp2=tmp;
31                  tmp = tmp->pnext();
32                  delete tmp2;
33              }
34          }
35          void add(T val)
36          {
37              if(head!=NULL)
38              {
39                  tail->pnext() = new Node<T>;
40                  tail->pnext()->val()=val;
41                  tail = tail->pnext();
42              }
43              else
44              {
45                  tail = new Node<T>;
46                  tail->val()=val;
47                  head=tail;
48              }
49          }
50          void display()
51          {
52              if(head!=NULL)
53              {
54                  Node<T> *tmp = head;
55                  while(tmp!=NULL)
56                  {
57                      cout<<tmp->val()<<"□";
58                      tmp = tmp->pnext();
59                  }
60              }
61
62          }
63
```

```
64      private:
65          Node<T> *head,*tail;
66      };
67
68      int main()
69      {
70          List<int> mylist;
71          mylist.add(21);
72          mylist.add(12);
73          mylist.display();
74          cout<<endl;
75          List<double> mylist2;
76          mylist2.add(1.2);
77          mylist2.add(2.1);
78          mylist2.display();
79          cout<<endl;
80          List<string> mylist3;
81          mylist3.add("abd");
82          mylist3.add("adadada");
83          mylist3.display();
84      }
```

2.3 矩阵

```
1       #include <iostream>
2       #include <cassert>
3       using namespace std;
4       template <class T> class Matrix
5       {
6       public:
7           Matrix(int m, int n)
8           {
9               row = m;
10              col = n;
11              all = new T*[m];
12              for(int i=0;i<m;i++)
13              {
14                  all[i] = new T[n];
15              }
```

```
16          }
17          ~Matrix()
18          {
19              for(int i=0;i<row;i++)
20              {
21                  delete[] all[i];
22              }
23              delete[] all;
24          }
25
26          Matrix(const Matrix& mat)
27          {
28              row = mat.row;
29              col = mat.col;
30              all = new T*[row];
31              for(int i=0;i<row;i++)
32              {
33                  all[i] = new T[col];
34              }
35              for(int i=0;i<row;i++)
36              {
37                  for(int j=0;j<col;j++)
38                  {
39                      all[i][j]=mat.all[i][j];
40                  }
41              }
42          }
43          Matrix& operator= (const Matrix& mat)
44          {
45              assert(row==mat.row && col==mat.col);
46              if(this==*mat) return *this;
47              for(int i=0;i<row;i++)
48              {
49                  for(int j=0;j<col;j++)
50                  {
51                      all[i][j]=mat.all[i][j];
52                  }
53              }
54          }
```

```
55          void display()
56          {
57              for(int i=0;i<row;i++)
58              {
59                  for(int j=0;j<col;j++)
60                  {
61                      cout<<all[i][j]<<"□";
62                  }
63                  cout<<endl;
64              }
65          }
66          void transpose()
67          {
68              T** new_all= new T*[col];
69              for(int i=0;i<col;i++)
70              {
71                  new_all[i] = new T[row];
72              }
73              for(int i=0;i<row;i++)
74              {
75                  for(int j=0;j<col;j++)
76                  {
77                      new_all[j][i]=all[i][j];
78                  }
79              }
80              for(int i=0;i<row;i++)
81              {
82                  delete[] all[i];
83              }
84              delete[] all;
85              all = new_all;
86              int tmp = row;
87              row = col;
88              col = tmp;
89          }
90          void setMatrix()
91          {
92              for(int i=0;i<row;i++)
93              {
```

```
94                  for(int j=0;j<col;j++)
95                  {
96                      cin>>all[i][j];
97                  }
98              }
99          }
100         void debug()
101         {
102             cout<<"row:"<<row<<"col:"<<col<<endl;
103         }
104
105
106         Matrix operator+(const Matrix &mat)
107         {
108             Matrix res(*this);
109             for(int i=0;i<row;i++)
110             {
111                 for(int j=0;j<col;j++)
112                 {
113                     res.all[i][j]=mat.all[i][j]+all[i][j];
114                 }
115             }
116             return res;
117         }
118         Matrix operator*(const Matrix &mat)
119         {
120             assert(col==mat.row);
121             Matrix res(row, mat.col);
122             for(int i=0;i<row;i++)
123             {
124                 for(int j=0;j<mat.col;j++)
125                 {
126                     T sum;
127                     for(int k=0;k<mat.row;k++)
128                     {
129                         if(!k)
130                         {
131                             sum = all[i][k]*mat.get(k,j);
132                         }
```

```
133                         else
134                         {
135                             sum+= all[i][k]*mat.get(k,j);
136                         }
137                     }
138                     res.get(i,j)=sum;
139                 }
140             }
141             return res;
142         }
143         void square()
144         {
145             Matrix<T> copy(*this);
146             copy.transpose();
147             Matrix<T>res = *this * copy;
148             res.display();
149 
150         }
151         T& get(int i, int j)
152         {
153             assert(i<row&&j<col);
154             return all[i][j];
155         }
156         T get(int i, int j) const
157         {
158             assert(i<row&&j<col);
159             return all[i][j];
160         }
161 
162 
163 
164     private:
165         T** all;
166         int row,col;
167     };
168 
169 
170 
171     int main()
```

```
172     {
173     #ifdef TEST1
174         Matrix<int> matrix1(2,3);
175         matrix1.setMatrix();
176         matrix1.display();
177         matrix1.transpose();
178         matrix1.display();
179         Matrix<int> matrix2(matrix1);
180         matrix2.transpose();
181         matrix2.display();
182 
183     #else
184 
185         Matrix<double> matrix1(2,3);
186         matrix1.setMatrix();
187         matrix1.display();
188         Matrix<double> matrix2(2,2);
189         matrix2.setMatrix();
190         matrix2.square();
191         Matrix<double> matrix3(2,3);
192         matrix3.setMatrix();
193         (matrix1+matrix3).display();
194     #endif
195 
196     }
```