

Минобрнауки России

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«САРАТОВСКИЙ национальный исследовательский ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра информатики и программирования

ОТЧЁТ

**Информатика и программирование**

студента 1 курса 141 группы   
направления 02.03.03 «Математическое обеспечение и администрирование  
информационных систем»  
факультета компьютерных наук и информационных технологий

Шарова Кирилла Владимировича

Саратов 2022

**ОГЛАВЛЕНИЕ**

[**УПРАЖНЕНИЯ** 3](#_Toc96002976)

[РАЗДЕЛ 1. СТРОКИ 3](#_Toc96002977)

[РАЗДЕЛ 2. ФАЙЛЫ 4](#_Toc96002978)

[РАЗДЕЛ 3. РЕКУРСИВНЫЕ ФУНКЦИИ. ПЕРЕГРУЗКА ФУНКЦИЙ И ИСПОЛЬЗОВАНИЕ ШАБЛОНОВ 5](#_Toc96002979)

[РАЗДЕЛ 4. КЛАССЫ 10](#_Toc96002980)

[РАЗДЕЛ 5. НАСЛЕДОВАНИЕ 11](#_Toc96002981)

[РАЗДЕЛ 6. СПИСКИ 14](#_Toc96002982)

[РАЗДЕЛ 7. СТЕК 17](#_Toc96002983)

[РАЗДЕЛ 8. ОЧЕРЕДЬ 18](#_Toc96002984)

[РАЗДЕЛ 9. VECTOR 19](#_Toc96002985)

[РАЗДЕЛ 10. СОРТИРОВКИ 20](#_Toc96002986)

[РАЗДЕЛ 11. АЛГОРИТМЫ 21](#_Toc96002987)

[РАЗДЕЛ 12. SET 22](#_Toc96002988)

[РАЗДЕЛ 13. MAP 23](#_Toc96002989)

# **УПРАЖНЕНИЯ**

## РАЗДЕЛ 1. СТРОКИ

1. Преобразование символов в числа.  
   Дан текст, содержащий целые числа. Вывести на экран наименьшее из имеющихся чисел.

|  |
| --- |
| Решение: |
| #include <iostream> |
| #include <string> |
| using namespace std; |
| int main() { |
| string S, SNum = ""; |
| int minZ; |
| bool flag = 0; |
| getline(cin, S); |
| for (int i = 0; i < S.length(); i++) { |
| if (isdigit(S[i]) || S[i] == '-' && isdigit(S[i + 1])) { |
| SNum += S[i]; |
| } |
| else if (SNum != "") { |
| int n = atoi(SNum.c\_str()); |
| if (!flag || n < minZ) { |
| minZ = n; |
| flag = 1; |
| } |
| SNum = ""; |
| } |
| } |
| cout << minZ; |
| } |

**Таблица входных/выходных данных:**

|  |  |
| --- | --- |
| **Input** | **Output** |
|  |  |
|  |  |

## РАЗДЕЛ 2. ФАЙЛЫ

1. Сложные действия со строками. Дано осмысленное текстовое сообщение (т.е. алфавитно-цифровая информация, разделенная пробелами и знаками препинания, в конце которого ставится точка).  
   Вывести на экран все слова сообщения, состоящие из *n* букв.

|  |
| --- |
| Решение: |
| #include <iostream> |
| #include <fstream> |
| #include <string> |
| using namespace std; |
| int main() { |
| double a, b; |
| int N; |
| cin >> N; |
| ofstream fout("infile.dat", ios::binary); |
| for (int i = 0; i < N; i++) { |
| cin >> a; |
| fout.write((char\*)&a, sizeof(double)); |
| } |
| fout.close(); |
| ifstream fin("infile.dat", ios::binary); |
| while (fin.read((char \*)&a, sizeof(double))) |
| if (a < 0) cout << a << ' '; |
| fin.close(); |
| } |

**Таблица входных/выходных данных:**

|  |  |
| --- | --- |
| **Input** | **Output** |
|  |  |
|  |  |
|  |  |

## РАЗДЕЛ 3. РЕКУРСИВНЫЕ ФУНКЦИИ. ПЕРЕГРУЗКА ФУНКЦИЙ И ИСПОЛЬЗОВАНИЕ ШАБЛОНОВ

1. Разработка нерекурсивных функций.   
   Разработать функцию, которая для заданного натурального числа *N* возвращает количество его делителей. С помощью данной функции для заданного числа *A* вывести на экран следующее по отношению к нему число, имеющее столько же делителей, сколько и число A.

|  |
| --- |
| Решение: |
| #include <iostream> |
| using namespace std; |
| int delamount(unsigned int N) { |
| int count = 0; |
| if (N > 1) { |
| for (int i = 1; i <= N / 2; i++) |
| if (N % i == 0) |
| count++; |
| return count + 1; //+ само число |
| } |
| else if (N == 0) |
| return 0; |
| else return 1; |
| } |
| int main() { |
| int A; |
| cin >> A; |
| for (int i = A + 1;;i++) |
| if (delamount(i) == delamount(A)) { |
| cout << i; |
| break; |
| } |
| } |

**Таблица входных/выходных данных:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 10 | 14 |
| 20 | 28 |

1. Разработать рекурсивную функцию, возвращающую значение функции .

