

Лабораторная работа № 21
Инструкционно-технологическая карта

Тема: Разработка различных по эффективности, применяемым средствам алгоритмов и программ с использованием множеств и регулярных выражений

Цель: научиться пользоваться множествами set и мультимножествами multiset, осуществлять поиск и замену участков символьных строк с помощью регулярных выражений

Время выполнения: 2 часа

1. Краткие теоретические сведения

Множества

В STL есть замечательный контейнер – set, он реализует такие сущности как множество и мультимножество. По сути, это контейнеры, которые содержат некоторое количество отсортированных элементов. Да, именно так, при добавлении нового элемента в множество он сразу становится на свое место так, чтобы не нарушать порядка сортировки. Потому как в множестве и мультимножестве все элементы сортируются автоматически. Но вот вопрос, в чем же разница между множествами и мультимножествами? Множества содержат только уникальные элементы, а мультимножества могут содержать дубликаты, вот такая вот небольшая разница.

Для того, чтобы использовать множество или мультимножество необходимо подключить следующий заголовочный файл:

```
#include <set>
```

Простой пример использования множества в программе:

```
#include <iostream>
#include <set> // заголовочный файл множеств и мультимножеств
#include <iterator>

using namespace std;
int main()
{
    set<char> mySet; // объявили пустое множество

    // добавляем элементы в множество
    mySet.insert('l');
    mySet.insert('n');
    mySet.insert('f');
    mySet.insert('i');
    mySet.insert('n');
    mySet.insert('i');
    mySet.insert('t');
    mySet.insert('y');

    copy(mySet.begin(), mySet.end(), ostream_iterator<char>(cout, " "));
    system("pause");
    return 0;
}
```

```
I f i n t y Для продолжения нажмите любую клавишу . . .
```

Разберёмся, почему же получился такой вывод программы. Во-первых, обратите внимание на порядок добавления элементов в коде, строки 11-18, и на реальный порядок расположения этих элементов в множестве. Порядок ввода и реальный порядок элементов в множестве – разный, это связано с тем, что элементы множества автоматически сортируются. Еще одной очень важной деталью является то, что в множество не сохранились дубликаты, хотя дубликаты были при добавлении элементов в множество. Как видно в выводе программы, каждый элемент множества уникален.

Мультимножества

Изменим тип данных на

```
multiset<char> mySet; // объявили пустое множество
```

```
I f i i n n t y Для продолжения нажмите любую клавишу . . .
```

Результат другой, так как мультимножество может хранить дубликаты элементов, все введенные буквы сохранились. Ну и конечно все элементы отсортировались, как и в множестве. Из всего этого стоит запомнить то, что порядок ввода элементов в множество никак не влияет на порядок хранения в множестве. А также, мультимножество умеет хранить дубликаты, в отличие от множества. Если вам нужно объявить множество, используйте класс set, если же вы хотите объявить мультимножество – воспользуйтесь классом multiset.

