Cuprins

șiere comune	2
locarea dinamica a memoriei. Tipuri specifice	16
Laborator 1	
Laborator 2	19
ablouri	22
Laborator 3	22
ste liniare simplu înlănțuite. Stive și cozi	25
Laborator 4	25
Laborator 5	26

Fisiere comume

utils.h

```
#pragma once
    # include <initializer_list>
    # include <iostream>
3
4
    constexpr size_t getSize(std::initializer_list<double> 1) {
5
        size_t n = 0;
6
        auto it = 1.begin();
        auto end = 1.end();
8
        while (it++ != end) ++n;
        return n;
10
    }
11
12
    constexpr void getSize(std::initializer_list<</pre>
13
                                 std::initializer_list<double>> list,
                             int& s1, int& s2) {
15
        constexpr int invalid = -1;
16
        s1 = 0;
17
        s2 = invalid;
18
19
        for (const auto& 1: list) {
20
            int i = getSize(1);
21
            ++s1;
22
            if (s2 != invalid && i != s2) {
23
                 //std::cerr << i << "!=" << s2 << "\n";
24
                 throw std::logic_error("Fixed line size expected");
25
            }
26
            s2 = i;
27
28
        //std::cerr << "size: " << s1 << ", " << s2 << "\n";
29
30
    constexpr int MAX_SZ = 256;
31
    inline size_t readSize(const char* name, int a = 1, int b = MAX_SZ) {
32
        int res;
33
        do {
34
            std::cout << name << ": ";
35
            std::cin >> res;
36
        } while (res < a || res > b);
37
        return res;
38
   }
39
```

```
40
41
    inline void assert(bool cond, const char* msg) {
42
        if (!cond) throw std::logic_error(msg);
43
44
    template<typename T>
45
    inline T read(const char* name) {
46
        T res;
47
        std::cout << name << ": ";
48
        std::cin >> res;
        return res;
50
    }
51
```

