

Лабораторная работа № 2

Тема: Реализация множеств.**Цель:** ознакомиться с понятием «множества», изучить основные алгоритмы реализации множеств, научиться применять полученные знания на практике.

1 Краткая теория

1.1 Set и multiset

Что такое set и multiset?

set — это контейнер, который автоматически сортирует добавляемые элементы в порядке возрастания. Но при добавлении одинаковых значений, set будет хранить только один его экземпляр. По-другому его еще называют множеством.

112323	=>	123
122223	=>	123
332313	=>	123
123557	=>	12357

Добавление элементов в SET

multiset — это контейнер, который также будет содержать элементы в отсортированном порядке при добавлении, но он хранит повторяющиеся элементы, по сравнению с множеством set. Часто его называют мультимножеством.

112323	=>	112233
122223	=>	122223
332313	=>	123333
123557	=>	123557

Добавление элементов в MULTiset

Из-за того, что set построен на списках, нельзя обратиться к определенному элементу по индексу, как в массиве или векторе:

```
1 st[3] = 19; // ошибка
```

Для этого придется оперировать итераторами.

Как создать set и multiset?

Для использования, с самого начала нужно подключить единственную библиотеку — <set>.

```
1 #include <set>
```

Далее используем данную конструкцию:

```
1 set < [тип] > <имя>;
2 multiset < [тип] > <имя>
```

[тип] — это тип данных для нашего контейнера.

[имя] — название нашего контейнера.

```
1 set <int> st; // пример
2 multiset <int> mst;
```

Чтобы добавить новый элемент нужно использовать функцию *insert()*:

```
1 st.insert(<значение>);
```

Добавление происходит за $\log n$. n — это размер контейнера.

Возможно понадобится изменить сторону сортировки в обратную (по убыванию), для этого можно сделать с помощью *greater*, вот так:

```
1 set < [тип], greater [тип] > [имя];
2 set <long long, greater <long long> > st; // пример
```

В каждом из пунктов [тип] должен быть одинаковый.

Также можно добавлять значения при инициализации контейнеров:

```
1 set <int> st{1,2,3,4,5};
```

Но это можно делать только в C++ 11 и выше.

Итераторы для set и multiset

Вот как выглядит создание итератора:

```
1 set < [тип] > :: iterator it;
2 multiset < [тип] > :: iterator it2;
```

Чтобы итератор работал на определенный set или multiset, [тип] должен совпадать с типом контейнера.

Для использования итераторов подключение сторонних библиотек будет лишним. Итераторы уже включены в библиотеку — *<iostream>*. А если будут нужны, то они находятся в библиотеке — *<iterator>*.

При создании итератора для set и multiset нужно знать, что мы сможем только наблюдать за их значением. Это значит, что мы не сможем изменять значения существующих элементов.

Вот что можно делать с итератором на множество и мультимножество:

- Использование операции разыменования — ***.
- Сравнивать итераторы на равенство (*==*) и неравенство (*!=*).
- Увеличивать или уменьшать итератор, с помощью инкремента (*++*) и декремента (*--*).

Но также у него есть свои ограничения:

- Нельзя изменять итератор используя операции сложения (*+*), умножения (***), деления (*/*).

- Также нельзя сравнивать итераторы знаками больше (>) и меньше (<). Чтобы обратиться к значению элемента, нужно воспользоваться операцией разыменование *.

```

1 #include <iostream>
2 #include <set>
3 using namespace std;
4
5 int main() {
6     set <int> st;
7
8     for (int i = 0; i < 5; i++) {
9         st.insert(i + 1);
10    }
11
12    set <int> :: iterator it = st.begin();
13
14    cout << *it; // 1
15    it++;
16    cout << *it; // 2
17    return 0;
18 }
```

Также можно сравнивать итераторы на равенство (==) и неравенство (!=). Мы часто будем пользоваться данной возможностью. Например, чтобы вывести все элементы.

```

1 #include <iostream>
2 #include <ctime>
3 #include <set>
4
5 using namespace std;
6
7 int main() {
8     setlocale(LC_ALL, "Russian");
9     srand(time(NULL));
10
11    multiset <int> mst;
12
13    cout << "Добавление случайных значений: " << endl;
14    for (int i = 0; i < 10; i++) {
15        int random = rand() % 10 + 1;
16        mst.insert(random);
17        cout << i + 1 << " ) " << random << endl;
18    }
19
20    multiset <int> :: iterator it = mst.begin();
21
22    cout << "Отсортированный вариант: " << endl;
23    for (int i = 1; it != mst.end(); i++, it++) {
24        cout << *it << " ";
25    }
26
27    system("pause");
28    return 0;
29 }
```

- В строке 11: создали multiset — mst.
- В строках 14 — 18: заполняем мультимножество случайными значениями.
- В строке 20: инициализируем итератор на mst.
- В строках 22 — 25: выводим значения mst пользователю.

