Практикум по курсам
"Основы информатики"и "Алгоритмы и структуры данных".
Лабораторная работа №13 на 2011/12 уч. год: 8 факультет, 1 курс, 1 семестр.
Множества

Поскольку множества — важнейшие объекты математики, то их реализация — элемент математической культуры языка программирования.

В языке Паскаль существует простая и вполне математическая реализация множеств — предопределённый тип **set**, — которая, к сожалению, не позволяет реализовать множество со структурными или строковыми компонентами иначе как гёделевской нумерацией!

Множество удобно отображать на битовую шкалу машинного слова, что, однако, не позволяет реализовать даже **set of char**. Реализации, опирающиеся на аппаратную поддержку строк в CISC-архитектурах, допускают множества значительно большей мощности. В языке Си типа множество нет, поскольку работа с множествами легко реализуется битовыми операциями языка. Множество задаётся битовой маской — набором флагов-битов, где значение флага (1 или 0) обозначает наличие или отсутствие соответствующего элемента в множестве.

Например, опишем множество букв латинского алфавита. Таких букв всего 26, значит битовая маска для нашего множества поместится в тип int (лучше использовать беззнаковый тип unsigned int).

Устройство битовой маски:

0 =	00000000000000000000000000000000	пустое множество
1 =	00000000000000000000000000000001	{ 'a' }
4 =	00000000000000000000000000000100	{ 'c' }
11 =	00000000000000000000000000001011	{ 'a', 'b', 'd' }
67108863 =	00000011111111111111111111111111	весь латинский алфавит

Получить множество, состоящее из одной буквы, можно так:

```
unsigned int set = 1u << (c - 'a');
```

где c — буква, включаемая в множество, а 1u — константа 1 типа unsigned int.

Проверить множество на пустоту можно просто сравнив его с нулём.

Добавить один элемент в множество можно, объединив его с множеством, состоящим из одного элемента.

Операция объединения множеств тождественна побитной дизъюнкции ("или") двух соответствующих битовых масок:

```
united_set = set_1 | set_2;
```

Пересечение множеств — побитная конъюнкция ("и"):

```
intersected_set = set_1 & set_2;
```

Если множество задаётся битовой маской, можно явно вычислить его дополнение с помощью оператора побитного отрицания ("не"):

```
complementary_set = ~set; // поразрядное отрицание
```

Разность множеств — это пересечение первого с дополнением второго:

```
set1_without_set2 = set_1 & ~set_2;
```

Соответственно, удалить элемент из множества можно так:

```
set_without_c = set & ~(1u << (c-'a'));
```

Рассмотрим программу построения и распечатки множества гласных букв входного потока:

```
#include <stdio.h>
#include <ctype.h> // для функции tolower(c)

// множество гласных ~--- обьединение множеств, состоящих из одной гласной
#define VOWELS (1u<<('a'-'a') | 1u<<('e'-'a') |1u<<('i'-'a') | 1u<<('o'-'a') | 1u<<('u'-'a'))

// функция порождения множества, состоящего из заданной буквы
unsigned int char_to_set(char c) {
    c = tolower(c); // регистр не учитываем
    if(c < 'a' || c > 'z') {
        // если это не буква - возвращается пустое множество
        return 0;
    } else {
        return 1u<<(c-'a');
    }
}

int main() {
    char alpha;
    int c;
    unsigned int letters_set = 0; // множество литер входного потока

    // чтение литер, пока не встретится EOF
    while((c = getchar()) != EOF) {
        // добавление считанного знака к множеству
        letters_set = letters_set | char_to_set(c);
    }

    // пересечение множества литер входного потока и множества гласных букв
    letters_set = letters_set & VOWELS;

    // перебор всего алфавита для распечатки полученного множества
    for(alpha = 'a'; alpha <= 'z'; alpha++) {
        // печать очередной буквы алфавита, если она входит в полученное множество
        if(letters_set & char_to_set(alpha) != 0) {
            printf("%c", alpha);
        }
    }
    printf("\n");
}
```

Рассмотренного представления множеств часто вполне достаточно для решения многих практических задач обработки текстов, особенно в связи с тем, что 64-разрядные целые на современных процессорах имеют аппаратную реализацию. Ниже приведена более совершенная реализация множеств на Си с размером битовой маски 256 (таким образом, в множество может быть помещен любой знак из ASCII-таблицы):