matrix.h

```
#pragma once
1
    #include "utils.h"
2
3
    #include <iostream>
4
    # include <cmath>
5
    # include <cstring>
6
    # include <utility>
7
   struct Mat {
9
        int m, n;
10
        double *data;
11
        Mat() : m(0), n(0), data(nullptr) {}
12
        Mat(int m, int n) : m(m), n(n) { data = new double[m*n]; }
        //template<int m, int n>
        Mat(std::initializer_list<std::initializer_list<double>> list) {
15
            getSize(list, m, n);
16
            data = new double[m*n];
18
            auto it = begin();
            for (const auto& 1 : list) {
20
                for (const auto& v : 1) *(it++) = v;
21
            }
22
23
        Mat(Mat&& rhs) noexcept :
24
            m(rhs.m), n(rhs.n),
25
            data(std::exchange(rhs.data, nullptr)) {
26
            //rhs.n = rhs.m = 0;
27
        }
28
29
        Mat(const Mat& rhs) : Mat(rhs.m, rhs.n) {
30
            std::memcpy(data, rhs.data, sizeof(double) *m*n);
31
32
        Mat& operator=(const Mat& rhs) {
33
            setSize(rhs.m, rhs.n);
34
            std::memcpy(data, rhs.data, sizeof(double) *m*n);
35
```

```
return *this;
36
        }
37
        Mat& operator=(Mat&& rhs) noexcept {
38
            this->~Mat();
39
            m = rhs.m;
40
            n = rhs.n;
41
            data = std::exchange(rhs.data, nullptr);
42
43
            return *this;
        }
45
46
        ~Mat() {
            delete[] data;
48
        double& at(int i, int j) {
50
            assert(i < m && j < n, "out of range");</pre>
51
            return data[i *n +j];
        }
        double at(int i, int j) const { return data[i * n + j]; }
54
        double* begin() { return data; }
55
        const double* begin() const { return data; }
56
        double* end() { return data + m*n; }
        const double* end() const { return data + m*n; }
58
59
        void setSize(int m, int n) {
60
            if (this->m == m && this->m == n) return;
61
            this->~Mat();
62
            new (this) Mat(m, n);
63
        }
64
65
        static Mat read() {
66
            Mat res(readSize("m"), readSize("n"));
67
            for (auto& i : res)
68
                 std::cin >> i;
69
            return res;
70
71
        void print(const char* name) const {
72
            std::cout << name << " = Mat " << m << "x" << n << "{\n";}
73
74
            for (int i = 0; i < m; ++i) {
75
                 for (int j = 0; j < n; ++j) {
76
                     std::cout << at(i, j) << " ";
77
                 }
78
                 std::cout << "\n";
79
80
            std::cout << "}\n";
81
        }
        friend std::ostream& operator<<(std::ostream& s, Mat& m) {
84
            s \ll "Mat " \ll m.m \ll "x" \ll m.n \ll "{\n"};
```

```
for (int i = 0; i < m.m; ++i) {
86
                 for (int j = 0; j < m.n; ++j) {
87
                      s << m.at(i, j) << " ";
88
89
                 s \ll "\n";
90
             }
91
             return s<< "}\n";
92
         }
93
94
         double norm1() const { return normImpl<Type::Row>(n, m); }
95
         double normInf() const { return normImpl<Type::Col>(m, n); }
96
         double normF() const {
97
             double res = 0;
98
             for (int i = 0; i < m; ++i)
                 for (int j = 0; j < n; ++j)
100
                      res += at(i, j) * at(i, j);
102
             return std::sqrt(res);
103
         }
104
        bool isStrictlyRowDiagonallyDominant() const {
105
             return isStrictlyDiagonallyDominantImpl<Type::Row>();
106
         }
107
108
        bool isStrictlyColDiagonallyDominant() const {
109
             return isStrictlyDiagonallyDominantImpl<Type::Col>();
110
         }
111
112
    private:
113
114
         enum class Type { Row, Col };
115
         template<Type type>
116
         bool isStrictlyDiagonallyDominantImpl() const {
117
             assert(m == n, "Matrix must be square");
118
             for (int i = 0; i < m; ++i) {
119
                 double val = std::abs(at(i, i));
120
                 double sum = -val;
121
                 for (int j = 0; j < m; ++j) {
122
                      sum += std::abs(type == Type::Col? at(j, i): at(i, j));
123
                 }
124
                 if (sum >= val) {
125
                      std::cout << "(" << sum << "," << val << ")";
126
                      return false;
127
                 }
128
             }
129
             return true;
130
131
         template<Type type>
         double normImpl(int sz1, int sz2) const {
133
             double max = -1;
134
             for (int j = 0; j < sz1; ++j) {
```

```
double x = 0;
136
                 for (int i = 0; i < sz2; ++i) {
137
                      x += std::abs(type == Type::Col? at(j, i) : at(i, j));
138
139
                 if (x > max) max = x;
140
             }
141
             return max;
142
         }
143
    };
144
145
    Mat& add(const Mat& a, const Mat& b, Mat& res) {
146
         assert(a.m == b.m && a.n == b.n, "Sizes don't match, can't add");
147
        res.setSize(a.m, a.n);
148
         for (int i = 0; i < a.m; ++i)
             for (int j = 0; j < a.n; ++j)
                 res.at(i, j) = a.at(i, j) + b.at(i, j);
151
        return res;
    }
153
154
    Mat& mul(double a, const Mat& b, Mat& res) {
155
        res.setSize(b.m, b.n);
156
         for (int i = 0; i < res.m; ++i)
158
             for (int j = 0; j < res.n; ++j)
159
                 res.at(i, j) = a * b.at(i, j);
160
        return res;
161
    }
162
163
    Mat& neg(const Mat& a, Mat& res) {
164
        return mul(-1, a, res);
165
166
    Mat& sub(const Mat& a, const Mat& b, Mat& res) {
167
         return add(a, neg(b, res), res);
168
169
    Mat& mul(const Mat& a, const Mat& b, Mat& res) {
170
         assert(a.n == b.m, "Sizes don't match, can't multiply");
171
        res.setSize(a.m, b.n);
172
173
         for (int i = 0; i < res.m; ++i)
174
             for (int j = 0; j < res.n; ++j) {
175
                 res.at(i, j) = 0;
176
                 for (int k = 0; k < a.n; ++k)
177
                      res.at(i, j) += a.at(i, k) * b.at(k, j);
178
179
        return res;
180
181
    Mat& trans(const Mat& a, Mat& res) {
183
         assert(a.data != res.data, "Can't transpose inplace");
184
        res.setSize(a.n, a.m);
```

```
for (int i = 0; i < res.m; ++i)

for (int j = 0; j < res.n; ++j)

res.at(i, j) = a.at(j, i);

return res;

}
```