Автоматическая сортировка элементов в множествах накладывает определенные ограничения. Например, в множествах нельзя изменить значение какого-то элемента напрямую, так как это могло бы сломать сортировку. Поэтому, если вам нужно это сделать, вы сначала должны удалить старый элемент, а потом добавить новый. Рассмотрим пример программы с удалением и добавлением нового элемента:

```
#include <iostream>

#include <set> // заголовочный файл множеств и мультимножеств
#include <iterator>
#include <cstdlib>
#include <windows.h>

using namespace std;
int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    srand(time(NULL));
    set<int> mySet; // объявили пустое множество

    // добавляем элементы в множество
    for (int i = 0; i < 10; i++) {
        mySet.insert(rand() % 100);
    }

    cout << "Элементы множества: ";
    copy(mySet.begin(), mySet.end(), ostream_iterator<int>(cout, " "));

    int del = 0;
    cout << "\nКакой элемент удалить? ";
    cin >> del;
```

```

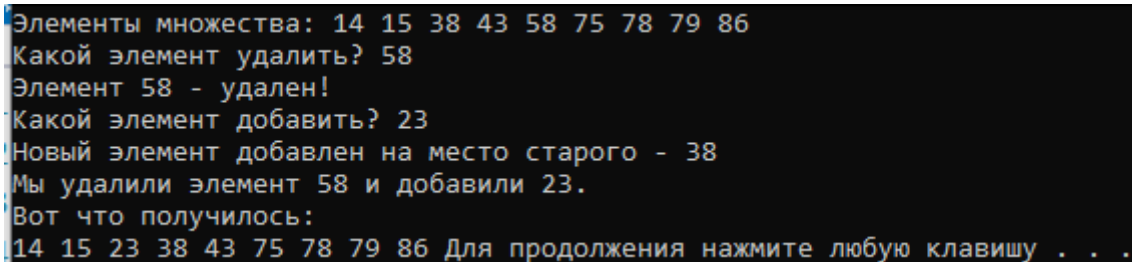
cout << "Элемент " << *mySet.find(del) << " - удален!" << endl;
mySet.erase(del);

int add = 0;
cout << "Какой элемент добавить? ";
cin >> add;

cout << "Новый элемент добавлен на место старого - " << *mySet.lower_bound(add) << endl;
mySet.insert(add);

cout << "Мы удалили элемент " << del << " и добавили " << add << ".\nВот что получилось: " <<
endl;
copy(mySet.begin(), mySet.end(), ostream_iterator<int>(cout, " "));
system("pause");
return 0;
}

```



```

Элементы множества: 14 15 38 43 58 75 78 79 86
Какой элемент удалить? 58
Элемент 58 - удален!
Какой элемент добавить? 23
Новый элемент добавлен на место старого - 38
Мы удалили элемент 58 и добавили 23.
Вот что получилось:
14 15 23 38 43 75 78 79 86 Для продолжения нажмите любую клавишу . . .

```

В этом примере мы сначала удаляем элемент множества, это делается в строке 29, с помощью метода `erase()`. Потом мы добавляем новый элемент, строка 36. Кроме этого, обратите внимание на метод – `find()`, строка 28, он возвращает указатель на первый элемент множества, который равен его аргументу. Еще один интересный метод, которым мы воспользовались в программе называется – `lower_bound()`, строка 35. Метод `lower_bound()` возвращает указатель на первый элемент множества, значение которого больше либо равно аргументу.

Работа с элементами set:

`S.swap(S2)` - меняет содержимое контейнеров местами,

`S.insert(a)` - вставка элемента `a`,

`S.erase(S2)` - удаляет последовательность элементов,

`S.clear()` - очистка контейнера `set`,

`S.count()` - количество элементов в контейнере,

`S.find(a)` - найти элемент `a` в контейнере,

`S.lower_bound(a)` - первый элемент, не меньший чем `a`,

`S.upper_bound(a)` - первый элемент, больший чем `a`,

`S.equal_range(a)` - пара элементов, первый - нижняя граница элементов с такими же значениями, что и `a`, второй - верхняя граница элементов с такими же значениями, что и `a`.

Работа с памятью:

`S.size()` - размер контейнера,

`S.max_size()` - максимальный размер контейнера.

Работа с контейнером:

`S.begin()` - указатель на начало контейнера,

`S.end()` - указатель на конец контейнера,

`S.rbegin()` - реверсивный указатель на конец контейнера,

`S.rend()` - реверсивный указатель на начало контейнера.

```

1 #include <iostream>
2 #include <set> //подключить библиотеку, для работы множеств set и multiset
3 using namespace std;
4 void showSet(set<int>&); //прототип новой функции
5
6 int main()
7 {
8     system("chcp 1251"); //подключение русского языка одной функцией (change code page 1251 - поменять кодовую страницу на №1251 (это кириллица в ОС MS Windows))
9     set<int> s; //создаю множество set для хранения целочисленных значений и даю ему имя s; в set значения сохранятся без повторов, а в multiset значения сохранятся все, в том числе повторяющиеся
10    cout << "Адрес множества s: " << &s << endl; //множество s создано нединамически, это имя, а не указатель, поэтому, чтобы узнать адрес множества, используем оператор взятия адреса &
11    if (s.empty() == true) //метод empty() проверяет множество s на пустоту и возвращает ИСТИНА, если множество пустое, или ЛОЖЬ, если множество НЕпустое
12    {
13        cout << "Пустое множество.\n";
14    }
15    else //иначе множество НЕпустое, т.е. в нем один или более элементов
16    {
17        cout << "НЕпустое множество.\n";
18    }
19    cout << "Размер множества (сколько в нем значений): " << s.size() << " штук.\n"; //максимальный размер множества (его максимальная ёмкость): " << s.max_size() << " штук.\n";
20    int n, x;
21    cout << "Сколько хотите сохранить значений в множество: ";
22    cin >> n;
23    for (int i = 0; i < n; i++)
24    {
25        cin >> x;
26        s.insert(x); //методом insert(x) помещаем во множество s значение x. Так можно заполнять множество до тех пор, пока не поместите в него max_size() штук значений
27    }
28    cout << "Печать содержимого множества s:\n";
29    for (set<int>::iterator i = s.begin(); i != s.end(); i++) //создаем в цикле локальный итератор-указатель на начало множества, сдвигаем его на размер int'a (4 байта), пока не достигнем адреса,
30    //следующего за множеством s
31    {
32        //метод end() возвращает указатель на воображаемый элемент, который следует сразу за последним элементом множества, то есть на адрес, находящийся сразу ЗА множеством
33        cout << "i << ' ' << &i << endl; //распечатаем значение по указателю. Адрес значения узнать нельзя, только само значение и адрес самого итератора (указателя) на значение, а поскольку итератор i
34        //один, то его адрес повторится (в нем меняются хранимые адреса значений, на которые он указывает, но сам он располагается в памяти неподвижно)
35    }
36    cout << "Какое значение искать: ";
37    cin >> x;
38    cout << "В этом множестве хранится " << s.count(x) << " штук значения " << x << endl; //метод count(x) подсчитывает, сколько значений x хранится в множестве. Поскольку во множестве set нет
39    //повторяющихся значений, то count() для set'a вернет или 0 (не нашло), или один (два и более быть не могут, в отличие от multiset'a)
40    set<int>::iterator i1; //итераторов можно создать много. Создаю итератор i1 на целочисленное множество, но пока ему ничего не присвоено
41    if ((i1 = s.find(x)) == s.end()) //присвоить итератору-указателю i1 указатель, который вернет метод поиска значения find(x), который возвращает указатель на найденный элемент, или на указатель end(),
42    //если нужного значения во множестве не найдет
43    {
44        cout << "Значения " << x << " в этом множестве нет.\n";
45    }
46    else
47    {
48        cout << "Значение " << *i1 << " есть в этом множестве.\n";
49        cout << "Вы искали значение: " << *s.find(x) << endl; //функция возвращает указатель, а мы возмем по нему значение оператором "звездочка" *. Но если значения не нашло, то вернет указатель на
50        //s.end(), а по нему взять значение нельзя - будет ошибка, поэтому делаем это в теле проверки, удостоверившись, что успешно нашли искомое значение
51    }
52    cout << "Найти в множестве значение, большее или равное: ";
53    cin >> n;
54    if (s.lower_bound(n) != s.end()) //метод lower_bound(n) возвращает указатель на значение, не меньшее искомого, т.е. большее либо равное искому, или вернет указатель на конец множества end(), если ничего
55    //подходящего не нашло
56    {
57        cout << "Найдено: " << *s.lower_bound(n) << endl; //распечатаем найденное значение, если знаем, что его нашли благодаря проверке if()
58    }
59    else //иначе значения не нашло, а по указателю end() ничего нет, поэтому нельзя брать значение оператором "звездочка" *
60    {
61        cout << "Нет такого.\n";
62    }
63    cout << "Найти в множестве значение, большее: ";
64    cin >> x;
65    if (s.upper_bound(x) != s.end()) //метод upper_bound(x) возвращает указатель на значение, строго большее искомого, или вернет указатель на конец множества end(), если ничего подходящего не нашло
66    {
67        cout << "Найдено: " << *s.upper_bound(x) << endl; //распечатаем найденное значение, если знаем, что его нашли благодаря проверке if()
68    }
69    else //иначе значения не нашло, а по указателю end() ничего нет, поэтому нельзя брать значение оператором "звездочка" *
70    {
71        cout << "Нет такого.\n";
72    }
73    if (s.lower_bound(n) != s.end() & s.upper_bound(x) != s.end()) //проверим, что ОБА граничных значения найдены
74    {