|  |
| --- |
| Решение: |
| #include <iostream> |
| using namespace std; |
| double rec(int N, int k = 1) { |
| if (k < N) |
| return sqrt(k + rec(N, k + 1)); |
| else return sqrt(N); |
| } |
| double F(int N) { |
| return N / rec(N); |
| } |
| int main() { |
| cout << F(2) << fixed; |
| } |

**Таблица входных/выходных данных:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 | 1.28719 |
|  |  |

1. Разработать рекурсивную функцию, не возвращающую значений. Дано натуральное число *n*. Разработать рекурсивную функцию для вывода на экран следующей последовательности чисел:

|  |
| --- |
| 1 2 2 3 3 3 … n n n … n |

|  |
| --- |
| Решение: |
| #include <iostream> |
| using namespace std; |
| void print(int N) { |
| if (N > 0) |
| print(N - 1); |
| for (int i = 0; i < N; i++, cout << N << '\t'); |
| cout << endl; |
| } |
| int main() { |
| int n; |
| cin >> n; |
| print(n); |
| } |

**Таблица входных/выходных данных:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3 | 1  2 2  3 3 3 |
| 5 | 1  2 2  3 3 3  4 4 4 4  5 5 5 5 5 |

## 

1. Используя механизм перегрузки функций, разработайте две версии функции *F*, заголовки которых выглядят следующим образом:  
   1) float F(float x);  
   2) void F(float x, float &y);  
   Продемонстрируйте работу данных функций на примерах.

**Решение:**

|  |
| --- |
| #include <iostream> |
| using namespace std; |
| float F(float x) { |
| if (x == 1 || x == -1) |
| return 1; |
| else if (x >= 0) |
| return -1 / (1 - x); |
| else return 1 / (1 + x); |
| } |
| void F(float x, float &y) { |
| if (x == 1 || x == -1) |
| y = 1; |
| else if (x >= 0) |
| y = -1 / (1 - x); |
| else y = 1 / (1 + x); |
| } |
| int main() { |
| float res; |
| cout << F(40) << endl; |
| F(40, res); |
| cout << res; |
| } |

**Таблица входных/выходных данных:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 40 | 0.025641  0.025641 |
|  |  |

1. Использование функций-шаблонов: для работы с двумерными массивами арифметических типов данных разработать шаблоны ввода и вывода массива, а также шаблон для решения основной задачи. Подсчитать количество элементов, не попадающих в заданный интервал.

|  |
| --- |
| Решение: |
| #include <iostream> |
| using namespace std; |
| template <typename AN> void arrayin2D(AN& Array, int X, int Y) { //ВВОД МАССИВА |
| for (int i = 0; i < X; i++) { |
| for (int j = 0; j < Y; j++) |
| cin >> Array[i][j]; |
| } |
| } |
| template <typename AN> void arrayout2D(AN &Array, int X, int Y) { //ВЫВОД МАССИВА |
| for (int i = 0; i < X; i++, cout << endl) |
| for (int j = 0; j < Y; j++, cout << '\t') |
| cout << Array[i][j]; |
| } |
| template <typename AN, typename BN> int bi2D(AN& Array, int X, int Y, BN A, BN B) { //beyond interval |
| BN temp; |
| int k = 0; |
| if (A > B) { //для универсальности, чтобы всегда A было больше B |
| temp = B; |
| B = A; |
| A = temp; |
| } |
| for (int i = 0; i < X; i++) |
| for (int j = 0; j < Y; j++) |
| if (Array[i][j] <= A || Array[i][j] >= B) |
| k++; |
| return k; |
| } |
| int main() { |
| int N, M; |
| cin >> N >> M; |
| int\*\* Arr = new int\* [N]; |
| int A, B; |
| for (int i = 0; i < N; i++) |
| Arr[i] = new int[M]; |
| arrayin2D(Arr, N, M); |
| arrayout2D(Arr, N, M); |
| cin >> A >> B; |
| cout << bi2D(Arr, N, M, A, B); |
| } |

**Таблица входных/выходных данных:**

|  |  |
| --- | --- |
| **Input** | **Output** |
|  |  |
|  |  |

## РАЗДЕЛ 4. КЛАСС

1. Создать класс *Vector3D* для работы с векторами в пространстве, содержащий следующие члены класса:
2. поля *int x, y, z*;
3. функции, позволяющие:

* вывести на экран вектор,
* вычислить длину вектора,
* сравнить два вектора на равенство.  
  Кроме того, необходимо реализовать перегрузку операций сложения, вычитания, скалярного и векторного произведения.