vector.h

```
#pragma once
1
    #include "utils.h"
2
3
    # include <iostream>
4
    # include <utility>
5
    # include <cmath>
6
   struct Vec {
8
        double *_begin, *_end;
9
10
        constexpr double* begin() { return _begin; }
11
        constexpr const double* begin() const { return _begin; }
12
13
        constexpr double* end() { return _end; }
14
        constexpr const double* end() const { return _end; }
15
16
        constexpr Vec() : _begin(nullptr), _end(nullptr) {}
        explicit Vec(size_t n) : _begin(new double[n]), _end(_begin+n) {}
18
        Vec(std::initializer_list<double> list) : Vec(getSize(list)) {
            auto it = _begin;
20
            for (const auto& v : list) *(it++) = v;
22
        Vec(const Vec& rhs) : Vec(rhs.size()) {
23
            auto it = _begin;
24
            for (const auto& v: rhs) *(it++) = v;
25
26
        Vec(Vec&& rhs) noexcept
27
             : _begin(std::exchange(rhs._begin, nullptr)),
28
            _end(std::exchange(rhs._end, nullptr)) {}
29
30
        Vec& operator=(const Vec& rhs) {
31
            if (size() != rhs.size()) {
32
                this->~Vec();
33
                new (this) Vec(rhs.size());
34
35
            auto it = _begin;
36
            for (const auto& v : rhs) *(it++) = v;
37
            return *this;
38
39
        Vec& operator=(Vec&& rhs) noexcept {
40
            this->~Vec();
41
```

```
_begin = std::exchange(rhs._begin, nullptr);
42
            _end = std::exchange(rhs._end, nullptr);
43
            return *this;
44
45
46
        ~Vec() { delete[] _begin; }
47
48
        constexpr size_t size() const { return _end - _begin; }
49
50
        constexpr double& operator[](size_t i) { return _begin[i]; }
51
        constexpr double operator[](size_t i) const { return _begin[i]; }
52
53
        void setSize(size_t n) {
54
            if (size() == n) return;
            *this = Vec(n);
        }
57
        friend std::ostream& operator<<(std::ostream& s, const Vec& v) {
            s << "(";
60
            double* it = v._begin;
            for (double* end = v._end - 1; it < end; ++it)
62
                 s << *it << ", ";
64
            if (it < v._end) s << *it;
65
66
            return s << ")";
67
68
        static Vec read() {
69
            Vec res(readSize("n"));
70
            for (auto& v: res) std::cin >> v;
71
            return res;
72
73
        double norm() const;
74
   };
75
   void assertSizes(const Vec& a, const Vec& b) {
76
        assert(a.size() == b.size(), "Sizes don't match");
77
   }
78
79
   Vec& add(const Vec& a, const Vec& b, Vec& res) {
80
        assertSizes(a, b);
81
        res.setSize(a.size());
82
        auto aIt = a.begin();
83
        auto bIt = b.begin();
84
        for (auto& v : res) v = *(aIt++) + *(bIt++);
85
        return res;
86
87
   Vec& mul(double a, const Vec& b, Vec& res) {
89
        res.setSize(b.size());
90
        auto it = b.begin();
91
```

```
for (auto& v: res) v = a * (*(it++));
92
        return res;
93
    }
94
95
    Vec& neg(const Vec& b, Vec& res) { return mul(-1, b, res); }
96
97
    Vec& sub(const Vec& a, const Vec& b, Vec& res) {
98
         return add(a, neg(b, res), res);
99
100
    double dot(const Vec& a, const Vec& b) {
101
        double res = 0;
102
        assertSizes(a, b);
        auto bIt = b.begin();
104
        for (auto& v: a) res += v * (*(bIt++));
        return res;
106
    double norm(const Vec& a) {
108
        return std::sqrt(dot(a, a));
109
    }
110
    double Vec::norm() const {
111
        return ::norm(*this);
112
    }
113
```