75        cout << "\nВсе значения, от не меньшего, чем " << n << " до равного " << x << ":\n"; //хотя s.upper_bound(x) должен вернуть строго большее значение, чем x, но ниже следующий метод copy() рассчитан
76        //принимать указатель на несуществующий элемент, а потому последний элемент не печатает
77        copy(s.lower_bound(n), s.upper_bound(x), ostream_iterator<int>(cout, "\n")); //функция copy() принимает указатель на начальный элемент, указатель на конечный элемент множества, итератор выходного
78        //потока, который будет целочисленные значения выводить на консоль, разделяя их символом "\n" (печатать в столбец). Метод copy() позволяет распечатать множество без явного использования цикла,
79        //хотя неважно внутри него угадывается цикл
80    }
81    cout << "Печать содержимого множества s:\n";
82    for (set<int>::iterator i = s.begin(); i != s.end(); i++) //создадим локальный итератор i, который пройдет по множеству от первого элемента до последнего
83    {
84        cout << "i << ' ' << &i << endl; //адрес значения узнать нельзя, только само значение и адрес самого итератора (указателя) на значение
85    }
86    set<int> s2; //создадим множество s2 для хранения целочисленных значений
87    for (set<int>::iterator i = s.begin(); i != s.end(); i++) //создадим локальный итератор i, который пройдет по множеству s от первого элемента до последнего
88    {
89        //каждое прочитанное значение *i увеличим в 10 раз и поместим результат в множество s2, поскольку в одном цикле удалять и добавлять значения в одно множество нельзя из-за автосортировки в нём
90        s2.insert(*i * 10); //удалять в цикле нельзя, так как set на ходу сортируется. Проще использовать другой set, помещая в него значения
91    }
92    s.swap(s2); //взаимно обменяем содержимое множеств s и s2 методом swap(). Их размеры НЕ обязаны совпадать
93    cout << "Печать содержимого множества s:\n";
94    for (set<int>::iterator i = s.begin(); i != s.end(); i++) //создадим локальный итератор i, который пройдет по множеству от первого элемента до последнего
95    {
96        cout << "i << ' ' << &i << endl; //адрес значения узнать нельзя, только само значение и адрес самого итератора (указателя) на значение
97    }
98    cout << "Печать содержимого множества s2:\n";
99    for (set<int>::iterator i = s2.begin(); i != s2.end(); i++) //создадим локальный итератор i, который пройдет по множеству от первого элемента до последнего
100    {
101        cout << "i << ' ' << &i << endl; //адрес значения узнать нельзя, только само значение и адрес самого итератора (указателя) на значение
102    }
103    cout << "Адрес множества s: " << &s << " Адрес множества s2: " << &s2 << endl; //адреса множеств разные, значит они не указывают на одну область оперативной памяти
104    showSet(s); //надлежит печатать s строк, чтобы распечатать множество? Напишем за main'ом функцию showSet(s) и её прототип перед main'ом и распечатаем любой целочисленный set кодом в одно слово
105    showSet(s2); //эта функция принимает ссылку на целочисленное множество (то есть знает адрес оригинала set из main'a), а итератор создается внутри функции
106    s.clear(); //очистить множество s от всех элементов
107    s = s2; //копируем все элементы из множества s2 в пустое множество s. Их размеры НЕ обязаны совпадать
108    cout << "Адрес множества s: " << &s << " Адрес множества s2: " << &s2 << endl; //адреса множеств разные, значит они не указывают на одну область оперативной памяти
109    cout << "Печать содержимого множества s:\n";
110    for (set<int>::iterator i = s.begin(); i != s.end(); i++) //создадим локальный итератор i, который пройдет по множеству от первого элемента до последнего
111    {
112        cout << "i << ' ' << &i << endl; //адрес значения узнать нельзя, только само значение и адрес самого итератора (указателя) на значение
113    }
114    cout << "Какое значение удалить из множества: ";
115    cin >> x;
116    s.erase(x); //удалить значение x из множества (если такое в нем найдётся)
117    cout << "Сколько в этом множестве всего хранится значений: " << s.size() << " штук.\n";
118    copy(s.begin(), s.end(), ostream_iterator<int>(cout, "\n")); //функция copy() принимает указатель на начало множества, указатель на конец множества, итератор выходного потока, который будет
119    //целочисленные значения выводить на консоль, разделяя их символом "\n". Метод copy() позволяет распечатать множество без явного использования цикла, хотя неважно внутри него угадывается цикл
120    cout << "Печать содержимого множества s:\n";
121    for (set<int>::iterator i = s.begin(); i != s.end(); i++) //создадим локальный итератор i, который пройдет по множеству от первого элемента до последнего
122    {
123        cout << "i << ' ' << &i << endl; //адрес значения узнать нельзя, только само значение и адрес самого итератора (указателя) на значение
124    }
125    cout << "Пройдем по множеству с конца:\n";
126    for (set<int>::reverse_iterator i = s.rbegin(); i != s.rend(); i++) //для прохода по множеству с конца нужно создать реверсный итератор, который инициализируем реверсным началом множества (rbegin()
127    //вернет указатель на последний элемент множества, и проходим по множеству с конца в начало, пока указатель не станет равным реверсному концу множества, возвращенному методом rend()- началу множества
128    {
129        cout << "i << ' ' << &i << endl;
130    }
131    s.clear(); //удалим все значения из множества s методом clear()

