**Задание 3**

Программно реализовать операции над множествами, используя следующие способы представления множества в памяти ЭВМ:

а)элементы множества *А* хранятся в массиве *А*.Элементы массива *А* неупорядочены;

б) элементы множества *А* хранятся в массиве *А*.Элементы массива *А* упорядочены по возрастанию;

в)элементы множества *А* хранятся в массиве *А*, элементы которого типа boolean. Если *i* ***∈*** *А*, то *Аi* = *true*,иначе *Ai* = *false*.

**Выполнение**

Операции над множествами

1. Включение *А* и *В* (*A ⊆ В*). Истинно, если каждый элемент множества *А* принадлежит множеству В.
2. Равенство *А* и *В* (*А = В*). Множества *А* и *В* равны, если *А ⊆ В* и *В ⊆ А*.
3. Строгое включение *А* и *В* (*А ⊂ В*). Результат истина, если истинно включение и ложно равенство, т.е. *А ⊆ В* и *А ≠ В*.
4. Объединение *А* и *В* (*А* ∪ *В*) – множество, состоящее из элементов, которые принадлежат *А* или *В*, т.е. *А* ∪ *В*={x|x *∈ A* или x *∈ B*}.
5. Пересечение *А* и *В* (*А* ∩ *В*) – множество, состоящее из элементов, которые принадлежат каждому из множеств *А* и *В*, т.е. *А* ∩ *В* = {x|x *∈ A* и x *∈ B*}.
6. Разность *А* и *В* (*А* – *В*) – множество, состоящее из элементов множества *А*, которые не принадлежат множеству *В*, т.е. *А* – *В* = {x|x *∈ A* и x *∉ B*}.
7. Симметрическая разность *А* и *В* (*А* Δ *В*) – множество, состоящее из элементов множества *А*, которые не принадлежат множеству *В*, и элементов множества *В*, которые не принадлежат множеству *А*, т.е. *А* Δ *В* = {x|x *∈ A* и x *∉ B* или x *∈ B* и x *∉ А*}.

*А* Δ *В*= (*А* – *В*) ∪ (*В* – *А*)

1. Дополнение *А* до универсума *U*. () – множество, состоящее из элементов универсума, которые не принадлежат множеству А, т.е. ={x|x *∉ А*},

**Описание спецификаций подпрограмм**

Подпрограммы для выполнения операций 1-3 (для случаев а-в):

* Включение

Функция inclusion

1. Заголовок: **function** inclusion (a, b: array of byte; n, m: byte): boolean;

2. Назначение: возвращает *true*, если множество *А* включено в множество *В*, иначе – *false*;

3. Входные параметры: a, b, n, m;

4. Выходные параметры: нет.

* Равенство

Функция equality

1. Заголовок: **function** equality (a, b: array of byte; n, m: byte): boolean;

2. Назначение: возвращает *true*,если множества *A* и *B* равны, в противном случае – *false*;

3. Входные параметры: a, b, n, m;

4. Выходные параметры: нет.

* Строгое включение

Функция strict\_inclusion

1. Заголовок: **function** strict\_inclusion (a, b: array of byte; n, m: byte): boolean;

2. Назначение: возвращает *true*, если множество *А* строго включено в множество *В*, иначе – *false*;

3. Входные параметры: a, b, n, m;

4. Выходные параметры: нет.

Подпрограммы для выполнения операций 4-8 (для случаи а-б):

* Объединение

Функция unification

1.Заголовок: **function** unification (a, b: array of byte; n, m: byte; var c: array of byte): byte;

2. Назначение: в массив *С* записывает результат объединения множеств *А* и *В* и возвращает длину множества *C* (мощность множества *С*);

3.Входные параметры: a, b, n, m;

4.Выходные параметры: C.

* Пересечение

Функция intersection

1.Заголовок: **function** intersection (a, b: array of byte; n, m: byte; var c: array of byte): byte;

2. Назначение: в массив *С* записывает результат пересечения множеств *А* и *В* и возвращает длину множества *C* (мощность множества *С*);

3.Входные параметры: a, b, n, m;

4.Выходные параметры: c.

* Разность

Функция difference

1.Заголовок:**function** difference (a, b: array of byte; n, m: byte; var c: array of byte): byte;

2. Назначение: в массив *С* записывает результат разности множеств *А* и *В* и возвращает длину множества C(мощность множества С);

3.Входные параметры: a, b, n, m;

4.Выходные параметры: c.