list.h

```
#include "utils.h"
1
2
    # include <type_traits>
3
    # include <utility>
5
6
    // We should never check for (bot == nullptr),
7
    // so we don't update it when the list becomes empty.
8
    template<typename T>
9
    struct List {
10
        struct Node {
11
            T val;
12
            Node* next;
13
        };
14
        Node* top;
15
        Node* bot;
16
        constexpr List(Node* top = nullptr, Node* bot = nullptr)
17
             : top(top), bot(bot) {}
18
        List(const List&) = delete;
19
        List& operator=(const List&) = delete;
20
        ~List() {
21
            while (top) {
22
                 Node* n = top;
23
                 top = top->next;
24
                 delete n;
25
```

```
}
26
27
        List(List&& rhs) noexcept
28
             : top(std::exchange(rhs.top, nullptr)),
29
               bot(std::exchange(rhs.bot, nullptr)) {}
30
        List& operator=(List&& rhs) noexcept {
31
             this->~List();
32
             this->top = std::exchange(rhs.top, nullptr);
33
             this->bot = std::exchange(rhs.bot, nullptr);
34
        }
35
36
        T& first() { return top->val; }
37
        T& last() { return bot->val; }
38
        T pop_front() {
             assert(top, "Empty list");
40
            Node* n = top;
41
            top = top->next;
42
             T res = n->val;
            delete n;
44
             return res;
45
        }
46
47
        void push_front(T val) {
48
             Node *n = new Node{ val, top };
49
             top = n;
50
        }
51
        void push_back(T val) {
52
             Node *n = new Node{ val, nullptr };
53
             if (top == nullptr) {
54
                 top = n;
55
56
             else {
57
                 bot->next = n;
58
59
60
             bot = n;
61
62
        bool empty() const { return top == nullptr; }
63
64
        bool operator== (const List& rhs) {
65
             for (auto it = top, rit = rhs.top;
66
                 it != nullptr;
67
                 it = it->next, rit = rit->next) {
68
                 if (it->val != rit->val) return false;
69
70
71
            return true;
        }
        static List read(const char* msg) {
            List q;
```

```
std::cout << msg << ":\n";
76
             int len1 = readSize("n");
77
             for (int i = 0; i < len1; ++i) {
78
                  Ts;
79
                  std::cin >> s;
80
                  q.push_front(s);
81
             }
82
             return std::move(q);
83
         }
84
85
         friend std::ostream& operator<<(std::ostream& s, const List& st) {
86
             if constexpr (std::is_same_v<T, char>) {
87
                  const Node* it = st.top;
88
                  while (it != nullptr) {
                      s << it->val;
90
                      it = it->next;
91
92
                  return s;
             }
94
             else {
95
                  if (st.empty()) return s << "{}";
96
                  s << "{";
97
                  const Node* it = st.top;
98
                  while (it->next != nullptr) {
99
                      s << it->val << ", ";
100
                      it = it->next;
101
                  }
102
103
                  return s << it->val << "}";
104
             }
105
         }
106
107
         void remove(T& val) {
108
             remove_if([&] (T& t) { return t ==val; });
109
110
         // removes all elements that satisfy p
111
         template<class P>
112
         void remove_if(P p) {
113
             apply_on(p, [] (Node* n) { delete n; } );
114
115
         //applies\ f() on all nodes that satisfy the predicate p()
116
         template < class P, class F>
117
         void apply_on(P p, F f) {
118
             for (;;) {
119
                  auto* n = top;
120
                  if (!n) {
121
                      return;
122
                  }
123
                  if (!p(n->val)) break;
124
                  top = n->next;
125
```

```
f(n);
126
             }
127
             auto* prev = top;
128
             auto* it = prev->next;
129
130
             while (it) {
131
                  if (p(it->val)) {
132
                      prev->next = it->next;
133
                      if (prev->next == nullptr) {
134
                           bot = prev;
135
                      }
136
                      f(it);
137
                      it = prev;
138
                  } else {
139
                      prev = it;
140
                      it = it->next;
                  }
142
             }
143
         }
144
    };
145
146
     // keeps in l all the elements that don't satisfy the predicate p
147
     // and returs pair of:
148
     // - a reference to the original list
149
     // - a list containing the elements that satisfy p
150
    template<typename T, typename P>
151
    constexpr auto partition_split(List<T>& 1, P p) {
152
         struct res_t {
153
             List<T>& notSatisfying;
154
             List<T> satisfying;
155
         } res = { 1, {} };
156
         List<T>& sat = res.satisfying;
157
         auto insertNode = [&sat](auto* n) {
158
             n->next = sat.top;
159
             sat.top = n;
160
             if (sat.bot == nullptr) {
161
                  sat.bot = n;
162
             }
163
         };
164
         1.apply_on(p, insertNode);
165
         return res;
166
    }
167
```

stack.h

```
# include "utils.h"

template<typename T>
struct Stack {
    struct Node {
```

```
T val;
6
             Node* next;
7
        };
8
9
        Stack(const Stack&) = delete;
10
        Stack& operator=(const Stack&) = delete;
11
        Node* top;
12
        Stack(Node* top = nullptr) : top(top) {}
13
        ~Stack() {
14
             while (top) {
15
                 Node* n = top;
16
                 top = top->next;
17
                 delete n;
18
             }
        }
20
        void push(T val) {
             Node *n = new Node{ val, top };
22
             top = n;
23
24
        void quickPop() {
25
             Node* n = top;
26
             top = top->next;
27
             delete n;
28
29
        T pop() {
30
             assert(top, "Empty stack");
31
             Node* n = top;
32
             top = top->next;
33
             T res = n->val;
34
             delete n;
35
             return res;
36
        }
37
        bool empty() const { return top == nullptr; }
38
        friend std::ostream& operator<<(std::ostream& s, const Stack& st) {
39
             if (st.empty()) return s << "{}";
40
             s << "{";
41
             const Node* it = st.top;
42
             while (it->next != nullptr) {
43
                 s << it->val << ", ";
44
                 it = it->next;
45
             }
46
47
             return s << it->val << "}";
48
        }
49
    };
50
```