```

```

130 cout << "Множество очищено методом clear().\n";
131 cout << "Печать содержимого множества s:\n";
132 for (set<int>::iterator i = s.begin(); i != s.end(); i++)//создадим локальный итератор i, который пройдет по множеству от первого элемента до последнего
133 {
134     cout << *i << " " << &i << endl;//адрес значения узнать нельзя, только само значение и адрес самого итератора (указателя) на значение
135 }
136 cout << "Сколько в этом множестве всего хранится значений: " << s.size() << " штук.\n";
137 multiset<double> s3;//multiset создается и работает аналогично сету, методы для него такие же, как и для сета, но мультисет может хранить повторяющиеся значения. Автосортировка в обоих по умолчанию.
138 cout << "Максимально возможное количество значений в мультимножестве s2: " << s3.max_size() << ", но сейчас в нем значений: " << s3.size() << endl;
139 s3.clear();//очистим мультимножество s3, хотя оно и так пустое
140 system("pause");
141 return 0;
142 }
143 void showSet(set<int>& m)//напишем сами функцию, которая принимает ссылку на целочисленное множество и распечатает его настоящий адрес и значения всех элементов данного множества
144 {
145     //оригинал из main'a
146     cout << "Печать содержимого множества, находящегося по адресу: " << &m << ":\n";
147     for (set<int>::iterator i = m.begin(); i != m.end(); i++)
148     {
149         cout << *i << " ";
150     }
151     cout << endl;
152 }

```

Тестирование на позитивных кейсах (тестовых данных):

C:\ D:\2019\Labs\64\Debug\Lab8_5множество_set.exe

Текущая кодовая страница: 1251

Адрес множества s: 0000007FF1CFE338

Пустое множество.

Размер множества (сколько в нем значений): 0 штук.

Максимальный размер множества (его максимальная ёмкость): 576460752303423487 штук.

Сколько хотите сохранить значений в множество: 6

9

7

5

7

3

1

Печать содержимого множества s:

1 0000007FF1CFE3C8

3 0000007FF1CFE3C8

5 0000007FF1CFE3C8

7 0000007FF1CFE3C8

9 0000007FF1CFE3C8

Какое значение искать: 7

В этом множестве хранится 1 штук значения 7

Значение 7 есть в этом множестве.