|  |
| --- |
| Решение: |
| #include <iostream> |
| using namespace std; |
| class Vector3D { |
| private: |
| int x, y, z; |
| public: |
| Vector3D(int A, int B, int C) : x(A), y(B), z(C) {}; |
| void ShowVector() { |
| cout << '(' << x << "; " << y << "; " << z << ')' << '\n'; |
| } |
| double LengthVector() { |
| return sqrt(x \* x + y \* y + z \* z); |
| } |
| void CompareVector(Vector3D secondvector) { |
| if (x == secondvector.x && y == secondvector.y && z == secondvector.z) |
| cout << "These are one vector" << '\n'; |
| else cout << "These are different vectors" << '\n'; |
| } |
| Vector3D operator + (Vector3D A) { |
| return Vector3D(x + A.x, y + A.y, z + A.z); |
| } |
| Vector3D operator - (Vector3D A) { |
| return Vector3D(x - A.x, y - A.y, z - A.z); |
| } |
| Vector3D operator \* (Vector3D A) { |
| return Vector3D(y \* A.z + z \* A.y, -x \* A.z - z \* A.x, x \* A.y + y \* A.x); |
| } |
| int operator % (Vector3D A) { |
| return x \* A.x + y \* A.y + z \* A.z; |
| } |
| }; |
| int main() { |
| int x, y, z; |
| cout << "Coordinates of vector A: "; |
| cin >> x >> y >> z; |
| Vector3D A(1, 2, 3); |
| cout << "Coordinates of vector B: "; |
| cin >> x >> y >> z; |
| Vector3D B(2, 3, 4); |
| A.ShowVector(); |
| B.ShowVector(); |
| cout << B.LengthVector() << '\n'; |
| A.CompareVector(B); |
| Vector3D C = A + B; |
| C.ShowVector(); |
| C = A - B; |
| C.ShowVector(); |
| C = A \* B; |
| C.ShowVector(); |
| cout << A % B; |
| } |

**Таблица входных/выходных данных:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3 4 5 1 2 3 | (1; 2; 3)  (2; 3; 4)  5.38516  These are different vectors  (3; 5; 7)  (-1; -1; -1)  (17; -10; 7)  20 |
|  |  |

## РАЗДЕЛ 5. НАСЛЕДОВАНИЕ

1. Реализовать каталог музыкальных компакт-дисков, который позволяет добавлять и удалять диски; добавлять и удалять песни на диске; просматривать содержимое целого каталога и каждого диска в отдельности; осуществлять поиск всех песен заданного исполнителя по всему каталогу.

**Решение:**

|  |
| --- |
| #include <iostream>  #include <string>  #include <vector>  #include <fstream>  using namespace std;  class MSong {  private:  string singer, name;  public:  string getSinger() {  return singer;  }  string getName() {  return name;  }  MSong(string A, string B) : singer(A), name(B) {};  };  class MCD {  public:  vector <MSong> CD;  void showCD() {  for (int i = 0; i < CD.size(); i++)  cout << i + 1 << ") " << CD[i].getSinger() << " - " << CD[i].getName() << endl;  }  void addSong(MSong Song) {  CD.push\_back(Song);  }  void removeSong(int n) {  CD.erase(CD.begin() + n - 1);  }  };  class MCatalog {  public:  vector <MCD> Catalog;  void showCatalog() {  for (int i = 0; i < Catalog.size(); i++) {  cout << "Disk " << i + 1 << ": " << endl;  Catalog[i].showCD();  }  }  void addCD(MCD Disk) {  Catalog.push\_back(Disk);  }  void removeCD(int n) {  Catalog.erase(Catalog.begin() + n - 1);  }  void search(string NAME) {  for (int i = 0; i < Catalog.size(); i++) {  for (int j = 0; j < Catalog[i].CD.size(); j++)  if (NAME == Catalog[i].CD[j].getSinger())  cout << Catalog[i].CD[j].getSinger() << " - " << Catalog[i].CD[j].getName() << endl;  }  }  void removeSong(int i, int n) {  Catalog[i].removeSong(n);  }  };  int main() {  setlocale(LC\_ALL, "Russian");  ifstream fin("input.txt");  MSong A("Vladimir", "Putin");  vector <MSong> Songs;  string singer, name;  for (int i = 0; i < 12; i++) {  fin >> singer >> name;  MSong OneSong(singer, name);  Songs.push\_back(OneSong);  }  MCD Disk1, Disk2, Disk3;  for (int i = 0; i < 4; i++) {  Disk1.CD.push\_back(Songs[i]);  Disk2.CD.push\_back(Songs[i + 4]);  Disk3.CD.push\_back(Songs[i + 8]);  }  MCatalog AllDisks;  AllDisks.Catalog.push\_back(Disk1);  AllDisks.Catalog.push\_back(Disk2);  AllDisks.Catalog.push\_back(Disk3);  AllDisks.showCatalog();  cout << endl;  AllDisks.Catalog[0].addSong(A);  cout << endl;  AllDisks.Catalog[0].showCD();  int N;  cin >> N;  AllDisks.removeSong(0, N);  cout << endl;  AllDisks.Catalog[0].showCD();  cout << endl << endl << endl;  AllDisks.showCatalog();  cout << endl;  cin >> N;  AllDisks.removeCD(N);  cout << endl;  AllDisks.showCatalog();  cout << endl;  AllDisks.addCD(Disk1);  cout << endl << endl << endl;  AllDisks.showCatalog();  cout << endl;  string nam;  cin >> nam;  AllDisks.search(nam);  } |

## РАЗДЕЛ 6. СПИСКИ

1. Создать список из чисел. Подсчитать количество пар соседних элементов, которые совпадают между собой. Оставить по одному из таких элементов, т.е. исключить все повторяющиеся, идущие подряд элементы.