queue.h

```
# include "utils.h"
```

```
template < class T>
3
    struct Queue {
4
        struct Node {
5
             T val;
6
             Node* next;
7
        };
8
        Node* top;
9
        Node* bot;
10
        Queue(Node* top = nullptr, Node* bot = nullptr) : top(top), bot(bot) {}
11
        /*
12
        Queue(const Queue&) = delete;
13
        Queue& operator=(const Queue&) = delete;
14
         ~Queue() {
15
             while (top) {
16
                 Node* n = top;
17
                 top = top->next;
18
                 delete n;
19
             }
20
             }*/
21
        void push(T val) {
^{22}
             Node *n = new Node{ val, nullptr };
23
             if (bot == nullptr) {
24
                 top = n;
25
             }
26
             else {
27
                 bot->next = n;
28
29
             bot = n;
30
        }
31
32
        T pop() {
33
             assert(top, "Empty queue");
34
             Node* old = top;
35
             T res = old->val;
36
             top = top->next;
37
             if (top == nullptr)
38
                 bot = nullptr;
39
             delete old;
40
             return res;
41
        }
42
        bool empty() const { return top == nullptr; }
43
        static Queue read(const char* msg) {
44
             Queue q;
45
             std::cout << msg << ":\n";
46
             int len1 = readSize("n");
47
             for (int i = 0; i < len1; ++i) {
48
                 Ts;
                 std::cin >> s;
                 q.push(s);
51
             }
```

```
return std::move(q);
53
      }
54
55
      56
          if (st.empty()) return s << "{}";</pre>
57
          s << "{";
58
          const Node* it = st.top;
59
          while (it->next != nullptr) {
60
             s << it->val << ", ";
61
             it = it->next;
62
          }
63
64
          return s << it->val << "}";
65
      }
66
   };
67
```

Alocarea dinamică a memoriei. Tipuri specifice.

Laborator 1

16. Scrieți funcții pentru implemetarea operațiilor specifice pe matrice de numere reale cu m linii și n coloane: suma, diferența și produsul al două matrice, produsul dintre o matrice și un scalar real, transpusa unei matrice, norme matriceale specifice¹, citirea de la tastatură a componentelor unei matrice, afișarea componentelor matricei. Pentru cazul particular al unei matrice patratice de ordin n, să se testeze dacă aceasta satisface criteriul de dominanță pe linii² sau pe coloane³. Se vor folosi tablouri bidimensionale alocate static.

```
#include "utils.h"
1
2
    # include <iostream>
3
    # include <cmath>
4
    struct Mat {
6
        double data[MAX_SZ][MAX_SZ] {};
7
        int m, n;
9
        Mat() : m(0), n(0) \{ \}
10
        Mat(int m, int n) : m(m), n(n) {}
11
12
        static Mat read() {
13
             Mat res(readSize("m"), readSize("n"));
14
15
             for (int i = 0; i < res.m; ++i)
16
                 for (int j = 0; j < res.n; ++j)
17
                      std::cin >> res.data[i][j];
18
             return res;
19
20
        void setSize(int m, int n) {
21
             this->m = m;
22
             this->n = n;
23
        }
24
```

¹Dacă $A \in \mathcal{M}_{m \times n}(\mathbb{R})$, atunci $||A||_1 = \max_{1 \le j \le n} \sum_{i=1}^m |a_{ij}|, ||A||_{\infty} = \max_{1 \le i \le m} \sum_{j=1}^n |a_{ij}|, ||A||_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n a_{ij}^2}.$

 $^{^2}A \in \mathcal{M}_n(\mathbb{R})$ este strict diagonal dominantă pe linii dacă $|a_{ii}| > \sum_{\substack{j=1 \ j \neq i}}^n |a_{ij}|$, pentru orice i=1,...,n.

