

0.1 Пример работы с векторами

Пример 0.1. УСЛОВИЕ:

Дан набор целых чисел. Удалить все четные элементы. Заменить все максимальные элементы новым элементом. Найти сумму элементов, кратных X . Отсортировать последовательность. Удалить все дубликаты.

РЕШЕНИЕ:

Сначала необходимо создать вектор. Заполняем его, используя метод `push_back(x)`. Крайне желательно вывести на экран полученный вектор:

Листинг 1. Заполнение вектора

```
1 vector<int> vec;
2 for(int i = 0; i < n; i++){//заполнение
3     cout << "a[" << i << "] = ";
4     cin >> x;
5     vec.push_back(x);
6 }
7
8 for(vector<int>::iterator iter = vec.begin(); iter != vec.end(); iter++)
9     cout << *iter;
10 cout << endl;
```

Для удаления всех четных элементов воспользуемся функцией `remove_if`. Предварительно необходимо написать функцию типа `bool`, которая возвращает `true`, если x — четное число:

Листинг 2. дополнительная функция

```
1 bool odd(int x){
2     return x % 2 == 0;
3 }
```

Функция `remove` не удаляет реально элементы, а только переставляет элементы в правильном порядке и возвращает итератор, показывающий на конец нового вектора. Для реального удаления элементов воспользуемся методом `erase`, удаляя все элементы, начиная с полученного итератора и до конца вектора. Полученный результат выводим на экран:

Листинг 3. удаление четных элементов

```
1 vector<int>::iterator it = remove_if(vec.begin(), vec.end(), odd);
2 vec.erase(it, vec.end());
3
4 for(vector<int>::iterator iter = vec.begin(); iter != vec.end(); iter++)
5     cout << *iter;
6 cout << endl;
```

Для нахождения максимального элемента воспользуемся функцией `max_element`. Эта функция возвращает итератор, показывающий на первый максимальный элемент. Для использования в качестве параметра в функции `replace` (замена значений) введем новую переменную и присвоим ей значение итератора, иначе будет заменен только элемент на который указывает итератор, а не все максимальные элементы. Полученный результат выводим на экран:

Листинг 4. замена максимальных

```
1 int Max = *max_element(vec.begin(), vec.end());
2 replace(vec.begin(), vec.end(), Max, Y);
3
4 for(vector<int>::iterator iter = vec.begin(); iter != vec.end(); iter++)
5     cout << *iter;
6 cout << endl;
```

Для нахождения суммы элементов, кратных X воспользуемся функцией `accumulate`. Предварительно опишем функцию, возвращающую `true`, если x не кратен X . Для использования этой функции необходимо подключить библиотеку `<numeric>`. Создадим копию вектора. Заменяем в этом векторе все элементы, не кратные X нулем, используя функции `replace_if`. После этого определим сумму элементов вектора:

Листинг 5. нахождение суммы

```
1 vector<int> temp(vec); //копия вектора
2 replace_if(temp.begin(), temp.end(), kratn, 0); //замена
3 cout << "sum = " << accumulate(temp.begin(), temp.end(), 0) << endl;
```

Для сортировки воспользуемся функцией `sort`. Для удаления дубликатов используем функцию `unique`. Результат работы этой функции аналогичен функции `remove` (не удаляет реально дубликаты). Поэтому также пользуемся функцией `erase`. После каждого действия выводим результаты на экран:

Листинг 6. сортировка

```

1 sort(vec.begin(), vec.end());
2 for(vector<int>::iterator iter = vec.begin(); iter != vec.end(); iter++)
3     cout << *iter;
4 cout << endl;
5
6 vector<int>::iterator iter1 = unique(vec.begin(), vec.end());
7 vec.erase(iter1, vec.end());
8 for(vector<int>::iterator iter = vec.begin(); iter != vec.end(); iter++)
9     cout << *iter;
10 cout << endl;