Вы искали значение: 7

Найти в множестве значение, большее или равное: 3

Найдено: 3

Найти в множестве значение, большее: 7

Найдено: 9

Все значения, от не меньшего, чем 3 до равного 7:

3

5

7

Печать содержимого множества s:

1 0000007FF1CFE428

3 0000007FF1CFE428

5 0000007FF1CFE428

7 0000007FF1CFE428

9 0000007FF1CFE428

Печать содержимого множества s:

10 0000007FF1CFE4B8

30 0000007FF1CFE4B8

50 0000007FF1CFE4B8

70 0000007FF1CFE4B8

90 0000007FF1CFE4B8

Печать содержимого множества s2:

1 0000007FF1CFE4E8

3 0000007FF1CFE4E8

5 0000007FF1CFE4E8

7 0000007FF1CFE4E8

9 0000007FF1CFE4E8

Адрес множества s: 0000007FF1CFE338 Адрес множества s2: 0000007FF1CFE458

Печать содержимого множества, находящегося по адресу: 0000007FF1CFE338:

10

30

50

70

90

```

Печать содержимого множества, находящегося по адресу: 0000007FF1CFE458:
1 3 5 7 9
Адрес множества s: 0000007FF1CFE338 Адрес множества s2: 0000007FF1CFE458
Печать содержимого множества s:
1 0000007FF1CFE518
3 0000007FF1CFE518
5 0000007FF1CFE518
7 0000007FF1CFE518
9 0000007FF1CFE518
Какое значение удалить из множества: 5
Сколько в этом множестве всего хранится значений: 4 штук.
1
3
7
9
Печать содержимого множества s:
1 0000007FF1CFE548
3 0000007FF1CFE548
7 0000007FF1CFE548
9 0000007FF1CFE548
Пройдем по множеству с конца:
9 0000007FF1CFE578
7 0000007FF1CFE578
3 0000007FF1CFE578
1 0000007FF1CFE578
Множество очищено методом clear().
Печать содержимого множества s:
Сколько в этом множестве всего хранится значений: 0 штук.
Максимально возможное количество значений в мультимножестве s2: 461168601842738790, но сейчас в нем значений: 0
Для продолжения нажмите любую клавишу . . .

```

Регулярные выражения

regex101.com – лучший сайт для изучения регулярных выражений.

Регулярные выражения представляют собой своеобразный язык описания строк. При этом, как и в любом языке, в нем есть определенные синтаксические конструкции и правила.

Регулярные выражения представляют собой похожий, но гораздо более сильный инструмент для поиска строк, проверки их на соответствие какому-либо шаблону и другой подобной работы.

Как и любой язык, регексы имеют спецсимволы, которые нужно экранировать. Вот их список: `^ $ * + ? { } [] \ | ()`. Экранирование осуществляется обычным способом – добавлением `\` перед спецсимволом.

Предположим, мы хотим найти в тексте все междометия, обозначающие смех. Просто Хаха нам не подойдет – ведь под него не попадут «Хехе», «Хохо» и «Хихи». Да и проблему с регистром первой буквы нужно как-то решить.

Здесь нам на помощь придут наборы – вместо указания конкретного символа, мы можем записать целый список, и, если в исследуемой строке на указанном месте будет стоять любой из перечисленных символов, строка будет считаться подходящей. Наборы записываются в квадратных скобках – паттерну `[abcd]` будет соответствовать любой из символов «a», «b», «c» или «d».

Диапазоны

Может возникнуть необходимость обозначить набор, в который входят буквы, например, от «б» до «ф». Вместо того, чтобы писать `[бвгдежзиклмнопрстуф]` можно воспользоваться механизмом диапазонов и написать `[б-ф]`. Так, паттерну `x[0-8A-F][0-8A-F]` соответствует строка «xA6», но не соответствует «xb9» (во-первых, из-за того, что в диапазоне указаны только заглавные буквы, во-вторых, из-за того, что 9 не входит в промежуток 0-8).

Механизм диапазонов особенно актуален для русского языка, ведь для него нет конструкции, аналогичной `\w`. Чтобы обозначить все буквы русского алфавита, можно использовать паттерн `[а-яА-ЯёЁ]`. Обратите внимание, что буква «ё» не включается в общий диапазон букв, и её нужно указывать отдельно.

Квантификаторы

Вернёмся к нашему примеру. Что, если в «смеющемся» междометии будет больше одной гласной между буквами «х», например, «Хаахаааа»? Наша старая регулярка уже не сможет нам помочь. Здесь нам придётся воспользоваться квантификаторами.