 $^{{}^3}A \in \mathcal{M}_n(\mathbb{R})$ este strict diagonal dominantă pe colonane dacă $|a_{jj}| > \sum_{\substack{i=1 \ i \neq j}}^n |a_{ij}|$, pentru orice j = 1, ..., n.

```
double& at(int i, int j) { return data[i][j]; }
25
        double at(int i, int j) const { return data[i][j]; }
26
27
        void print(const char* name) const {
28
            std::cout << name << " = Mat " << m << "x" << n << "{\n";}
29
            for (int i = 0; i < m; ++i) {
30
                for (int j = 0; j < n; ++ j)
31
                     std::cout << at(i, j) << " ";
32
                std::cout << "\n";
33
            }
34
            std::cout << "}\n";
35
36
   private:
37
        enum class Type { Row, Col };
        template<Type type>
39
        bool isStrictlyDiagonallyDominantImpl() const {
40
            assert(m == n, "Matrix must be square");
41
            for (int i = 0; i < m; ++i) {
                double val = std::abs(at(i, i));
43
                double sum = -val;
44
                for (int j = 0; j < m; ++j)
45
                     sum += std::abs(type == Type::Col ? at(j, i) : at(i, j));
46
                if (sum >= val) return false;
47
            }
48
            return true;
49
        }
50
51
        template<Type type>
52
        double normImpl(int sz1, int sz2) const {
53
            double max = -1;
            for (int j = 0; j < sz1; ++j) {
55
                double x = 0;
56
                for (int i = 0; i < sz2; ++i)
57
                     x += std::abs(type == Type::Col ? at(j, i) : at(i, j));
58
                if (x > max) max = x;
59
            }
60
            return max;
61
        }
62
   public:
63
        double norm1() const { return normImpl<Type::Row>(n, m); }
64
        double normInf() const { return normImpl<Type::Col>(m, n); }
65
        double normF() const {
66
            double res = 0;
67
            for (int i = 0; i < m; ++i)
68
                for (int j = 0; j < n; ++j)
69
                     res += at(i, j) * at(i, j);
70
            return std::sqrt(res);
72
73
        bool isStrictlyRowDiagonallyDominant() const {
```

```
return isStrictlyDiagonallyDominantImpl<Type::Row>();
75
         }
76
         bool isStrictlyColDiagonallyDominant() const {
77
             return isStrictlyDiagonallyDominantImpl<Type::Col>();
78
         }
79
    };
80
81
    Mat& add(const Mat& a, const Mat& b, Mat& res) {
82
         assert(a.m == b.m && a.n == b.n, "Sizes don't match, can't add");
83
         res.setSize(a.m, a.n);
84
         for (int i = 0; i < a.m; ++i)
85
             for (int j = 0; j < a.n; ++j)
86
                 res.at(i, j) = a.at(i, j) + b.at(i, j);
87
        return res;
89
    Mat& mul(double a, const Mat& b, Mat& res) {
90
        res.setSize(b.m, b.n);
91
        for (int i = 0; i < res.m; ++i)
93
             for (int j = 0; j < res.n; ++j)
94
                 res.at(i, j) = a * b.at(i, j);
95
        return res;
96
97
    Mat& neg(const Mat& a, Mat& res) { return mul(-1, a, res); }
98
    Mat& sub(const Mat& a, const Mat& b, Mat& res) {
99
         return add(a, neg(b, res), res);
100
101
    Mat& mul(const Mat& a, const Mat& b, Mat& res) {
102
         assert(a.n == b.m, "Sizes don't match, can't multiply");
103
        res.setSize(a.m, b.n);
104
105
         for (int i = 0; i < res.m; ++i)
106
             for (int j = 0; j < res.n; ++j) {
107
                 res.at(i, j) = 0;
108
                 for (int k = 0; k < a.n; ++k)
109
                      res.at(i, j) += a.at(i, k) * b.at(k, j);
110
111
        return res;
112
    }
113
    Mat& trans(const Mat& a, Mat& res) {
114
         assert(a.data != res.data, "Can't calculate the transpose inplace");
115
         res.setSize(a.n, a.m);
116
117
         for (int i = 0; i < res.m; ++i)
118
             for (int j = 0; j < res.n; ++j)
119
                 res.at(i, j) = a.at(j, i);
120
        return res;
121
    }
122
```

Laborator 2

18. Scrieți funcții pentru implementarea operațiilor specifice pe vectori din \mathbb{R}^n : suma, diferența și produsul scalar al doi vectori, produsul dintre un vector și un scalar real, negativarea unui vector, norma euclidiană a unui vector, citirea de la tastură a celor n componente ale unui vector, afișarea componentelor vectorului sub forma unui n-uplu de elemente. Se vor folosi tablouri unidimensionale alocate dinamic.

```
#include "utils.h"
1
2
    #include <iostream>
3
    #include <utility>
    # include <cmath>
5
    struct Vec {
7
        double *_begin, *_end;
8
        constexpr double* begin() { return _begin; }
10
        constexpr const double* begin() const { return _begin; }
12
        constexpr double* end() { return _end; }
14
        constexpr const double* end() const { return _end; }
15
16
        constexpr Vec() : _begin(nullptr), _end(nullptr) {}
        explicit Vec(size_t n) : _begin(new double[n]), _end(_begin+n) {}
18
        Vec(std::initializer_list<double> list) : Vec(getSize(list)) {
19
            auto it = _begin;
20
            for (const auto& v : list) *(it++) = v;
21
22
        Vec(const Vec&) = delete;
23
        Vec(Vec&& rhs) noexcept
24
            : _begin(std::exchange(rhs._begin, nullptr)),
25
            _end(std::exchange(rhs._end, nullptr)) {}
26
27
        Vec& operator=(const Vec&) = delete;
28
        Vec& operator=(Vec&& rhs) noexcept {
29
            this->~Vec();
30
            _begin = std::exchange(rhs._begin, nullptr);
31
            _end = std::exchange(rhs._end, nullptr);
32
            return *this;
33
        }
34
35
        ~Vec() { delete[] _begin; }
36
37
        constexpr size_t size() const { return _end - _begin; }
38
39
        constexpr double& operator[](size_t i) { return _begin[i]; }
40
        constexpr double operator[](size_t i) const { return _begin[i]; }
41
        void setSize(size_t n) {
43
```

```
if (size() == n) return;
44
            *this = Vec(n);
45
        }
46
47
        friend std::ostream& operator<<(std::ostream& s, const Vec& v) {
48
            s << "(";
49
            double* it = v._begin;
50
            for (double* end = v._end - 1; it < end; ++it)
51
                 s << *it << ", ";
52
53
            if (it < v._end) s << *it;
54
55
            return s << ")";
56
        }
        static Vec read() {
            Vec res(readSize("n"));
            for (auto& v: res) std::cin >> v;
60
            return res;
62
        double norm() const;
63
    };
64
    void assertSizes(const Vec& a, const Vec& b) {
65
        assert(a.size() == b.size(), "Sizes don't match");
66
    }
67
68
    Vec& add(const Vec& a, const Vec& b, Vec& res) {
69
        assertSizes(a, b);
70
        res.setSize(a.size());
71
        auto aIt = a.begin();
72
        auto bIt = b.begin();
73
        for (auto& v : res) v = *(aIt++) + *(bIt++);
74
        return res;
75
76
77
    Vec& mul(double a, const Vec& b, Vec& res) {
78
        res.setSize(b.size());
79
        auto it = b.begin();
80
        for (auto& v: res) v = a * (*(it++));
81
        return res;
82
    }
83
84
    Vec& neg(const Vec& b, Vec& res) { return mul(-1, b, res); }
85
86
    Vec& sub(const Vec& a, const Vec& b, Vec& res) {
87
        return add(a, neg(b, res), res);
88
89
    double dot(const Vec& a, const Vec& b) {
        double res = 0;
91
        assertSizes(a, b);
92
        auto bIt = b.begin();
93
```

```
for (auto& v : a) res += v * (*(bIt++));
94
        return res;
95
96
    double norm(const Vec& a) {
97
        return std::sqrt(dot(a, a));
98
99
    double Vec::norm() const {
100
        return ::norm(*this);
101
    }
102
```