```

ПРОГРАММА

```

1 #include<iostream>
2 #include<vector>
3 #include<algorithm>
4 #include<numeric>
5 using namespace std;
6
7 bool odd(int x){//четные
8     return x % 2 == 0;
9 }
10
11 int X; //предикат может содержать только один параметр
12 bool kratn(int x){//не кратный X
13     return x % X != 0;
14 }
15
16 int main(){
17     setlocale(LC_ALL, "russian");
18     int n;
19     cout << "n = "; cin >> n;
20     vector<int> vec;
21
22     int x;
23     for(int i = 0; i < n; i++){//создание вектора
24         cout << "a[" << i << "] = ";
25         cin >> x;
26         vec.push_back(x);
27     }
28
29     for(vector<int>::iterator iter = vec.begin(); iter != vec.end(); iter++)//вывод
30         cout << *iter << " ";
31     cout << endl;
32
33     cout << " Удаление четных элементов:\n";
34     vector<int>::iterator it = remove_if(vec.begin(), vec.end(), odd);
35     vec.erase(it, vec.end());
36
37     for(vector<int>::iterator iter = vec.begin(); iter != vec.end(); iter++)
38         cout << *iter << " ";
39     cout << endl;
40
41     cout << "Замена максимальных\n";
42     int Y;
43     cout << "Y = "; cin >> Y;
44     int Max = *max_element(vec.begin(), vec.end());
45     replace(vec.begin(), vec.end(), Max, Y);
46
47     for(vector<int>::iterator iter = vec.begin(); iter != vec.end(); iter++)
48         cout << *iter << " ";
49     cout << endl;
50
51     cout << "Сумма элементов:\n";
52     cout << "X = "; cin >> X;
53     vector<int> temp(vec); //копия вектора
54     replace_if(temp.begin(), temp.end(), kratn, 0);

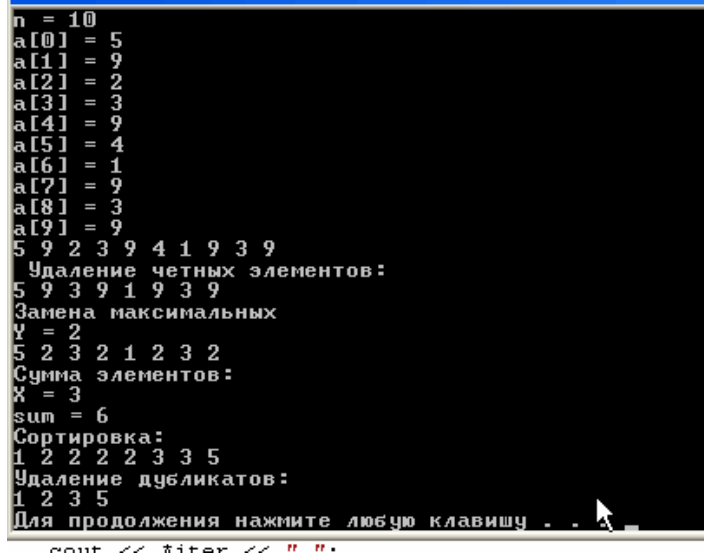
```

```

55 cout << "sum = " << accumulate(temp.begin(), temp.end(), 0) << endl;
56
57 cout << "Сортировка:\n";
58 sort(vec.begin(), vec.end());
59 for(vector<int>::iterator iter = vec.begin(); iter != vec.end(); iter++)
60     cout << *iter << " ";
61 cout << endl;
62
63 cout << "Удаление дубликатов:\n";
64 vector<int>::iterator iter1 = unique(vec.begin(), vec.end());
65 vec.erase(iter1, vec.end());
66 for(vector<int>::iterator iter = vec.begin(); iter != vec.end(); iter++)
67     cout << *iter << " ";
68 cout << endl;
69
70 system("pause");
71 return 0;
72 }

```

РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ:



```

n = 10
a[0] = 5
a[1] = 9
a[2] = 2
a[3] = 3
a[4] = 9
a[5] = 4
a[6] = 1
a[7] = 9
a[8] = 3
a[9] = 9
5 9 2 3 9 4 1 9 3 9
Удаление четных элементов:
5 9 3 9 1 9 3 9
Замена максимальных
У = 2
5 2 3 2 1 2 3 2
Сумма элементов:
Х = 3
sum = 6
Сортировка:
1 2 2 2 2 3 3 5
Удаление дубликатов:
1 2 3 5
Для продолжения нажмите любую клавишу . . .