- В строке 23: мы сравнивали итераторы на неравенство (\neq), чтобы не выйти за диапазон значений контейнера и в последствии чтобы этот цикл не выполнялся бесконечно.

Вот пример данной программы:

```

==_and_!=_in_multiset.cpp

Добавление случайных чисел:
1) 7
2) 5
3) 1
4) 5
5) 9

Отсортированный вариант: 1 5 5 7 9
Process returned 0 (0x0) execution time : 0.010 s
Press any key to continue.

```

Методы для set и multiset

- **copy**

Сначала подключите библиотеку — `<iterator>`, она понадобится нам для использования — `ostream_iterator`.

Эта функция используется для различных операций. Одна из таких операций вывод элементов контейнера. Снизу находится конструкция вызова данной функции:

1 `copy` ([начала], [конец], `ostream_iterator`< [тип] >(`cout`, [отступ]));

- [начала] — итератор указывающий на первый элемент, который мы хотим вывести. Так мы можем начать выводить со второго или третьего элемента.
- [конец] — итератор указывающий на ячейку, до которой будет производиться вывод.
- [тип] — тип данных выводимых элементов (тип контейнера).
- <отступ> — здесь можно указать, что выводить между элементами. Обычно указывают пробел (`cout`, " ").

Вот пример использования `copy` для `set`:

```

1 #include <iostream>
2 #include <iterator> // ostream_iterator
3 #include <set>       // set
4 #include <ctime>     // time
5 using namespace std;
6
7 int main() {
8     setlocale(LC_ALL, "Russian");
9
10    set<int> st;
11
12    cout << "Введите 5 чисел: " << endl;
13
14    for (int i = 0; i < 5; i++) {
15        cout << i + 1 << " ) ";
16        int dig; cin >> dig;
17        st.insert(dig);
18    }
19
20    cout << "Содержимое set: ";
21
22    copy(st.begin(), st.end(), ostream_iterator<int>(cout, " "));
23
24    system("pause");
25    return 0;
26 }

```

- В строках 14 — 18: считываем значения пользователя.

- В строке 22: выводим все элементы контейнера `st`.

Вот как будет выглядеть выполненная программа:

```
copy_for_set.cpp
Введите 5 чисел:
1) 15
2) 19
3) 3
4) 12
5) 7
Содержимое set: 3 7 12 15 19
Process returned 0 (0x0) execution time : 0.010 s
Press any key to continue.
```

• **erase**

С помощью нее вы сможете:

1. Удалить какой-то конкретный элемент — `<имя>.erase([итератор])`
2. Удалить все элементы с данным значением — `<название>.erase([ключ])`. Это отличная функция для мультимножества. А для `set` мы как раз удалим конкретный элемент, это удобнее чем: найти итератор на определенное значение через функции `find` или `lower_bound`, а потом его удалить.
3. Либо удалить определенный диапазон значений.

```
1 <имя>.erase([начала], [конец]);
```

- `[начала]` — с какого элемента будет происходить удаление (включительно).
- `[конец]` — до какого элемента будет продолжаться (не включительно).

Пример:

```
1 mst.erase(3); // удалим все элементы со значением 3
```

• **lower_bound**

Часто приходится проверить есть ли элемент, который равен определенному ключу, либо больше его. Это функция находит элемент который больше или равен ключ (`<=key`).

```
1 <имя>.lower_bound(key);
```

- `key` — это наш ключ. Значение, с которым будут сравниваться элементы.

```
Значения в множестве: 1 3 7 8 9 11
lower_bound(3)    =>    3
lower_bound(5)    =>    7
lower_bound(10)   =>   11
```

upper_bound

Этот метод идентичен функции `lower_bound`:

- У него такая же конструкция вызова

- При поиске также используется бинарный поиск
Но искать она будет элемент, который именно больше ключа — >key.