Tablouri

Laborator 3

7. Folosind structurile de date VECTOR și MATRICE definite la curs și funcțiile necesare, rezolvați următorul sistem algebric liniar cu n ecuații și n necunoscute folosind metoda lui Gauß de eliminare.

```
\begin{cases}
2x_1 - x_2 = 1 \\
-x_1 + 2x_2 - x_3 = 1 \\
-x_2 + 2x_3 - x_4 = 1 \\
\dots \\
-x_{n-2} + 2x_{n-1} - x_n = 1 \\
-x_{n-1} + 2x_n = 1, \quad n \in \mathbb{N}, 2 \le n \le 50
\end{cases}
```

```
#include "utils.h"
1
2
    #include "vector.h"
3
    # include "matrix.h"
4
5
    # include <iostream>
6
    # include <cmath>
7
8
    constexpr double eps = 1e-7;
9
10
    Vec& mul(const Mat& m, const Vec& v, Vec& res) {
11
        assert(v.begin() != res.begin(), "Can't multiply inplace");
12
        assert(v.size() == size_t(m.n), "Sizes don't match");
13
        res.setSize(m.m);
14
15
16
        for (size_t i = 0; i < res.size(); ++i) {</pre>
17
             res[i] = 0;
18
             for (size_t k = 0; k < v.size(); ++k)</pre>
19
                 res[i] += m.at(i, k) * v[k];
20
21
        return res;
22
    }
23
24
    // A * X = b
25
    struct System {
26
        Mat A;
```

```
Vec b;
28
29
        System(int n, int m) : A(n, m), b(n) {}
30
31
        System(std::initializer_list<std::initializer_list<double>> A,
32
                std::initializer_list<double> b) : A(A), b(b) {
33
                assert(std::size_t(this->A.m) == this->b.size(),
34
                       "sizes don't match");
35
        }
36
37
        friend std::ostream& operator<<(std::ostream& s, const System& sys) {
38
            s << "System " << sys.A.m << "x" << sys.A.n << ": \n";
39
            auto& A = sys.A;
40
            for (int i = 0; i < A.m; ++i) {
                 s << "{";
42
                 for (int j = 0; j < A.n; ++j) {
43
                     //showpos shows a '+' in front of positive numbers
44
                     if (std::abs(A.at(i, j)) > eps)
                         s << std::showpos << A.at(i, j)
46
                            << std::noshowpos << "*x" << (j+1) << " ";
47
                 }
48
                 s<< "= " << sys.b[i] << "\n";
49
            }
50
51
            return s;
52
        }
53
54
        // L_i += f * L_j
55
        void addLines(int i, double f, int j) {
56
            for (int k = 0; k < A.n; ++k) {
57
                 A.at(i, k) += f * A.at(j, k);
58
59
            b[i] += f * b[j];
60
        }
61
62
        // L_i *= f
63
        void multiplyLine(int i, double f) {
64
            for (int k = 0; k < A.n; ++k) {
65
                 A.at(i, k) *= f;
66
67
            b[i] *= f;
68
        }
69
70
        Vec solveTriangulated() {
71
            for (int i = A.m-1; i > 0; --i) {
72
                 addLines(i-1, -A.at(i-1, i), i);
73
                 multiplyLine(i, 1 / A.at(i, i));
            }
75
            return b;
76
        }
77
```

```
78
         bool checkSolution(const Vec& x) const {
79
             Vec r;
80
             mul(A, x, r);
81
             sub(b, r, r);
82
             for (auto& v : r) {
83
                  if (std::abs(v) > eps) return false;
84
             }
85
             return true;
86
         }
87
88
         static Vec solveCustom(int n) {
89
             System s(n, n);
90
             for (auto& v : s.A) v = 0;
             s.A.at(0,0) = 2;
92
             s.A.at(0,1) = -1;
             s.b[0] = 1;
94
             for (int i = 1; i < n - 1; ++i) {
                  s.b[i] = 1;
96
97
                  s.A.at(i,i-1) = -1;
98
                  s.A.at(i,i) = 2;
                  s.A.at(i,i+1) = -1;
100
             }
101
             s.b[n-1] = 1;
102
             s.A.at(n-1, n-2) = -1;
103
             s.A.at(n-1, n-1) = 2;
104
             s.customTriangulate();
105
             return s.solveTriangulated();
106
107
         void customTriangulate() {
108
             multiplyLine(0, 1 / A.at(0, 0));
109
             for (int i = 1; i < A.m; ++i) {
110
                  addLines(i, 1, i-1);
111
                  multiplyLine(i, 1 / A.at(i, i));
112
             }
113
         }
114
    };
115
116
     int main() {
117
         try {
118
             int n = readSize("n", 2, 51);
119
             std::cout << "x = "<< System::solveCustom(n) << "\n";
120
         } catch (std::exception& e) {
121
             std::cerr << "Error" << e.what() << "\n";
             return 1;
         }
         return 0;
125
    }
```

