

0.1 Пример работы с данными

Пример 0.1.

УСЛОВИЕ: Дан файл, содержащий следующие данные о сотрудниках: фамилия, дата рождения, зарплата. Используя глупую сортировку, отсортировать данные сначала по фамилии, потом по году рождения, потом по зарплате.

РЕШЕНИЕ.

Сначала создадим структуру `date`:

Листинг 1.

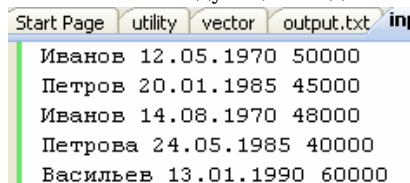
```
1 struct date{//дата
2     int dd, mm, yy;
3 };
```

И структуру `people`:

Листинг 2.

```
1 struct people{//данные о человеке
2     string Surname; //фамилия
3     date DateOfBirth; //дата рождения
4     int Salary; //зарплата
5 };
```

Файл имеет следующий вид:



```
Start Page utility vector output.txt input.txt
Иванов 12.05.1970 50000
Петров 20.01.1985 45000
Иванов 14.08.1970 48000
Петрова 24.05.1985 40000
Васильев 13.01.1990 60000
```

Видно, что дата рождения введена как строка. Можно использовать разные способы считывания. Считаем дату рождения как строку, потом воспользуемся функцией, уже описанной при работе со структурами, перевода из строки в дату (строки 22–31 Листинга 6).

Считываем из файла данные с помощью оператора `>>` до пробелов и записываем в соответствующие поля структуры:

Листинг 3.

```
1 vector<people> inFile(){//ввод из файла
2     vector<people> x;
3     people temp;
4     while(in.peek() != EOF){
5         in >> temp.Surname;//фамилия
6         string tmp;//дата рождения
7         in >> tmp;
8         temp.DateOfBirth = Str_to_Date(tmp);
9         in >> temp.Salary;//зарплата
10        x.push_back(temp);
11    }
12    return x;
13 }
```

Для вывода в новый файл воспользуемся манипуляторами, т. к., фамилии обычно имеют разную длину. Не забудьте подключить библиотеку `<iomanip>`. Будем выравнивать по левому краю с помощью `left` и выделять определенное количество позиций для записи отдельного поля (`setw(N)`). Не забудьте добавить нули перед цифрами в случае, если день или месяц рождения меньше 10:

Листинг 4.

```
1 void print(people x){//вывод в файл
2     out << setw(10) << left << x.Surname;//по левому краю, 10 позиций для фамилии
3     if (x.DateOfBirth.dd < 10) out << left << '0' << x.DateOfBirth.dd << ' ' << ' ';//добавляем 0
4     else out << left << x.DateOfBirth.dd << ' ' << ' ';
5     if (x.DateOfBirth.mm < 10) out << left << '0' << x.DateOfBirth.mm << ' ' << ' ';
6     else out << left << x.DateOfBirth.mm << ' ' << ' ';
7     out << left << setw(6) << x.DateOfBirth.yy;//на год 6 позиций
8     out << left << setw(10) << x.Salary << endl;//зарплата
9 }
```

Отсортировать сначала по фамилии, потом по году рождения, потом по зарплате означает, сортировку по алфавиту, если фамилии одинаковые, тогда по году рождения, а если и год рождения совпадает, то по зарплате. C++ позволяет переопределять операторы с помощью ключевого слова `operator`. В нашем случае переопределим оператор `<`:

Листинг 5.

```
1 bool operator < (people a, people b){//переопределяем оператор < в соответствии с условием
2     if (a.Surname < b.Surname) return true;
3     if (a.Surname == b.Surname && a.DateOfBirth.yy < b.DateOfBirth.yy) return true;
4     if (a.Surname == b.Surname && a.DateOfBirth.yy == b.DateOfBirth.yy && a.Salary < b.Salary) return true;
5     return false;
6 }
```

В результате при сортировке теперь можно просто сравнивать два элемента типа `people`.