**Решение:**

|  |
| --- |
| #include <iostream>  #include <fstream>  using namespace std;  template <class Item>  class List {  struct Element {  Item inf;  Element\* next;  Element(Item x) : inf(x), next(NULL) {}  };  Element\* head;  int size;  Element\* Find(int index) {  if (index < 0 || index > size)  return NULL;  else {  Element\* current = head;  for (int i = 1; i < index; i++)  current = current->next;  return current;  }  }  public:  List() : head(NULL), size(0) {}  ~List() {  while (!Empty())  Remove(0);  }  bool Empty() { return head == NULL; }  int GetLength() { return size; }  Item Get(int index) {  if (index >= 0 && index < size) {  Element\* r = Find(index);  Item i = r->inf;  return i;  }  }  void Insert(int index, Item data) {  if (index >= 0 && index < size) {  Element\* newPtr = new Element(data);  size++;  if (index == 0) {  newPtr->next = head;  head = newPtr;  }  else {  Element\* prev = Find(index - 1);  newPtr->next = prev->next;  prev->next = newPtr;  }  }  }  void Remove(int index) {  if (index >= 0 && index < size) {  Element\* current;  --size;  if (index == 0) {  current = head;  head = head->next;  }  else {  Element\* prev = Find(index - 1);  current = prev->next;  prev->next = current->next;  }  current->next = NULL;  delete current;  }  }  void Print(ofstream& out) {  for (Element\* current = head; current != NULL; current = current->next) {  out << current->inf << '\n';  }  }  void Print() {  for (Element\* current = head; current != NULL; current = current->next) {  cout << current->inf << '\n';  }  }  void Push\_back(Item data) {  if (head == NULL)  head = new Element(data);  else {  Element\* current = head;  while (current->next != NULL)  current = current->next;  current->next = new Element(data);  }  size++;  }  int pairs() {  int count = 0;  for (int i = 0; i < size - 1; i++) {  if (Get(i) == Get(i + 1)) {  Remove(i);  i--;  count++;  }  }  return count;  }  };  int main() {  ifstream fin("input.txt");  int inta;  List<int> L;  for (int i = 0; fin >> inta; i++)  L.Push\_back(inta);  fin.close();  ofstream fout("output.txt");  L.Print();  cout << endl << endl;  cout << L.pairs() << endl << endl;  L.Print();  } |

1. Создать список из чисел. Подсчитать количество пар соседних элементов, которые совпадают между собой. Оставить по одному из таких элементов, т.е. исключить все повторяющиеся, идущие подряд элементы.

**Решение:**

|  |
| --- |
| #include <iostream>  #include <fstream>  using namespace std;  template <class Item>  class DoubleLinkedList {  struct Element {  Item inf;  Element\* next;  Element\* prev;  Element(Item x) : inf(x), next(0), prev(0) {}  };  Element\* head;  Element\* tail;  int size;  bool inrange(int index) { return index >= 0 && index <= size; }  Element\* Find(int index) {  if (inrange(index)) {  Element\* cur = head;  for (int i = 0; i < index; i++)  cur = cur->next;  return cur;  }  return NULL;  }  public:  DoubleLinkedList() : head(0), tail(0), size(0) {}  ~DoubleLinkedList() {  while (!Empty())  Remove(0);  }  bool Empty() { return head == 0; }  int GetLength() { return size; }  Item Get(int index) {  if (inrange(index)) {  Element\* r = Find(index);  Item i = r->inf;  return i;  }  return NULL;  }  void InsertLeft(int index, Item data) {  if (inrange(index)) {  Element\* newPtr = new Element(data);  size++;  Element\* cur = Find(index);  if (cur == NULL) {  head = newPtr;  tail = newPtr;  }  else {  Element\* prev = Find(index - 1);  newPtr->next = cur;  newPtr->prev = cur->prev;  cur->prev = newPtr;  if (cur == head)  head = newPtr;  else newPtr->prev->next = newPtr;  }  }  }  void InsertRight(int index, Item data) {  if (inrange(index)) {  Element\* newPtr = new Element(data);  size++;  Element\* cur = Find(index);  if (cur == NULL) {  head = newPtr;  tail = newPtr;  }  else {  newPtr->next = cur->next;  newPtr->prev = cur;  cur->next = newPtr;  if (cur == tail)  tail = newPtr;  else newPtr->next->prev = newPtr;  }  }  }  void Remove(int index) {  if (inrange(index)) {  Element\* cur = Find(index);  --size;  if (size == 0) {  head = NULL;  tail = NULL;  }  else if (cur == head) {  head = head->next;  head->prev = NULL;  }  else if (cur == tail) {  tail = tail->prev;  tail->next = NULL;  }  else {  cur->prev->next = cur->next;  cur->next->prev = cur->prev;  }  cur->next = NULL;  cur->prev = NULL;  delete cur;  }  }  void Print(ofstream& out) {  for (Element\* current = head; current != NULL; current = current->next)  out << current->inf << ' ';  }  void Print() {  for (Element\* current = head; current != NULL; current = current->next)  cout << current->inf << ' ';  }  int pairs() {  int count = 0;  for (int i = 0; i < size - 1; i++) {  if (Get(i) == Get(i + 1)) {  Remove(i);  i--;  count++;  }  }  return count;  }  };  int main() {  ifstream fin("input.txt");  DoubleLinkedList<int> DLL;  int a;  fin >> a;  DLL.InsertRight(0, a);  for (int i = 0; fin >> a; i++)  DLL.InsertRight(i, a);  fin.close();  ofstream fout("output.txt");  DLL.Print();  cout << endl << endl;  cout << DLL.pairs() << endl << endl;  DLL.Print();  } |

## РАЗДЕЛ 7. СТЕК

1. Создать список из чисел. Подсчитать количество пар соседних элементов, которые совпадают между собой. Оставить по одному из таких элементов, т.е. исключить все повторяющиеся, идущие подряд элементы.