Liste liniare simplu înlănțuite Stive și cozi

Laborator 4

5. Se citește un text de la tastatura (poate conține orice caracter, inclusiv spații) și se încarcă în două stive: o stivă va conține doar litere mici, iar cealaltă doar litere mari. Se citește de la tastatură o vocală a alfabetului englez (literă mare sau mică). Ștergeți stiva corespunzătore până la întâlnirea vocalei citite.

```
#include "utils.h"
1
    #include "stack.h"
2
3
    #include <iostream>
4
    # include <cstdlib>
5
6
    bool isVowel(char c) {
7
        c = tolower(c);
8
        return c == 'a' || c== 'e' || c== 'i' || c=='o' || c=='u';
9
    }
10
11
    int main() {
12
        std::string str;
13
        std::cout << "str: ";
14
        std::getline(std::cin, str);
15
16
        char v;
17
        do {
             v = read<char>("vowel");
19
        } while (!isVowel(v));
20
21
        Stack<char> lower;
22
        Stack<char> upper;
23
        auto printStacks = [&] (const char* s) {
24
                                  std::cout << s;</pre>
25
                                  std::cout << "lower:" << lower << "\n";
26
                                  std::cout << "upper:" << upper << "\n";
27
                             };
28
29
        printStacks("Before:\n");
30
        for (auto& c : str) {
31
             if (islower(c)) lower.push(c);
32
             else if (isupper(c)) upper.push(c);
33
```

```
}
34
35
        printStacks("After Adding:\n");
36
        Stack<char>& stack = isupper(v)? upper: lower;
37
        while (!stack.empty()) {
38
             if (char c = stack.pop(); c == v) {
39
                 break;
40
             }
41
        }
42
        printStacks("Result:\n");
43
        return 0;
44
    }
```

Laborator 5

11. Creați o listă liniară simplu înlănțuită în nodurile căreia sunt memorate numere naturale. Sepratați numerele naturale memorate în listă, în două liste, una corespunzătore numerelor pare și cealaltă, numerelor impare. Afișați cele două liste. Ștergeți din lista numerelor pare, o valoare pară x, citită de la tastatură, ori de câte ori aprare în listă.

```
#include "utils.h"
1
    #include "list.h"
2
3
    #include <iostream>
4
5
    int main() {
6
        auto nums = List<int>::read("numbers");
7
8
        auto [evens, odds] = partition_split(nums, [](int n) {return n % 2;});
9
        std::cout << "odd: " << odds << "\n";
10
        std::cout << "even: " << evens << "\n";
11
12
        int x;
13
        do {
14
             std::cout << "x (must be even): ";</pre>
15
             std::cin >> x;
16
        } while (x % 2);
17
18
        std::cout << "removing...\n";</pre>
19
        evens.remove(x);
20
21
        std::cout << "even: " << evens << "\n";
22
        return 0;
    }
24
```

17. Modelați printr-o LLSI un stoc de produse caracterizate prin: denumire, unitate de măsură, cantitate și preț unitar. Implementați principalele operații pe stoc: crearea stocului, introducerea unui produse nou, eliminarea unui produs in cazul în care acesta a fost vândut în întregime, modificare informații despre un produs (de exemplu, modificarea cantitații unui produs, în cazul vânzării), calculul valorii stocului la un moment dat, listare stoc.

include <iostream>