Итак, окончательный код программы:

Листинг 6.

```
1 #include<iostream>
2 #include<fstream>
3 #include<string>
4 #include<vector>
5 #include<iomanip>
6 using namespace std;
7
8
9 ifstream in("input.txt");
10 ofstream out("output.txt");
11
12 struct date{//дата
13     int dd, mm, yy;
14 };
15
16 struct people{//данные о человеке
17     string Surname; //фамилия
18     date DateOfBirth; //дата рождения
19     int Salary; //зарплата
20 };
21
22 date Str_to_Date(string str){//из строки в дату
23     date x;
24     string temp = str.substr(0, 2); //день
25     x.dd = atoi(temp.c_str());
26     temp = str.substr(3, 2); //месяц
27     x.mm = atoi(temp.c_str());
28     temp = str.substr(6, 4); //год
29     x.yy = atoi(temp.c_str());
30     return x;
31 }
32
33 vector<people> inFile(){//ввод из файла
34     vector<people> x;
35     people temp;
36     while(in.peek() != EOF){
37         in >> temp.Surname; //фамилия
38         string tmp; //дата рождения
39         in >> tmp;
40         temp.DateOfBirth = Str_to_Date(tmp);
41         in >> temp.Salary; //зарплата
42         x.push_back(temp);
43     }
44     return x;
45 }
46
47 void print(people x){//вывод в файл
48     out << setw(10) << left << x.Surname; //по левому краю, 10 позиций для фамилии
49     if (x.DateOfBirth.dd < 10) out << left << '0' << x.DateOfBirth.dd << ' '; //добавляем 0
50     else out << left << x.DateOfBirth.dd << ' ';
51     if (x.DateOfBirth.mm < 10) out << '0' << x.DateOfBirth.mm << ' ';
```

```

52 else out << x.DateOfBirth.mm << ' .';
53 out << left << setw(6) << x.DateOfBirth.yy; //на год 6 позиций
54 out << left << setw(10) << x.Salary << endl; //заплата
55 }
56
57 bool operator < (people a, people b){ //переопределяем оператор < в соответствии с условием
58     if (a.Surname < b.Surname) return true;
59     if (a.Surname == b.Surname && a.DateOfBirth.yy < b.DateOfBirth.yy) return true;
60     if (a.Surname == b.Surname && a.DateOfBirth.yy == b.DateOfBirth.yy && a.Salary < b.Salary) return true;
61     return false;
62 }
63
64 void stupid_sort(vector<people> &x){ //глупая сортировка
65     for(int i = 0; i < x.size() - 1; i++)
66         if(x[i + 1] < x[i]){ //не на месте
67             swap(x[i], x[i + 1]); //поменяли
68             i = 0; //возвращаемся в начало
69         }
70     else i++; //идем дальше
71 }
72
73 int main(){
74     vector<people> x;
75     x = inFile();
76     stupid_sort(x);
77     for(int i = 0; i < x.size(); i++)
78         print(x[i]);
79     return 0;
80 }

```

Результат работы:

Start Page	utility	vector	output.txt	input.t
Васильев	13.01.1990	60000		
Иванов	14.08.1970	48000		
Иванов	12.05.1970	50000		
Петров	20.01.1985	45000		
Петрова	24.05.1985	40000		

□

0.2 Пример работы с массивами

Пример 0.2.

УСЛОВИЕ: Дан файл, содержащий матрицу целых чисел размерностью $N \times N$. Диагонали, параллельные побочной, отсортировать по возрастанию с помощью глупой сортировки.

РЕШЕНИЕ.

О диагоналях можно говорить только в случае квадратной матрицы. Рассмотрим матрицу размерностью 4×4 .