* Симметрическая разность

Функция symmetric\_difference

1.Заголовок: **function** symmetric\_difference (a, b: array of byte; n, m: byte; var c: array of byte): byte;

2. Назначение: в массив *С* записывает результат симметрической разности множеств *А* и *В* и возвращает длину множества *C* (мощность множества *С*);

3.Входные параметры: a, b, n, m;

4.Выходные параметры: c.

* Дополнение

Функция complement

1.Заголовок: **function** complement(a: array of byte; n: byte; var c: array of byte): byte;

2. Назначение: в массив *С* записывает результат дополнения множества *А* до универсума и возвращает длину множества *C* (мощность множества *С*);

3.Входные параметры: a, n;

4.Выходные параметры: c.

Подпрограммы для выполнения операций 4-8(для случая в):

* Объединение

Процедура unification

1.Заголовок: **procedure** unification (a, b: array of boolean; var c: array of boolean);

2. Назначение: в массив *С* записывает результат объединения множеств *А* и *В*;

3.Входные параметры: a, b;

4.Выходные параметры: c.

* Пересечение

Процедура intersection

1.Заголовок: **procedure** intersection (a, b: array of boolean; var c: array of boolean);

2. Назначение: в массив *С* записывает результат пересечения множеств *А* и *В*;

3.Входные параметры: a, b;

4.Выходные параметры: c.

* Разность

Процедура difference

1.Заголовок:**procedure** difference (B, A: array of boolean; var c: array of boolean);

2. Назначение: в массив *С* записывает результат разности множеств *А* и *B*;

3.Входные параметры: a, b;

4.Выходные параметры: c.

* Симметрическая разность

Процедура symmetric\_difference

1.Заголовок: proceduresymmetric\_difference (a, b: array of boolean; var c: array of boolean);

2. Назначение: в массив *С* записывает результат симметрической разность множеств *А* и *В*;

3.Входные параметры:a, b;

4.Выходные параметры: c.

* Дополнение

Процедура complement

1.Заголовок: **procedure** complement (a: array of boolean; var c: array of boolean);

2. Назначение: в массив *С* записывает результат дополнения множества *А* до универсума;

3.Входные параметры: a;

4.Выходные параметры: c.

Текст модулей и программ для реализации смотреть в «Приложение 1» а)-в) соответственно.

**Задание 4**

Написать программы для вычисления значений выражений (см. “Задания”, п.1 и п.2).

**Выполнение**

Программную реализацию вычисления выражений смотреть в «Приложение 2».

**Задание 5**

Используя программы (см. “Задания”,п.4), вычислить значения выражений (см. “Задания”,п.1 и п.2).

**Выполнение**

Программную реализацию вычисления выражений смотреть в «Приложение 3».

Приложение 1

**а) Unit** un\_arr;

**interface**

**const** maxlen = 10;

**type** t\_arr = **array** [1..maxlen] **of** byte;

**procedure** input (**var** x: t\_arr; k: byte);

**procedure** output (x: t\_arr; k: byte);

**function** check (b: t\_arr; m, x: byte): boolean;

**function** check\_a\_b (a, b: t\_arr; n, m: byte): boolean;

**function** inclusion (a, b: t\_arr; n, m: byte): boolean;

**function** equality (a, b: t\_arr; n, m: byte): boolean;

**function** strict\_inclusion (a, b: t\_arr; n, m: byte): boolean;

**function** unification (a, b: t\_arr; n, m: byte; **var** c: t\_arr): byte;

**function** intersection (a, b: t\_arr; n, m: byte; **var** с: t\_arr): byte;

**function** difference (a, b: t\_arr; n, m: byte; **var** с: t\_arr): byte;

**function** symmetric\_difference (a, b: t\_arr; n, m: byte; **var** c: t\_arr): byte;

**function** complement (a: t\_arr; n: byte; **var** c: t\_arr): byte;

**implementation**

**procedure** input;

**var** i:byte;

**begin**

**for** i:=1 **to** k **do**

read(x[i]);

**end**;

**procedure** output;

**var** i:byte;

**begin**

**for** i:=1 **to** k-1 **do**

write(x[i], ‘ ‘);

writeln;