Значения в мультимножестве: 3 5 5 7 9 10 11

upper_bound(3)	=>	5
upper_bound(7)	=>	9
upper_bound(10)	=>	11

- **find**

Сравнительно часто может пригодиться нахождение итератора на элемент, либо проверить существует ли он вообще.

```
1 find([ключ]);
```

Эта функция может вернуть:

- Местонахождение [ключа] — итератор.
- Значение на конец контейнера (оно будет равняться вызову st.end()), если ячейки с таким значением не существует.

Чтобы проверить есть ли данный элемент, можно воспользоваться данным кодом:

```
1 if ([контейнер].find([ключ]) == [контейнер].end()) {
2     cout << "Такого значение не существует";
3 }
```

Если условие верно, то такого элемента нет (он равен концу контейнера).
Вот пример:

```
1 set <int> numbers;
2
3 ... считывание значений для numbers ...
4
5 for (int i = 1; i <= 30; i++) {
6     dig = i;
7
8     if (numbers.find(dig) == numbers.end()) {
9         cout << "Такого значение " << dig << " не существует" << endl;
10    }
11 }
```

Выше мы проверяем, есть ли значения меньше 31 и больше 0 в контейнере. Если нет, то выводим это значение.

- **count**

Возвращает количество элементов с заданным значением.

```
1 st.count([ключ]);
```

Для обычного set эту функцию можно использовать, чтобы проверить существует ли заданное значение. А вот для multiset данная функция может пригодиться гораздо лучше из-за возможных повторяющихся значений.

- **equal_range**

Эта функция предназначена для `multiset`. Допустим вы добавили несколько одинаковых чисел, и вам требуется узнать: от куда этот диапазон начинается и до куда заканчивается.

```
1 <имя>.equal_range([ключ]);
```

- <имя> — название нашего мультимножества.

Возвращаемым значением этой функции будет — `pair`.

- В первой ячейке будет находится значение итератора, который указывает на начала этого диапазона. Если мы вызовем `lower_bound([ключ])` (ключ при этом одинаковый), то значение итераторов будут одинаковые. Найдет число $\geq key$.

- Во второй ячейке будет находится значение итератора, если бы мы вызвали `upper_bound([ключ])` (с тем же ключом). Найдет число $> key$.

Вот пример использования этой функции:

```
1 multiset <int> chisla;
2 multiset <int> :: iterator
3 pair <multiset <int> :: iterator,
4     multiset <int> :: iterator> it;
5
6 ... инициализация ... // 1 1 1 2 2 3 4 8 9 10 end()
7
8 it = chisla.equal_range(1);   ^   ^
9
10 it = chisla.equal_range(4);           ^ ^
11
12 chisla.insert(4); // 1 1 1 2 2 3 4 4 8 9 10 end()
13
14 it = chisla.equal_range(4);           ^ ^
15
16 it = chisla.equal_range(10);           ^^ ^
```

Плюсы и минусы использования `set` и `multiset`

Плюс: быстрая сортировка элементов.

Минус: нельзя обратиться к конкретной ячейке по индексу `[]`.

Если у вас мало обращений к определенным ячейкам, то использовать однозначно нужно.

Если же вам придется очень часто обращаться к произвольным ячейкам — то использовать лучше `vector` или массив, если это возможно.

Пример

Создать интерфейс приложения, который пользователь сможет использовать для регулирования папок. В итоге у нас должна получиться программа для настройки псевдо файловой системы.

Вот какой функционал она будет иметь:

1. Добавление новых элементов.
2. Удаление одного или сразу всех элементов со значением *value*.
3. Использование операции `lower_bound` для поиска папок.
4. Оперирование операцией `upper_bound` для поиска папок.

Для начала хотелось бы, чтобы пользователь уже имел какую-то файловую систему — так сказать «бэграунд».

```

1  cout << "Введите начальное количество файлов: ";
2  int n; cin >> n;
3
4  multiset <int> files;
5
6  for (int i = 0; i < n; i++) {
7      cout << i + 1 << ") Введите индекс папки для добавления: ";
8      int a; cin >> a;
9      files.insert(a);
10 }
```

- В строке 4: мы создали наше мультимножество.
- В строках 6 — 10: считываем начальные индексы папок.