Квантификатор	Число повторений	Пример	Подходящие строки
{n}	Ровно n раз	Xa{3}xa	Хаааха
{m,n}	От m до n включительно	Xa{2,4}xa	Хаа, Хааа, Хааааха
{m,}	Не менее m	Xa{2,}xa	Хааха, Хаааха, Хааааха и т. д.
{,n}	Не более n	Xa{,3}xa	Хха, Хаха, Хааха, Хаааха

Обратите внимание, что квантификатор применяется только к символу, который стоит перед ним.

Некоторые часто используемые конструкции получили в языке RegEx специальные обозначения:

Квантификатор	Аналог	Значение
?	{0,1}	Ноль или одно вхождение
*	{0,}	Ноль или более
+	{1,}	Одно или более

Таким образом, с помощью квантификаторов мы можем улучшить наш шаблон для междометий до `[Xx][аоеи]+х[аоеи]*`, и он сможет распознавать строки «Хааха», «хееееех» и «Хихии».

Скобочные группы

Для нашего шаблона «смеющегося» междометия осталась самая малость – учесть, что буква «х» может встречаться более одного раза, например, «Хахахахааахахооо», а может и вовсе заканчиваться на букве «х». Вероятно, здесь нужно применить квантификатор для группы `[аоеи]+х`, но если мы просто напишем `[аоеи]х+`, то квантификатор `+` будет относиться только к символу «х», а не ко всему выражению. Чтобы это исправить, выражение нужно взять в круглые скобки: `([аоеи]х)+`.

Пример

Определить, является ли строка адресом e-mail.

```

#include <iostream>
#include <regex> //подключить для работы регулярных выражений
using namespace std;

int main()
{
    system("chcp 1251");
    char st[80];
    cout << "\nВведите email: "; //например, на английском "ivan_petrov2021@gmail.com"
    cin.getline(st, 256);
    regex reg("\\w+\\@+\\w+\\.\\w+"); //создаем конструктором регулярное выражение, даем ему
    //имя reg и передаем ему "формулу" для английского текста для поиска, то есть наше
    //регулярное выражение не воспримет кириллицу, знак тире- и иные символы препинания,
    //но воспримет нижнее подчеркивание, а также если адрес почты будет встроен в адрес другой почты
    if (regex_match(st, reg)) //возвращает true - если найдет, и false - если НЕ найдет
    {
        cout << "Email правильный.\n";
    }
    else
    {
        cout << "Email НЕправильный.\n";
    }
    main(); //проверить другой e-мейл
    system("pause");
    return 0;
}

```

D:\2019\Labs\64\Debug\Lab8_gregex.exe

Текущая кодовая страница: 1251

Введите email: ivan_petrov2021@gmail.com
Email правильный.
Текущая кодовая страница: 1251

Введите email: p8o9i8u7y6_rt65t@dev.by
Email правильный.
Текущая кодовая страница: 1251

Введите email: v6c5@narod.ru
Email правильный.
Текущая кодовая страница: 1251

Введите email: иван@gmail.com
Email НЕправильный.
Текущая кодовая страница: 1251

Введите email: asd-fgh@yahoo.com
Email НЕправильный.
Текущая кодовая страница: 1251

Введите email: qwe@rty.uio@pasd@fghj987_8.com4
Email правильный.
Текущая кодовая страница: 1251

Введите email: awe,tyuu@dev.by
Email НЕправильный.
Текущая кодовая страница: 1251

Проблемы не найдены.

Названия функций, которые работают с регулярными выражениями и используются для достижения различных целей (поиска, замены, обмена и т.д.):

Имя	Описание
regex_match	Проверяет, совпадает ли регулярное выражение со всей целевой строкой.
regex_replace	Заменяет соответствующие регулярные выражения.

Имя	Описание
regex_search	Поиск соответствия регулярному выражению.
swap	Меняет местами <code>basic_regex</code> или два <code>match_results</code> объекта.

Порядок выполнения работы

1. Изучить теоретические сведения к лабораторной работе.
2. Реализовать алгоритм решения задачи.
3. Отлаженную, работающую программу сдать преподавателю. Работу программы показать с помощью самостоятельно разработанных тестов.
4. Ответить на контрольные вопросы.