**Решение:**

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <stack>  using namespace std;  class Stack {  struct Element {  int inf;  Element\* next;  Element(int x, Element\* p) : inf(x), next(p) {}  };  Element\* head;  public:  Stack() : head(0) {}  bool empty() {  return head == 0;  }  int pop() {  if (empty()) { return 0; }  Element\* r = head;  int i = r->inf;  head = r->next;  delete r;  return i;  }  void push(int data) {  head = new Element(data, head);  }  int top() {  if (empty()) return 0;  else return head->inf;  }  int pairs() {  Stack Second;  int count = 0, kpop;  while (!empty()) {  kpop = pop();  if (kpop == top())  count++;  Second.push(kpop);  }  while (!Second.empty())  push(Second.pop());  delete Second.head;  return count;  }  void delpairs() {  Stack Second;  while (!empty()) {  if (pop() != top())  Second.push(top());  }  Second.pop();  while (!Second.empty())  push(Second.pop());  delete Second.head;  }  };  int main() {  ifstream fin("input.txt");  ofstream fout("output.txt");  Stack St;  int i;  while (fin >> i)  St.push(i);  fin.close();  cout << St.pairs() << endl; //количество пар  St.delpairs(); //удаляем пары  while (!St.empty())  fout << St.pop() << ' ';  fout.close();  } |

## РАЗДЕЛ 8. ОЧЕРЕДЬ

1. Создать список из чисел. Подсчитать количество пар соседних элементов, которые совпадают между собой. Оставить по одному из таких элементов, т.е. исключить все повторяющиеся, идущие подряд элементы.

**Решение:**

|  |
| --- |
| #include <iostream>  #include <fstream>  using namespace std;  template <class Item>  class Queue {  struct Element {  Item inf;  Element\* next;  Element(Item x) :inf(x), next(0) {}  };  Element\* head, \* tail;  int size;  public:  int Size() { return size; }  Queue() : head(0), tail(0), size(0) {}  bool Empty() { return head == 0; }  Item Get() {  if (!Empty()) {  Element\* t = head;  Item i = t->inf;  head = t->next;  if (head == NULL)  tail = NULL;  delete t;  size--;  return i;  }  }  void Put(Item data) {  Element\* t = tail;  tail = new Element(data);  if (!head)  head = tail;  else t->next = tail;  size++;  }  void Print(ofstream& out) {  for (Element\* current = head; current != NULL; current = current->next) {  out << current->inf << ' ';  }  }  void Print() {  for (Element\* current = head; current != NULL; current = current->next) {  cout << current->inf << ' ';  }  }  };  int pairs(Queue<int>& queue) {  if (queue.Size() > 1) {  int count = 0, lpop = queue.Get();  for (int i = 0, size = queue.Size(), rpop; i < size; i++) {  rpop = queue.Get();  if (lpop == rpop)  count++;  else queue.Put(lpop);  lpop = rpop;  }  queue.Put(lpop);  return count;  }  else return 0;  }  int main() {  ifstream fin("input.txt");  ofstream fout("output.txt");  int k;  Queue<int> Q;  while (fin >> k)  Q.Put(k);  fin.close();  Q.Print();  cout << endl << pairs(Q) << endl;  Q.Print(fout);  Q.Print();  } |

## РАЗДЕЛ 9. VECTOR - LIST

1. Решить одну задачу с использованием вектора, другую – списка. Для обеих теоретически обосновать временную сложность решения и выбор контейнера.
2. Вставить нулевые элементы до и после последнего чётного.
3. Заменить последний элемент на минимальных из двух первых.

**Решение:**

|  |
| --- |
| #include <iostream>  #include <vector>  #include <list>  #include <fstream>  using namespace std;  int main() {  int a;  ifstream fin("input.txt");  vector<int> Vector; //Б  list <int> List; //А  while (fin >> a) {  Vector.push\_back(a); //Б  List.push\_back(a); //А  }  /\*------------------------------ Задача А -----------------------------\*/  cout << "Задача А: Вставить нулевые элементы до и после последнего четного." << endl;  list<int>::iterator mem\_iter;  for (list<int>::iterator iter = List.begin(); iter != List.end(); iter++)  if (\*iter % 2 == 0)  mem\_iter = iter;  List.insert(mem\_iter, 0);  List.insert(++mem\_iter, 0);  for (list<int>::iterator iter = List.begin(); iter != List.end(); iter++)  cout << \*iter << ' ';  /\*------------------------------ Задача Б -----------------------------\*/  cout << endl << "Задача А: Вставить нулевые элементы до и после последнего четного." << endl;  if (Vector[0] > Vector[1])  Vector[Vector.size() - 1] = Vector[1];  else Vector[Vector.size() - 1] = Vector[0];  for (int i = 0; i < Vector.size(); i++)  cout << Vector[i] << ' ';  } |

1. Решить одну задачу с использованием вектора, другую – списка. Для обеих теоретически обосновать выбор контейнера. Создать класс "чего-то" и класс "список <чего-то>". Выбрать контейнер исходя из операций. Спиcок товаров на складе. Операции:
2. Разбить всё что тяжелее N кг на две идущие подряд записи по N/2 кг. Добавить новый товар в конец списка.
3. Разбить всё что тяжелее N кг на две записи по N/2 кг, первую оставить на прежнем месте, а вторую приписать в конец списка. Показать товар, расположенный в середине списка.

