SP SITHUNGU

200000000

PRACTICAL 7 DESIGN

```
T:typename
                              GenericStorage2D
- data: T**
- rows: int
 cols: int
+DEFAULT ROWS: const int = 2
+DEFAULT COLS: const int = 2
+MIN DIMENSION SIZE: const int = 2
-MAX DIMENSION SIZE: const int = 100000
+GenericStorage2D()
+GenericStorage2D(intRows:int,intCols:int)
+GenericStorage2D(objOriginal:const GenericStorage<T>&)
+operator=(objRHS:const GenericStorage2D<T>&): const GenericStorage2D<T>&
+operator()(intRow:int,intCol:int): T&
+operator<<(istream&:isLHS,objRHS:const GenericStorage2D<T1>& objRHS): istream&
+operator==(objRHS:const GenericStorage2D<T>&): bool
+operator!=(objRHS:const GenericStorage2D<T>&): bool
+getRows(): int
+getCols(): int
+~GenericStorage2D()
-toString()(): string
-getValueAt(intRow:int,intCol:int): T
-setValueAt(intRow:int,intCol:int,tValue:T): void
-alloc(intRows:int,intCols:int): void
-dealloc(): void
-clone(objOriginal:const GenericStorage2D<T>&): void
-enforceRange(intArg:int,intMin:int,intMax:int): void
```

```
#ifndef GenericStorage2D_H
 1
   #define GenericStorage2D_H
 3
 4
   #include <iostream>
5
   #include <string>
 6
   enum ERROR_CODE{
7
8
       SUCCESS,
9
        ERROR_ARGS,
10
        ERROR_RANGE
11
   };
12
13
   template <typename T>
14
   class GenericStorage2D{
15 public:
        GenericStorage2D();
16
17
        GenericStorage2D(int intRows, int intCols);
18
        GenericStorage2D(const GenericStorage2D<T>& objOriginal);
19
20
        const GenericStorage2D& operator=(const GenericStorage2D<T>& objRHS);
21
        T& operator()(int intRow, int intCol);
22
        template<typename T1>
23
        friend std::ostream& operator<<(std::ostream& osLHS, const GenericStorage2D<T1>&
objRHS);
        template<typename T1>
24
        friend std::istream& operator<<(std::istream& osLHS, const GenericStorage2D<T1>&
25
objRHS);
26
        bool operator==(const GenericStorage2D<T>& objRHS);
27
       bool operator!=(const GenericStorage2D<T>& objRHS);
28
29
        int getRows() const;
30
        int getCols() const;
31
32
        static const int DEFAULT_ROWS = 2;
33
        static const int DEFAULT_COLS = 2;
34
        static const int MIN_DIMENSION_SIZE = 2;
35
        static const int MAX_DIMENSION_SIZE = 100000;
36
37
        ~GenericStorage2D();
38 private:
39
        std::string toString() const;
        int getValueAt(int intRow, int intCol) const;
40
        void setValueAt(int intRow, int intCol, T tValue);
41
        void alloc(int intRows, int intCols);
42
43
        void dealloc();
44
        void clone(const GenericStorage2D<T>& objOriginal);
45
        void enforceRange(int intArg, int intMin, int intMax) const;
46
        T** _data;
        int _rows;
47
48
        int _cols;
49
   };
50
51
   #include "GenericStorage2D.imp"
52
53
   #endif // GenericStorage2D H
```

```
#include <cassert>
  1
  2 #include <iostream>
    #include <sstream>
  3
    #include <string>
  4
 5
  6
    using namespace std;
  7
  8 template <typename T>
 9 GenericStorage2D<T>::GenericStorage2D() : GenericStorage2D(DEFAULT_ROWS,
DEFAULT_COLS){}
 10
 11 template <typename T>
12 GenericStorage2D<T>::GenericStorage2D(int intRows, int intCols){
13
         alloc(intRows, intCols);
14 }
15
 16 template <typename T>
 17 GenericStorage2D<T>::GenericStorage2D(const GenericStorage2D& objOriginal)
 18
                         : GenericStorage2D(objOriginal._rows, objOriginal._cols){
 19
         clone(objOriginal);
 20
    }
 2.1
 22
    template <typename T>
    const GenericStorage2D<T>& GenericStorage2D<T>::operator=(const GenericStorage2D<T>&
 23
objRHS){
 24
         if(this != &objRHS){ // Check for self-assignment.
 25
             dealloc();
 26
             alloc(objRHS._rows, objRHS._cols);
 27
             clone(objRHS);
 2.8
         return *this;
 29
 30
 31
    template <typename T>
 32
    T& GenericStorage2D<T>::operator()(int intRow, int intCol){
 33
 34
         enforceRange(intRow, 0, _rows - 1);
         enforceRange(intCol, 0, _cols - 1);
 35
 36
         return _data[intRow][intCol];
 37
 38
 39
    template <typename T1>
 40
    ostream& operator << (ostream& osLHS, const GenericStorage 2D < T1 > & objRHS) {
 41
 42
          * We can use the already existing
 43
          * toString member function from P5.