**end**;

**function** check: boolean;

**var** i: byte;

**begin**

i:=1;

**while** (i<=m) **and** (x<>b[i]) **do**

inc(i);

check:=i<=m;

**end**;

**function** check\_a\_b: boolean;

**var** i: byte;

**begin**

i:=1;

**while** (i<=n) **and** check(b, m, a[i]) **do**

inc(i);

check\_a\_b:=i>n;

**end**;

**function** inclusion:boolean;

**begin**

inclusion:=(n<=m) **and** check\_a\_b(a, b, n, m);

**end**;

**function** equality: boolean;

**begin**

equality:=(n=m) **and** check\_a\_b(a, b, n, m);

**end**;

**function** strict\_inclusion:boolean;

**begin**

strict\_inclusion:=(n<m) **and** check\_a\_b(a, b, n, m);

**end**;

**function** unification: byte;

**var** i, k: byte;

**begin**

**for** i:=1 **to** high(a) **do**

c[i]:=a[i];

k:=n+1;

**for** i:=1 **to** m **do**

**if not** check(a, n, b[i]) **then**

**begin**

c[k]:=b[i];

inc(k);

**end**;

unification:=k;

**end**;

**function** intersection: byte;

**var** i, k:byte;

**begin**

k:=1;

**for** i:=1 **to** n **do**

**if** check(b, m, a[i]) **then**

**begin**

c[k]:=a[i];

inc(k);

**end**;

intersection:=k;

**end**;

**function** difference: byte;

**var** i, k: byte;

**begin**

k:=1;

**for** i:=1 **to** n **do**

**if not** check(b, m, a[i]) **then**

**begin**

c[k]:=a[i];

inc(k);

**end**;

difference:=k;

**end**;

**function** symmetric\_difference:byte;

**var** i,k: byte;

**begin**

k:=1;

**for** i:=1 **to** n **do**

**if not** check(b, n, a[i]) **then**

**begin**

c[k]:=a[i];

inc(k);

**end**;

**for** i:=1 **to** m **do**

**if not** check(a, n, b[i]) **then**

**begin**

c[k]:=b[i];

inc(k);

**end**;

symmetric\_difference:=k;

**end**;

**function** complement: byte;

**var** i, k:byte;

**begin**

k:=1;

**for** i:=1 **to** high(a) **do**

**begin**

**if not** check(a, n, i) **then**

**begin**

c[k]:=i;

inc(k);

**end**;

**end**;

complement:=k;

**end**;

**end**.

**program** z3\_a;

**uses** un\_arr;

**var** a, b, c:t\_arr;

n, m, t, p, count, q:byte;

**begin**

writeln('Введите количество элементов множества А');

read(n);

writeln('Введите элементы множества А');

input(a, n);

writeln('Введите количество элементов множества B');

read(m);

writeln('Введите элементы множества B');

input(b, m);

writeln('Введите число операций');

readln(count);

**for** q:=1 **to** count **do**

**begin**

writeln('Введите число от 1 до 8 (1-Включение, 2-Равенство, 3-Строгое включение, 4-Объединение, 5-Пересечение, 6-разность, 7-Симметрическая разность, 8-Дополнение)');

readln(p);

**case** p **of**

1: **if** inclusion(a, b, n, m) **then**

writeln('Множество А включено в множество В')

**else**

writeln('Множество А не включено в множество В');

2: **if** equality(a, b, n, m) **then**

writeln('Множества А и В равны')

**else**

writeln('Множества А и В неравны');

3: **if** strict\_inclusion(a, b, n, m) **then**

writeln('Множество А строго включено в множество В')

**else**

writeln('Множество А не включено строго в множество В');

4: t:= unification(a, b, n, m, c);

5: t:= intersection(a, b, n, m, c);

6: t:= difference(a, b, n, m, c);

7: t:= symmetric\_difference(a, b, n, m, c);

8: t:= complement(a, n, c);

**end**;

**if** p>=4 **then**

output(c, t);

**end**;

**end**.

б) **Unit** or\_arr;