Решение:

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <vector>  #include <list>  using namespace std;  class merchandise {  string name;  int weight;  public:  merchandise() : weight(0) {}  merchandise(string n, int w) {  if (w > 0) {  weight = w;  name = n;  }  }  string getname() { return name; }  int getweight() { return weight; }  friend istream& operator>>(istream& in, merchandise& m) { return in >> m.name >> m.weight; }  friend ostream& operator<<(ostream& out, merchandise m) { return out << m.getname() << ' ' << m.getweight(); }  bool operator>(int w) { return weight > w; }  merchandise operator=(merchandise A) { return A; }  merchandise operator/=(int n) { return { name, weight /= 2 }; }  };  class storage\_list {  list<merchandise> wares;  public:  void push\_back(merchandise m) { wares.push\_back(m); }  void divide(int w) {  if (!wares.empty() && w > 0)  for (list<merchandise>::iterator iter = wares.begin(); iter != wares.end(); iter++)  if (\*iter > w) {  \*iter /= 2;  wares.insert(iter, \*iter);  }  }  void show() {  if (!wares.empty())  for (list<merchandise>::iterator iter = wares.begin(); iter != wares.end(); iter++)  cout << \*iter << '\n';  }  };  class storage\_vector {  vector<merchandise> wares;  public:  void push\_back(merchandise m) { wares.push\_back(m); }  merchandise showmidst() { return wares[wares.size() / 2]; }  void divide(int w) {  if (!wares.empty() && w > 0)  for (int i = 0; i < wares.size(); i++)  if (wares[i] > w) {  wares[i] /= 2;  wares.push\_back(wares[i]);  }  }  void show() {  if (!wares.empty())  for (vector<merchandise>::iterator iter = wares.begin(); iter != wares.end(); iter++)  cout << \*iter << endl;  }  };  int main() {  ifstream fin("input.txt");  int N;  merchandise Merch;  storage\_list List;  storage\_vector Vector;  fin >> N;  while (fin >> Merch) {  List.push\_back(Merch);  Vector.push\_back(Merch);  }  //------------- Задача А: Разбить всё что тяжелее N кг на две идущие подряд записи по N/2 кг. Добавить новый товар в конец списка -------------  cout << "A: " << endl << "1) Showing list" << endl;  List.show();  cout << endl << "2) Divide method call" << endl;  List.divide(N);  List.show();  cout << endl << "3) Push back method call" << endl;  List.push\_back(Merch);  List.show();  //- Задача Б: Разбить всё что тяжелее N кг на две записи по N/2 кг, первую оставить на прежнем месте, а вторую приписать в конец списка. Показать товар, расположенный в середине списка -  cout << "B: " << endl << "1) Showing vector" << endl;  Vector.show();  cout << endl << "2) Divide method call" << endl;  Vector.divide(N);  Vector.show();  cout << endl << "3) Showing middle element method call" << endl;  cout << Vector.showmidst();  } |

## РАЗДЕЛ 10. СОРТИРОВКИ

1. В файле *input.txt* содержатся сведения о группе студентов в формате: номер группы, фамилия, имя, отчество, год рождения, оценки по пяти предметам.

Переписать данные файла *input.txt* в файл *output.txt*, отсортировав их в алфавитном порядке по фамилии, а затем по возрастанию года рождения методами пузырька, вставки, выбора.

**Решение:**

|  |
| --- |
| #include <iostream>  #include <string>  #include <fstream>  using namespace std;  class student {  int group\_number;  string surname, name, lastname;  int wasborn;  int O[5];  public:  student() : group\_number(000), surname("\_"), name("\_"), lastname("\_"), wasborn(0) {  for (int i = 0; i < 5; i++)  O[i] = 0;  }  student(int g, string s, string n, string l, int y, int\* o) : group\_number(g), surname(s), name(n), lastname(l), wasborn(y) {  for (int i = 0; i < 5; i++)  O[i] = o[i];  }  string getSurname() {  return surname;  }  string getName() {  return name;  }  string getLastname() {  return lastname;  }  int getYear() {  return wasborn;  }  int getGroup() {  return group\_number;  }  int getO(int n) {  if (n >= 0 && n < 5)  return O[n];  else return 0;  }  };  bool srav(student x, student y) {  if (x.getSurname() != y.getSurname())  return x.getSurname() < y.getSurname();  else if (x.getName() != y.getName())  return x.getName() < y.getName();  else if (x.getLastname() != y.getLastname())  return x.getLastname() < y.getLastname();  else return x.getYear() < y.getYear();  }  void bubblesort(student\* A, int n) {  student temp;  int i, j;  for (i = 0; i < n - 1; i++)  for (j = n - 1; j > i; j--)  if (srav(A[j], A[j - 1])) {  temp = A[j];  A[j] = A[j - 1];  A[j - 1] = temp;  }  }  void insertsort(student\* A, int n) {  student temp;  int i, j;  for (i = 1; i < n; i++) {  j = i;  while (srav(A[j], A[j - 1])) {  temp = A[j];  A[j] = A[j - 1];  A[j - 1] = temp;  j--;  if (j == 0)  break;  }  }  }  void selectsort(student\* A, int n) {  student temp, low;  int lowindex, i, j;  for (i = 0; i < n - 1; i++) {  lowindex = i;  low = A[i];  for (j = i + 1; j < n; j++)  if (srav(A[j], low)) {  low = A[j];  lowindex = j;  }  temp = A[i];  A[i] = A[lowindex];  A[lowindex] = temp;  }  }  int main() {  student A[10];  ifstream fin("input.txt");  ofstream fout("output.txt");  string s, n, l;  int g, y;  int o[5];  for (int i = 0; i < 10; i++) {  fin >> g >> s >> n >> l >> y >> o[0] >> o[1] >> o[2] >> o[3] >> o[4];  student Temp(g, s, n, l, y, o);  A[i] = Temp;  }  bubblesort(A, 10);  //insertsort(A, 10);  //selectsort(A, 10);  for (int i = 0; i < 10; i++) {  fout << A[i].getGroup() << ' ' << A[i].getSurname() << ' ' << A[i].getName() << ' ' << A[i].getLastname() << ' ' << A[i].getYear() << ' ' << A[i].getO(0) << ' ' << A[i].getO(1) << ' ' << A[i].getO(2) << ' ' << A[i].getO(3) << ' ' << A[i].getO(4) << '\n';  }  } |