Варианты индивидуальных заданий:

Задание 1. Написать функцию, возвращающую значение логического типа, определяющую, является ли входная строка правильной (подходящей):

1. Дана строка «aAXa aeffa aGha aza ax23a a3sSa». Напишите регулярное выражение, которое найдет строки следующего вида: по краям стоят буквы 'a', а между ними – маленькие латинские буквы, не затронув остальных.

2. Дана строка «aAXa aeffa aGha aza ax23a a3sSa». Напишите регулярное выражение, которое найдет строки следующего вида: по краям стоят буквы 'a', а между ними – маленькие и большие латинские буквы, не затронув остальных.

3. Дана строка «aAXa aeffa aGha aza ax23a a3sSa». Напишите регулярное выражение, которое найдет строки следующего вида: по краям стоят буквы 'a', а между ними – маленькие латинские буквы и цифры, не затронув остальных.

4. Дана строка «aaa bbb ёёё ззз ййй ААА БББ ЁЁЁ ЗЗЗ ЙЙЙ». Напишите регулярное выражение, которое найдет все слова по шаблону: любая кириллическая буква любое количество раз.

5. Дана строка «aaa aaa aaa». Напишите регулярное выражение, которое заменит первое «aaa» на «!».

6. Дана строка «aeaaa aeaa aeaa aha axxa axxxa». Напишите регулярное выражение, которое найдет строки следующего вида: по краям стоят буквы 'a', а между ними – или буква 'e' любое количество раз или по краям стоят буквы 'a', а между ними – буква 'x' любое количество раз.

7. Дана строка «aeaaa aeaa aeaa aha axxa axxxa». Напишите регулярное выражение, которое найдет строки следующего вида: по краям стоят буквы 'a', а между ними – или буква 'e' два раза или буква 'x' любое количество раз.

8. Дана строка «a\ a abc». Напишите регулярное выражение, которое заменит строку «a\ a» на «!».

9. Дана строка «a\ a \\a a\\a». Напишите регулярное выражение, которое заменит строку «a\\a» на «!».

Задание 2.

1. Существует множество A и B. Вывести те числа, которые есть в обоих множествах.
2. Существует множество A и B. Вывести те числа, которые есть во множестве A, но нет во множестве B.
3. Существует множество A и B. Вывести те числа, которые есть во множестве B, но которых нет во множестве A.
4. Существует множество A и B. Вывести те числа, которые есть во множестве A, но которых нет во множестве B.
5. Определить, на какой позиции во множестве A находится число C. Если такого числа нет, вывести 0.
6. Определить, есть ли во множестве число, которое соответствует размеру этого множества.

7. Существует множество A и B. Найти во множестве A число, большее чем введённое C и определить, есть ли это число во множестве B.
8. Существует множество A и B. Найти в обоих множествах числа, следующие за введённым C и определить, совпадают ли они.
9. Вводится строка S. При помощи множества определить, есть ли в строке S повторяющиеся символы.
10. Дан двумерный целочисленный массив. Определить первую его строку, все элементы которой различны используя множества.

Контрольные вопросы

1. Для чего используются регулярные выражения?
2. Приведите пример регулярного выражения в C++.
3. Для чего нужны множества?
4. В чём отличие множества от мультимножества?

Литература

1. **Лоспинозо, Д.** C++ для профи. / Д. Лоспиназо. СПб.: Питер, 2021. –816 с.
2. **Окулов, С.М.** Программирование в алгоритмах / С. М. Окулов. – 7-е изд., электрон. –М.: Лаборатория знаний, 2021.– 386 с.
3. **Бхаргава, А.** Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. / А. Бхарова СПб.: Питер, 2017. – 288 с.
4. **Конова, Е. А.** Алгоритмы и программы. Язык C++: Учебное пособие – 2 е изд., стер. / Е.А. Конова., Г.А. Поллак. – СПб.: Издательство «Лань», 2017 – 384 с.
5. **Зиборов, В.В.** MS Visual C++ 2015 в среде .NET. Библиотека программиста. / В.В. Зиборов – СПб.: Питер, 2017 — 320 с.: ил.

Преподаватель

Рогалевич А.В.
Шаляпин Ю.В.