Реализуем каждую операцию. Начнем с добавление нового элемента.

```

1  if (operation == "add" || operation == "+") {
2      cout << "Введите индекс новой папки: ";
3      int value; cin >> value;
4
5      files.insert(value);
6
7      cout << "Новые значения индексов: ";
8      copy(files.begin(), files.end(), ostream_iterator(cout, " "));
9
10     cout << endl;
11 }
```

Обратите внимание, что для каждой операции мы сделали длинную и короткую форму вызова.

Вторая операция будет удаление — это уже сложнее. Потому что у нас два вида выполнения данной функции: удаление одного элемента и удаление всех элементов с заданным ключом.

```

1  if (operation == "erase" || operation == "-") {
2      cout << "Укажите какой именно тип операции вам подходит: ";
3      string temp; cin >> temp;
4
5      if (temp == "one" || temp == "1") {
6          cout << "Введите значение: ";
7          int value; cin >> value;
8
9          multiset <int> :: iterator it = files.find(value);
10         if (it == files.end()) {
11             cout << "Такого индекса не существует!" << endl;
12             continue;
13         }
14
15         files.erase(it);
16     }
17
18     if (temp == "all" || temp == "*") {
19         cout << "Введите значение: "; int value; cin >> value;
20
21         multiset <int> :: iterator it = files.find(value);
22
23         if (!files.count(value)) {
24             cout << "Таких индексов не существует!" << endl;
25             continue;
26         }
27
28         files.erase(value);
29     }
30     cout << "Новые значения индексов: ";
31     copy(files.begin(), files.end(), ostream_iterator(cout, " ")); // выводим
32     cout << endl;
33 }
```


- В строке 3: считываем вид операции.
- В строках 5 — 15: выполняется операции с удалением одной ячейки.
- В строках 18 — 29: выполняется операции с удалением всех значений.

- В строках 10 и 23: реализованы условия для проверки существования данных значений в мультимножестве.

1. В строке 10 это реализовано с помощью функции *find()*. Использования *find()* необходимо в данном случае, чтобы потом удалить значение данного итератора.

2. В строке 23 это реализовано с помощью *count()*, потому что для удаление нам не понадобится на их итератор — *erase([значение])*.

Третья и четвертая операция похожие и просты. Вот реализация операции *lower_bound*:

```

1 if (operation == "lower_bound" || operation == ">=") {
2     cout << "Введите значение для поиска: ";
3     int value; cin >> value;
4
5     multiset <int> :: iterator it;
6     it = files.lower_bound(value); // получаем итератор
7
8     if (it == files.end()) { // проверяем существует ли данный элемент
9         cout << "Элемента >=" << value << " не существует!" << endl;
10        continue;
11    }
12    cout << *it << endl;
13 }
```

И последняя операция *upper_bound*:

```

1 f (operation == "upper_bound" || operation == ">") {
2     cout << "Введите значение для поиска: ";
3     int value; cin >> value;
4
5     multiset <int> :: iterator it;
6     it = files.upper_bound(value); // получаем местоположение
7
8     if (it == files.end()) { // проверяем существование
9         cout << "Элемента > " << value << " не существует!" << endl;
10        continue;
11    }
12
13    cout << *it << endl;
14 }
```

А вот полностью все приложение:

```

C++
1 #include <iostream>
2 #include <set>
3 #include <iterator>
4
5 using namespace std;
6
7 int main() {
8     setlocale(0, "");
9     cout << "Введите начальное количество файлов: ";
10    int n; cin >> n;
11
12    multiset <int> files;
13 }
```

```

14 for (int i = 0; i < n; i++) {
15     cout << i + 1 << " ) Введите индекс папки для добавления: ";
16     int a; cin >> a;
17     files.insert(a);
18 }
19
20 cout << endl << "Укажите количество операций: ";
21 q; cin >> q;
22
23 for (int i = 0; i < q; i++) {
24     cout << i + 1 << " ) ";
25     string operation; cin >> operation;
26
27     if (operation == "add" || operation == "+") {
28         cout << "Введите индекс новой папки: ";
29         int value; cin >> value;
30
31         files.insert(value);
32
33         cout << "Новые значения индексов: ";
34         copy(files.begin(), files.end(), ostream_iterator(cout, " "));
35
36         cout << endl;
37     }
38
39     if (operation == "erase" || operation == "-") {
40         cout << "Укажите какой именно тип операции вам подходит: ";
41         string temp; cin >> temp;
42
43         if (temp == "one" || temp == "1") {
44             cout << "Введите значение: ";
45             int value; cin >> value;
46
47             multiset<int>::iterator it = files.find(value);
48             if (it == files.end()) {
49                 cout << "Такого индекса не существует!" << endl;
50                 continue;
51             }
52
53             files.erase(it);
54         }
55
56         if (temp == "all" || temp == "*") {
57             cout << "Введите значение: ";
58             int value; cin >> value;
59
60             multiset<int>::iterator it = files.find(value);
61
62             if (!files.count(value)) {
63                 cout << "Таких индексов не существует!" << endl;
64                 continue;
65             }
66
67             files.erase(value);
68         }
69         cout << "Новые значения индексов: ";
70         copy(files.begin(), files.end(), ostream_iterator(cout, " "));
71         cout << endl;
72     }
73
74     if (operation == "lower_bound" || operation == ">=") {
75         cout << "Введите значение для поиска: ";
76         int value; cin >> value;
77