 44
 45
         osLHS << objRHS.toString() << endl;
 46
         return osLHS;
 47
 48
 49
    template <typename T1>
 50
    istream& operator<<(istream& isLHS, const GenericStorage2D<T1>& objRHS){
         for(int r = 0; r < objRHS._rows; r++){
 51
 52
             for(int c = 0; c < objRHS._cols; c++){</pre>
 53
                 isLHS >> objRHS._data[r][c];
 54
 55
 56
         return isLHS;
 57
    }
 58
 59
    template <typename T>
 60 bool GenericStorage2D<T>::operator==(const GenericStorage2D<T>& objRHS){
 61
         if(_rows != objRHS._rows | _cols != objRHS._cols)
 62
             return false;
 63
 64
         for(int r = 0; r < _rows; r++){</pre>
```

```
for(int c = 0; c < _cols; c++){</pre>
 65
 66
                 if(_data[r][c] != objRHS._data[r][c])
 67
                      return false;
 68
 69
 70
         return true;
 71
 72
 73
    template <typename T>
 74
    bool GenericStorage2D<T>::operator!=(const GenericStorage2D<T>& objRHS){
 75
         if(_rows != objRHS._rows | _cols != objRHS._cols)
 76
             return true;
 77
 78
         for(int r = 0; r < _rows; r++){</pre>
             for(int c = 0; c < _cols; c++){</pre>
 79
                 if(_data[r][c] != objRHS._data[r][c])
 80
 81
                      return true;
 82
 83
 84
         return false;
 85
 86
 87
    template <typename T>
 88 string GenericStorage2D<T>::toString() const{
 89
         stringstream ssReturn;
         for(int r = 0; r < _rows; r++){</pre>
 90
             for(int c = 0; c < _cols; c++){</pre>
 91
 92
                 ssReturn << _data[r][c] << ' ';
 93
 94
             ssReturn << endl;
 95
 96
         return ssReturn.str();
 97
 98
 99
     template <typename T>
100
     int GenericStorage2D<T>::getRows() const{
101
         return _rows;
102
103
104
     template <typename T>
105
     int GenericStorage2D<T>::getCols() const{
106
         return _cols;
107
108
109
     template <typename T>
110
     int GenericStorage2D<T>::getValueAt(int intRow, int intCol) const{
111
         enforceRange(intRow, 0, _rows - 1);
112
         enforceRange(intCol, 0, _cols - 1);
113
         return _data[intRow][intCol];
114
115
116
    template <typename T>
117
    void GenericStorage2D<T>::setValueAt(int intRow, int intCol, T tValue){
         enforceRange(intRow, 0, _rows - 1);
118
119
         enforceRange(intCol, 0, _cols - 1);
120
         _data[intRow][intCol] = tValue;
121
122
123
    template <typename T>
124
    void GenericStorage2D<T>::alloc(int intRows, int intCols){
125
         enforceRange(intRows, MIN_DIMENSION_SIZE, MAX_DIMENSION_SIZE);
126
         enforceRange(intCols, MIN_DIMENSION_SIZE, MAX_DIMENSION_SIZE);
         _rows = intRows;
127
128
         _cols = intCols;
129
         _data = new int*[_rows];
130
         for(int r = 0; r < _rows; r++){</pre>
```

```
131
             _data[r] = new int[_cols];
132
133
134
135 template <typename T>
136 void GenericStorage2D<T>::dealloc(){
         assert(_data != nullptr);
137
138
         for(int r = 0; r < _rows; r++){</pre>
139
             delete [] _data[r];
140
141
         delete [] _data;
142
143
144 template <typename T>
145 void GenericStorage2D<T>::clone(const GenericStorage2D& obj0riginal){
146
         for(int r = 0; r < _rows; r++){</pre>
147
             for(int c = 0; c < _cols; c++){</pre>
148
                 _data[r][c] = objOriginal._data[r][c];
149
         }
150
151
152
153
    template <typename T>
154 void GenericStorage2D<T>::enforceRange(int intArg, int intMin, int intMax) const{
155
             if(intArg < intMin | intArg >intMax){
                 cerr << intArg << " must be in [" << intMin</pre>
156
157
                      << ", " << intMax << "]" << endl;
                 exit(ERROR_RANGE);
158
             }
159
160
161
162
     template <typename T>
163
     GenericStorage2D<T>::~GenericStorage2D(){
164
         dealloc();
165
```

```
1 #include "GenericStorage2D.h"
 3
   #include <iostream>
 4
 5 using namespace std;
 6
 7
   int rangedRandom(int intLB, int intUB);
 8
9 int main()
10 {
11
        // Testing the fully parameterised constructor.
12
        GenericStorage2D<int> objStorage(10, 10);
13
        // Testing the copy constructor.
        GenericStorage2D<int> objCopy = objStorage;
14
        cout << "objCopy == objStorage? ";</pre>
15
        cout << (objCopy == objStorage ? "Yes" : "No") << endl;</pre>
16
17
        //Testing the indexing operator.
        for(int r = 0; r < objCopy.getRows(); r++){</pre>
18
19
            for(int c = 0; c < objCopy.getCols(); c++){</pre>
20
                objCopy(r, c) = rangedRandom(0, 8);
21
22
23
        // Testing the overloaded assignment operator.
24
        GenericStorage2D<int> objSmallerSizedStorage(2, 2);
25
        objSmallerSizedStorage = objCopy;
        cout << "objSmallerSizedStorage != objCopy? ";</pre>
26
27
        cout << (objSmallerSizedStorage != objCopy ? "Yes" : "No") << end1;</pre>
28
        // Testing the stream insertion operator.
29
        cout << objSmallerSizedStorage << "Done!" << endl;</pre>
30
        return 0;
31
32
33
    int rangedRandom(int intLB, int intUB){
34
        return rand() % (intUB - intLB + 1) + intLB;
35
```