```

0.2 Пример работы с множествами

Пример 0.2. Дан текст, содержащий предложения, разделенных знаками препинания из набора «.?!». Предложения в свою очередь состоят из слов, отделенных друг от друга пробелами. Найти слова (без учета регистра) и их количество, которые встречаются в вопросительных, но не встречаются в восклицательных предложениях.

Алгоритм 1: Работа с множествами

Вход: `str` — текст.

Выход: слова, удовлетворяющие условию

начало алгоритма

- Создать три множества: `vopr`, `vosk`, `pov` ;
- строка `razd = "!.?."`;
- определить конец первого предложения с помощью функции `k = str.find_first_of(razd, pos)`;
- цикл пока не дошли до конца текста выполнять**
 - сохранить в отдельной переменной `tmp = str[k]`;
 - найденное предложение записать в новую переменную `temp`, используя функцию `str.substr(pos, k - pos)`;
 - добавить в конце предложения пробел;
 - `pos1 = 0` ;
 - найти конец первого слова с помощью функции `k1 = temp.find_first_of(" ", pos1)`;
 - цикл пока не дошли до конца предложения выполнять**
 - найденное слово записать в новую переменную `word`, используя функцию `substr()`;
 - используя функции `isupper()` и `tolower()`, привести слова к одному регистру;
 - используя оператор `switch` в зависимости от знака препинания (значения `tmp`), записать полученное слово в соответствующее множество;
 - увеличить `pos1`;
 - найти следующее слово;
 - увеличить `pos`;
 - найти следующее предложение;
- Создать вектор `res`, размер которого совпадает с размером множества `vopr`;
- Используя алгоритм `set_difference(vopr.begin(), vopr.end(), vosk.begin(), vosk.end(), res.begin())`, записать результат в вектор `res` и вывести на экран;

конец алгоритма

0.3 Пример работы с отображениями

Пример 0.3. Условие: Дан входной файл, содержащий слова и целые числа, разделенные пробелами. Вывести те числа, которые встречаются в файле меньше число раз, чем встречается первое слово. Решение: Для решения лучше всего воспользоваться контейнером `map<string, int>` `a`, где ключом являются слова, а значением — сколько раз они встречаются в тексте.

Так как `map` является ассоциативным контейнером, первое слово, встречающееся в файле, необязательно будет первым в контейнере, следовательно, его необходимо сохранить в отдельной переменной.

Воспользуемся `map` как ассоциативным массивом: `a[str]++`. Если слова `str` нет, оно будет добавлено в контейнер, `str` будет ключом, а значением — единица. Если слово уже есть, значение будет увеличено на единицу. В итоге получим контейнер, содержащий все слова (уникальные) задания и их количество.

В нашей задаче проще создать два контейнера, один из слов, другой из чисел (слов, состоящих только из цифр). Чтобы определить, число или слово считали, надо воспользоваться функцией `find_first_not_of(number)`, где `number` — строка, содержащая все цифры. Если результат работы этой функции `string::npos`, считано число (т. к. данная функция возвращает позицию первой буквы, не встречающейся в строке `number`).

□

Алгоритм 2: Работа с map

Вход: Файл, содержащий слова двух типов: состоящие только из цифр (целые числа) и остальные слова

Выход: числа, удовлетворяющие условию

начало алгоритма

- создаем 2 map<string, int> let, num;
- создаем строку, содержащую все цифры number;

цикл пока не дошли до конца файла выполнять

- считываем слово;

если считали число то

- └ · num[слово]++;

иначе

если let не содержит элементов то

- └ · firstWord = слово;

- └ · let[слово]++;

- int p = let[firstWord];

цикл для всех слов из let выполнять

если it->second < p то

- └ · вывести it->first;

конец алгоритма