```

```

78     multiset<int> :: iterator it;
79     it = files.lower_bound(value);
80
81     if (it == files.end()) {
82         cout << "Элемента >= " << value << " не существует!" << endl;
83         continue;
84     }
85     cout << *it << endl;
86 }
87
88 if (operation == "upper_bound" || operation == ">") {
89     cout << "Введите значение для поиска: ";
90     int value; cin >> value;
91
92     multiset<int> :: iterator it;
93     it = files.upper_bound(value);
94
95     if (it == files.end()) {
96         cout << "Элемента > " << value << " не существует!" << endl;
97         continue;
98     }
99
100    cout << *it << endl;
101 }
102 }
103
104 system("pause");
105 return 0;
106}

```

Пример выполнения данной программы:

```

game_on_multiset.cpp
Введите начальное количество файлов: 5
1) Введите индекс папки для добавления: 1
2) Введите индекс папки для добавления: 2
3) Введите индекс папки для добавления: 3
4) Введите индекс папки для добавления: 5
5) Введите индекс папки для добавления: 7

Укажите количество операций: 5
1) +
Введите индекс новой папки: 3
Новые значения индексов: 1 2 3 3 5 7
2) -
Укажите какой именно тип операции вам подходит: all
Введите значение: 3
Новые значения индексов: 1 2 5 7
3) >=
Введите значение для поиска: 6
7
4) -
Укажите какой именно тип операции вам подходит: 1
Введите значение: 2
Новые значения индексов: 1 5 7
5) >
Введите значение для поиска: 7
Элемента > 7 не существует!

Process returned 0 (0x0) execution time : 0.010 s
Press any key to continue.

```

! Дополнительный материал !

1. Контейнеры: map, set, multiset:

<https://brestprog.by/topics/containers/>

2. Множества set и multiset в C++:

https://proproprogs.ru/structure_data/std-set-i-multiset-v-c

3. Алгоритмы и структуры данных: множества:
<https://tproger.ru/translations/sets-for-beginners/>
4. C++ STL: map и set:
<https://codeforces.com/blog/entry/9702>
5. Теория множеств: основы и базовые операции над множествами:
<https://ru.hexlet.io/blog/posts/teoriya-mnozhestv-osnovy-i-bazovye-operatsii-nad-mnozhestvami>

2 Задания

Обязательное: Необходимо создать *set* и сделать так, чтобы пользователь мог использовать отдельную операцию, введя определенный символ:

- Добавление — при вводе *push* и значение нового элемента.
- Удаление — при вводе *delete* и значение удаляемого элемента.

! Контрольные вопросы !

1. Определение понятия *set*.
2. Плюсы и минусы использования *set* и *multiset*.
3. Для чего используется функция *equal_range()*?
4. Какая функция позволяет удалить все элементы с данным значением в мультимножестве?
5. Определение понятия *multiset*.
6. Как создать *set*?
7. Что возвращает функция *count()*?
8. Как создать *multiset*?
9. В чем отличие функций *lower_bound()* и *upper_bound()*?
10. С помощью какой функции можно удалить определенный диапазон значений?

Содержание отчёта

1. Ф.И.О., группа, название лабораторной работы.
2. Цель работы.
3. Описание проделанной работы.
4. Результаты выполнения лабораторной работы.
5. Ответы на контрольные вопросы.
6. Выводы.