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <ctype.h>

typedef unsigned set_data_elem;
enum {
  bits_per_char = 8,
  bits_per_elem = sizeof(set_data_elem) * bits_per_char,
  datalen = (1 << bits_per_char) / bits_per_elem
};

typedef struct {
```

```
  set_data_elem data[datalen];
} set;

void set_clear(set *s)
{
  memset(s->data, 0, sizeof(s->data));
}

void set_insert(set *s, int c)
{
  s->data[c / bits_per_elem] |= 1u << (c % bits_per_elem);
}

void set_generate(set *s, bool indicator(int))
{
  set_clear(s);
  for (int i = 0; i != 1 << bits_per_char; ++i)
    if (indicator(i)) set_insert(s, i);
}

void set_erease(set *s, int c)
{
  s->data[c / bits_per_elem] &= ~(1u << c % bits_per_elem);
}

bool set_in(const set *s, int c)
{
  return (s->data[c / bits_per_elem] & (1u << c % bits_per_elem)) != 0;
}

int set_size(const set *s)
{
  int size = 0;
  for (int i = 0; i != 1 << bits_per_char; ++i)
    if (set_in(s, i)) ++size;
  return size;
}

bool set_equal(const set *s1, const set *s2)
{
  for (int i = 0; i != datalen; ++i)
    if (s1->data[i] != s2->data[i]) return false;
  return true;
}

bool set_includes(const set *s1, const set *s2)
{
  for (int i = 0; i != datalen; ++i)
    if ((s1->data[i] | s2->data[i]) != s1->data[i]) return false;
  return true;
}

set set_union(const set *s1, const set *s2)
{
  set result;
  for (int i = 0; i != datalen; ++i)
    result.data[i] = s1->data[i] | s2->data[i];
  return result;
}
```

```
set set_intersection(const set *s1, const set *s2)
{
  set result;
  for (int i = 0; i != datalen; ++i)
    result.data[i] = s1->data[i] & s2->data[i];
  return result;
}

set set_difference(const set *s1, const set *s2)
{
  set result;
  for (int i = 0; i != datalen; ++i)
    result.data[i] = s1->data[i] & ~(s2->data[i]);
  return result;
}

set set_symmetric_difference(const set *s1, const set *s2)
{
  set result;
  for (int i = 0; i != datalen; ++i)
    result.data[i] = s1->data[i] ^ s2->data[i];
  return result;
}

bool is_alpha(int c) { return isalpha(c); }
bool is_digit(int c) { return isdigit(c); }

int main()
{
  set s1, s2;
  set_clear(&s1);
  set_generate(&s1, is_alpha);
  set_generate(&s2, is_digit);
  set_insert(&s2, 'a');
  s1 = set_symmetric_difference(&s1, &s2);
  for (int i = 0; i != 1 << bits_per_char; ++i) {
    if (set_in(&s1, i))
      printf("%d belongs set\n", i);
    else
      printf("%d doesn't belong set\n", i);
  }
  printf("size is %d\n", set_size(&s2));
}
```

Полноценные множества присутствуют в более совершенных языках программирования. В C++ имеются шаблоны std::set и std::multiset для множеств и мультимножеств соответственно, реализованные с помощью т. н. красно-черных деревьев. Для множества битов-флагов, реализованных выше на Си, в C++ определен шаблон std::bitset. В Python имеются типы set для обычных множеств и frozenset для неизменяемых множеств, т. е. функциональность на уровне STL. Множества также могут быть реализованы вручную на базе ассоциативных контейнеров (словарей), а в Java — также и на базе встроенных хэш-функций. В стандартной библиотеке Java имеется встроенная реализация множеств в виде класса HashSet, позволяющего добавлять элементы в множество, удалять их из него, производить объединение множеств и вычислять их пересечения.

Переведём первый пример на язык Java:

```
import java.io.IOException;
import java.util.HashSet;
```

```
public class SetDemo {
    public static void main(String[] args) throws IOException {
        HashSet<Character> vowels = new HashSet<Character>();
        vowels.add('a'); vowels.add('e'); vowels.add('i'); vowels.add('o'); vowels.add('u');

        HashSet<Character> chars = new HashSet<Character>();

        int c;
        // заполнение массива chars всеми литерами, прочитанными с входного потока
        while((c = System.in.read()) != -1) {
            chars.add((char)c);
        }

        // пересечение множеств прямо по месту
        chars.retainAll(vowels);

        // проход по коллекции (в т.ч. множеству) - конструкция языка Java
        for(char ch : chars) {
            System.out.print(ch);
        }
        System.out.println();
    }
}
```

Приведём пример манипуляций с множествами на C++:

```
// Find all words contain vowels 'y', 'o', 'u' only.
#include <algorithm>
#include <iostream>
#include <iterator>
#include <set>
#include <string>

typedef std::set<char> charset;

const std::string sVowels = "aeiouy";
charset vowels(sVowels.begin(), sVowels.end());
const std::string sGoodVowels = "you";
charset goodVowels(sGoodVowels.begin(), aGoodVowels.end());

bool bad_vowel(const char c)
{
  return (vowels.find(c) != vowels.end()) && (goodVowels.find(c) == goodVowels.end());
}

void check_word(const std::string& s)
{
  std::cout << s << (std::find_if(s.begin(), s.end(), bad_vowel) == s.end() ? "␣OK" : "␣FAILED") << "\n
      ";
}

int main()
{
  typedef std::istream_iterator<std::string> iter;
  std::for_each(iter(std::cin), iter(), check_word);
}
```

В стуктурированном языке запросов SQL тоже есть операции над множествами, поскольку таблицы БД являются множествами.

Наконец, существует язык программирования SETL, базирующийся на множествах!

Сопоставим средства работы с множествами в различных языках программирования.

	Математика	**C (set by AVL)**	**Паскаль**
	TEX	**C++**	**SETL**
Пустое множество	$\varnothing$	set_clear(&s)	[ ]
	$\varnothing$	set<> s	[ ]
Мощность	$\#A$	set_size(&A)	
	$\#A$	A.size()	#A
Равенство	$A = B$	set_equal(&A, &B)	A = B
	$A = B$	A == B	A = B
Подмножество	$A \subseteq B$	set_includes(&A, &B)	A <= B
	$A \subseteq B$	includes(A, B)	A subset B
Надмножество	$A \supseteq B$	set_includes(&B, &A)	A >= B
	$A \supseteq B$	includes(B, A)	A incs B
Включение элемента	$a \in A$	set_in(&A, a)	a in A
	$a \in A$	A.find(a) != A.end()	a in A
Объединение	$A \cup B$	set_union(&A, &B)	A + B
	$A \cup B$	set_union(A, B)	A + B
Пересечение	$A \cap B$	set_intersection(&A, &B)	A * B
	$A \cap B$	set_intersection(A, B)	A * B
Разность	$A \setminus B$	set_difference(&A, &B)	A − B
	$A \setminus B$	set_difference(A, B)	A − B
Симметрическая разность	$A \triangle B$	set_symmetric_difference(&A, &B)	(A − B) + (B − A)
	$A \triangle B$	set_symmetric_difference(A, B)	(A − B) + (B − A)

Постановка задачи Входные строки представляют собой последовательности слов, разделенных пробелами, запятыми, табуляциями или границами строк. В соответствии с вариантом задания составить программу проверки характеристик введенных последовательностей слов и печати развернутого ответа. Тестирование проводить не менее чем на трех строках вплоть до конца входного файла. В качестве алфавита берется один из европейских алфавитов, соответствующих заданию (кириллица, греческий, латиница, . . .). При использовании русских букв необходимо учитывать особенности их кодировки на разных платформах и лексические стандарты конкретных систем программирования.

Варианты заданий

1, 2. Есть ли слово, все гласные (согласные) которого различны?

3, 4. Есть ли слово, хотя бы одна гласная (согласная) которого повторяется?

5, 6. Есть ли слова, содержащие гласные только 1-го (2-го) рода (*аоуэыи* и *яёюе*)?

7, 8. Есть ли слова, начинающиеся и заканчивающиеся гласными (согласными)?

9, 10. Есть ли соседние слова, состоящие из одного и того же (разных) набора(ов) букв?

11, 12. Есть ли гласная (согласная), входящая в состав всех слов?

13, 14. Есть ли гласная (согласная), не входящая ни в одно слово?

15, 16. Есть ли два соседних слова, гласные (согласные) в которых совпадают?

17, 18. Есть ли два соседних слова с неперескающимся набором гласных (согласных)?

19, 20. Есть ли слово, содержащее одну гласную (согласную), возможно несколько раз?

21, 22. Есть ли слово, содержащее ровно одну гласную (согласную)?

23–26. Есть ли слово, все согласные которого — звонкие (глухие, шипящие, свистящие).

27. Есть ли слово, содержащее все гласные алфавита?

28. Есть ли слово, состоящее только из гласных?

29. Есть ли слово, содержащее более одной прописной буквы?

Задание подготовили: Зайцев В. Е., Дубинин А. В., Лебедев А. В. и Перетягин И. А.