	0	1	2	3	
4	a_{33}	a_{23}	a_{13}	a_{03}	
5	a_{32}	a_{22}	a_{12}	a_{02}	
6	a_{31}	a_{21}	a_{11}	a_{01}	
	a_{30}	a_{20}	a_{10}	a_{00}	

Диагонали, параллельные побочной, показаны курсивом. Пронумеруем их по часовой стрелке. Видно, что всего будет $2N - 1$ диагональ (можно не рассматривать первую и последнюю, содержащие по одному элементу.)

Если сложить индексы элементов, расположенных на одной диагонали, то видно, что их сумма равна номеру диагонали. Этим свойством и воспользуемся для определения одномерного массива, который необходимо отсортировать.

Входной файл:

Вход: a — двумерный массив, размерностью $N \times N$

Выход: отсортированный массив

начало алгоритма

цикл для k от 1 до $2N - 2$ выполнять

· создаем вектор;

цикл для i от 0 до N выполнять

если $k - i < N$ и $k - i \geq 0$ то

· добавляем в вектор элемент $a[i][k - i]$;

· сортируем полученный вектор;

· $L = 0$;

цикл для i от 0 до N выполнять

если $k - i < N$ и $k - i \geq 0$ то

· $a[i][k - i]$ равен L -ому элементу вектора;

· увеличиваем L ;

· очищаем вектор;

конец алгоритма

Start Page	utility
5	
2	3 7 2 1
4	5 8 1 3
7	2 9 1 4
6	9 2 1 8
0	4 8 3 1

Выходной файл:

Start Page	utility	vecc
5		
2	3 5 2 0	
4	7 2 1 1	
7	6 1 2 1	
8	9 3 4 3	
9	4 8 8 1	

□

Пример 0.3.

УСЛОВИЕ: Дан файл, содержащий матрицу целых чисел размерностью $N \times N$. Диагонали, параллельные главной, отсортировать по возрастанию с помощью глупой сортировки.

РЕШЕНИЕ.

О диагоналях можно говорить только в случае квадратной матрицы. Рассмотрим матрицу размерностью 4×4 .

0	1	2	3
a_{33}	a_{23}	a_{13}	a_{03}
a_{32}	a_{22}	a_{12}	a_{02}
a_{31}	a_{21}	a_{11}	a_{01}
a_{30}	a_{20}	a_{10}	a_{00}
	1	2	3

Диагонали, параллельные главной, показаны курсивом. Пронумеруем их по часовой стрелке.

Будет два разных случая: выше главной диагонали и ниже ее. Главную диагональ можно отнести к любому из этих случаев.

Видно, что выше главной диагонали индекс j элемента равен $k + i$, где k — номер диагонали.

Ниже главной диагонали, наоборот, индекс i элемента равен $k + j$.

Входной файл:

Вход: a — двумерный массив, размерностью $N \times N$

Выход: отсортированный массив

начало алгоритма

цикл для k от 1 до $N - 1$ выполнять

· создаем вектор;

цикл для i от 0 до N выполнять

если $k + i < N$ то

└ · добавляем в вектор элемент $a[i][k + i]$;

· сортируем полученный вектор;

· $L = 0$;

цикл для i от 0 до N выполнять

если $k - i < N$ то

└ · $a[i][k + i]$ равен L -ому элементу вектора;

└ · увеличиваем L ;

· очищаем вектор;

цикл для j от 0 до N выполнять

если $k + j < N$ то

└ · добавляем в вектор элемент $a[k + j][j]$;

· сортируем полученный вектор;

· $L = 0$;

цикл для j от 0 до N выполнять

если $k + j < N$ то

└ · $a[k + j][j]$ равен L -ому элементу вектора;

└ · увеличиваем L ;

· очищаем вектор;

конец алгоритма

```
Start Page utility
5
2 3 7 2 1
4 5 8 1 3
7 2 9 1 4
6 9 2 1 8
0 4 8 3 1
```

Выходной файл:

```
in.cpp Start Page
5
1 1 1 2 1
2 1 3 4 3
7 2 2 8 7
4 8 3 5 8
0 6 9 4 9
1
```