1. Дана матрица размерностью ***N × N***, содержащая целые числа. Отсортировать диагонали матрицы, расположенные выше главной, по возрастанию элементов, а диагонали матрицы, расположенные ниже главной, по убыванию элементов методами быстрой и пирамидальной сортировок.

**Решение:**

|  |
| --- |
| #include <iostream>  using namespace std;  void pushdown(int\*& A, int first\_index, int last\_index, bool descending) {  int r = first\_index;  while (r <= last\_index / 2) {  if (last\_index == 2 \* r) {  if (A[r] > A[2 \* r])  swap(A[r], A[2 \* r]);  r = last\_index;  }  else if (A[r] > A[2 \* r] && A[2 \* r] <= A[2 \* r + 1]) {  swap(A[r], A[2 \* r]);  r = 2 \* r;  }  else if (A[r] > A[2 \* r + 1] && A[2 \* r + 1] < A[2 \* r]) {  swap(A[r], A[2 \* r + 1]);  r = 2 \* r + 1;  }  else r = last\_index;  }  }  void heapsort(int\*& A, int N, bool descending = 0) {  for (int i = N / 2; i >= 1; i--)  pushdown(A, i, N, descending);  for (int i = N; i >= 2; i--) {  swap(A[1], A[i]);  pushdown(A, 1, i - 1, descending);  }  }  void quicksort(int\*& A, int first\_index, int last\_index, bool descending = 0) {  int l = first\_index, r = last\_index;  int pivot = A[(l + r) / 2];  if (!descending) {  do {  while (A[l] < pivot)  l++;  while (A[r] > pivot)  r--;  if (l <= r) {  swap(A[l], A[r]);  l++;  r--;  }  } while (l < r);  }  else {  do {  while (A[l] > pivot)  l++;  while (A[r] < pivot)  r--;  if (l <= r) {  swap(A[l], A[r]);  l++;  r--;  }  } while (l < r);  }  if (first\_index < r)  quicksort(A, first\_index, r);  if (l < last\_index)  quicksort(A, l, last\_index);  }  void diagonal\_sort(int\*\*& A, int A\_size) {  int\*\* D = new int\* [(A\_size - 2) \* 2]; //будущий одномерный массив одномерных массивов диагоналей (кривой двумерный массив)  for (int k = 1; k < A\_size - 1; k++) { //создание массивов диагоналей  D[k - 1] = new int[A\_size - k]; //D[j] ?  D[k + A\_size - 3] = new int[A\_size - k]; //D[i - 1 + A\_size - 2] = D[i + A\_size - 2] - смещение для отличия индексов  }  for (int i = 0; i < A\_size; i++)  for (int j = 0; j < A\_size; j++) {  if (i < j && (i != 0 || j != A\_size - 1))  D[j - i - 1][i] = A[i][j];  else if (i > j && (i != A\_size - 1 || j != 0))  D[i - j - 1 + A\_size - 2][j] = A[i][j];  }  for (int i = 0, j = A\_size - 2; i < (A\_size - 2) \* 2; i++, j--) { //специфическая пробежка по кривому массиву с вызовом сортировки для каждого подмассива (каждой диагонали)  if (i < A\_size - 2) {  //quicksort(D[i], 0, j);  heapsort(D[i], j + 1);  }  else {  //quicksort(D[i], 0, j, true);  heapsort(D[i], j + 1, true);  }  }  for (int i = 0; i < A\_size; i++)  for (int j = 0; j < A\_size; j++) {  if (i < j && (i != 0 || j != A\_size - 1))  A[i][j] = D[j - i - 1][i];  else if (i > j && (i != A\_size - 1 || j != 0))  A[i][j] = D[i - j - 1 + A\_size - 2][j];  }  void print\_array(int\*\* A, int A\_size) {  for (int i = 0; i < A\_size; i++, cout << endl)  for (int j = 0; j < A\_size; j++, cout << '\t')  cout << A[i][j];  }  int main() {  int N;  cin >> N;  int\*\* Array = new int\* [N];  for (int i = 0, k = 36; i < N; i++) {  Array[i] = new int[N];  for (int j = 0; j < N; j++, k--)  Array[i][j] = k;  }  //print\_array(Array, N);  diagonal\_sort(Array, N);  print\_array(Array, N);  } |

1. В файле *input.txt* содержатся сведения о группе студентов в формате: номер группы, фамилия, имя, отчество, год рождения, оценки по пяти предметам.

Переписать данные файла *input.txt* в файл *output.txt*, отсортировав их по убыванию суммы оценок алгоритмом *stable\_sort*.

**Решение:**

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <algorithm>  #include <vector>  using namespace std;  class student {  int group\_number;  string surname, name, lastname;  int wasborn;  int O[5];  public:  student() : group\_number(000), surname("\_"), name("\_"), lastname("\_"), wasborn(0) {  for (int i = 0; i < 5; i++)  O[i] = 0;  }  student(int g, string s, string n, string l, int y, int\* o) : group\_number(g), surname(s), name(n), lastname(l), wasborn(y) {  for (int i = 0; i < 5; i++)  O[i] = o[i];  }  string getSurname() { return surname; }  string getName() { return name; }  string getLastname() { return lastname; }  int getYear() { return wasborn; }  int getGroup() { return group\_number; }  int getO(int n) {  if (n >= 0 && n < 5)  return O[n];  else return 0;  }  friend istream& operator>>(istream& in, student& S) { return in >> S.group\_number >> S.surname >> S.name >> S.lastname >> S.wasborn >> S.O[0] >> S.O[1] >> S.O[2] >> S.O[3] >> S.O[4]; }  };  int main() {  ifstream fin("input.txt");  vector<student> Students;  for (student a; fin >> a; Students.push\_back(a));  stable\_sort(Students.begin(), Students.end(), [](student a, student b) {  return a.getO(0) + a.getO(1) + a.getO(2) + a.getO(3) + a.getO(4) > b.getO(0) + b.getO(1) + b.getO(2) + b.getO(3) + b.getO(4);  });  for (student st : Students)  cout << st.getGroup() << ' ' << st.getSurname() << ' ' << st.getName() << ' ' << st.getLastname() << ' ' << st.getYear() << ' ' << st.getO(0) + st.getO(1) + st.getO(2) + st.getO(3) + st.getO(4) << endl;  } |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 141 Sharov Kirill Vladimirovich 2003 5 5 4 3 3  151 Semenov Vladimir Mikhailovich 2002 5 4 5 5 5  241 Tarenkov Ivan Batkovich 2002 5 4 3 3 3  341 Kotov Sergei Grigorievich 2001 3 4 3 3 3  141 Presnyakov Alexey Romanovich 2003 5 4 4 4 3  281 Presnyakov Alexey Romanovich 2002 5 4 5 4 5  111 Sharovsky Kirill Vladimirovich 2003 5 5 5 5 5  221 Sharov Kirill Vasilievich 2002 5 4 3 3 5  441 Sharov Kirill Vladimirovich 2001 5 4 5 5 5  161 Tarelkin Alexander Ustinovich 2003 5 5 5 5 5 | 111 Sharovsky Kirill Vladimirovich 2003 25  161 Tarelkin Alexander Ustinovich 2003 25  151 Semenov Vladimir Mikhailovich 2002 24  441 Sharov Kirill Vladimirovich 2001 24  281 Presnyakov Alexey Romanovich 2002 23  141 Sharov Kirill Vladimirovich 2003 20  141 Presnyakov Alexey Romanovich 2003 20  221 Sharov Kirill Vasilievich 2002 20  241 Tarenkov Ivan Batkovich 2002 18  341 Kotov Sergei Grigorievich 2001 16 |
|  |  |

## РАЗДЕЛ 11. АЛГОРИТМЫ

1. Точки на плоскости заданы парами целочисленных координат.
2. Удалить все точки из нижней половины системы координат.
3. Подсчитать количество точек, лежащих на одной вертикальной или горизонтальной прямой с данной.
4. Найти последнюю точку, удалённую от начала координат не менее чем на 10.
5. Расположить в порядке возрастания суммы координат.

**Решение:**

|  |
| --- |
|  |

## РАЗДЕЛ 12. SET

1. Даны N целых чисел. Найти все такие цифры, которые встречаются в первом и последнем числах, но не встречаются ни в одном из остальных.

**Решение:**

|  |
| --- |
| #include <iostream>  #include <set>  #include <algorithm>  using namespace std;  int numbers\_amount(int N) {  int count = 0;  for (; N > 0; count++)  N /= 10;  return count;  }  int main() {  int N;  cin >> N;  set<int> Set;  for (int i = 0; i < N; i++)  Set.insert(rand() % 1000);  cout << \*Set.begin() << ' ' << \*(--Set.end()); //проверка первого и последнего  set<int> Numbers\_FL;  for (int i = 0, num1 = \*Set.begin(); i < numbers\_amount(\*Set.begin()); i++, num1 /= 10)  for (int j = 0, num2 = \*(--Set.end()); j < numbers\_amount(\*(--Set.end())); j++, num2 /= 10)  if (num1 % 10 == num2 % 10)  Numbers\_FL.insert(num1 % 10);  for (auto S = Set.begin(); S != Set.end(); S++)  for (N = \*S ; N > 0; N /= 10)  remove\_if(Numbers\_FL.begin(), Numbers\_FL.end(), [N](int a) { return a == N % 10; });  for (auto Number : Numbers\_FL)  cout << Number << ' ';  } |

## РАЗДЕЛ 13. MAP