**interface**

**const** maxlen = 10;

**type** t\_arr = **array** [1..maxlen] **of** byte;

**procedure** input(**var** x:t\_arr; k:byte);

**function** sort\_arr (x: t\_arr; k: byte):Boolean;

**procedure** output (x: t\_arr; k: byte);

**function** check\_a\_b (a, b: t\_arr; n, m: byte): boolean;

**function** inclusion (a, b: t\_arr; n, m:byte): boolean;

**function** equality (a, b: t\_arr; n, m: byte): boolean;

**function** strict\_inclusion (a, b: t\_arr; n, m: byte): boolean;

**function** unification (a, b: t\_arr; n, m: byte; **var** c: t\_arr): byte;

**function** intersection (a, b: t\_arr; n, m: byte; **var** c: t\_arr): byte;

**function** difference (a, b: t\_arr; n, m: byte; **var** c: t\_arr): byte;

**function** symmetric\_difference (a, b: t\_arr; n, m: byte; **var** c: t\_arr): byte;

**function** complement (a: t\_arr; n: byte; **var** c: t\_arr): byte;

**implementation**

**procedure** input;

**var** i: byte;

**begin**

**for** i:=1 **to** k **do**

read(x[i]);

**end**;

**function** sort\_arr: boolean;

**var** j: byte;

**begin**

j:=2;

**while** (j<=k) **and** (x[j]>x[j-1]) **do**

inc(j);

sort\_arr:=j>k;

**end**;

**procedure** output;

**var** i: byte;

**begin**

**for** i:=1 **to** k-1 **do**

write(x[i], ‘ ‘);

writeln;

**end**;

**function** check\_a\_b: boolean;

**var** i, j: byte;

**begin**

i:=1;

j:=1;

**while** (i<=n) **and** (j<=m) **do**

**if** a[i]=b[j] **then**

**begin**

inc(i);

inc(j);

**end**

**else** inc(j);

check\_a\_b:=i>n

**end**;

**function** inclusion: boolean;

**begin**

inclusion:=(n<=m) **and** check\_a\_b(a, b, n, m);

**end**;

**function** equality: boolean;

**var** i: byte;

f: boolean;

**begin**

i:=1;

f:=n=m;

**while** (i<=n)**and**(f) **do**

**begin**

f:=a[i]=b[i];

inc(i);

**end**;

equality:=f;

**end**;

**function** strict\_inclusion: boolean;

**begin**

strict\_inclusion:=(n<m) **and** check\_a\_b(a, b, n, m);

**end**;

**function** unification: byte;

**var** i, k, j, p: byte;

**begin**

i:=1;

j:=1;

k:=1;

**while** (i<=n) **and** (j<=m) **do**

**begin**

**if** a[i]=b[j] **then**

**begin**

c[k]:=a[i];

inc(i);

inc(j);

**end**

**else**

**if** a[i] > b[j] **then**

**begin**

c[k]:=b[j];

inc(j);

**end**

**else**

**begin**

c[k]:=a[i];

inc(i);

**end**;

inc(k);

**end**;

**if** i>n **then**

**for** p:=j **to** m **do**

**begin**

c[k]:=b[p];

inc(k);

**end**

**else**

**for** p:=i **to** N **do**

**begin**

c[k]:=a[p];

inc(k);

**end**;

unification:=k;

**end**;

**function** intersection: byte;

**var** i, j, k: byte;

**begin**

i:=1;

j:=1;

k:=1;

**while** (i<=n) **and** (j<=m) **do**

**if** a[i]=b[j] **then**

**begin**

c[k]:=a[i];

inc(k);

inc(i);

inc(j);

**end**

**else**

**if** a[i]>b[j] **then**

inc(j)

**else**

inc(i);

intersection:=k;

**end**;

**function** difference: byte;

**var** i, j, k, p:byte;

**begin**

i:=1;

j:=1;

k:=1;

**while** (i<=n) **and** (j<=m) **do**

**if** a[i] = b[j] **then**

**begin**

inc(j);

inc(i);

**end**

**else**

**if** a[i]>b[j] **then**

inc(j)

**else**

**begin**

c[k]:=a[i];

inc(i);

inc(k);

**end**;

**if** i<=n **then**

**for** p:=i **to** n **do**

**begin**

c[k]:= a[p];

inc(k);

**end**;

difference:=k;

**end**;

**function** symmetric\_difference: byte;

**var** i, j, k, f, l:byte;

**begin**

i:=1;

j:=1;

k:=1;

**while** (i<=n) **and** (j<=m) **do**

**if** a[i] = b[j] **then**

**begin**

inc(i);

inc(j);

**end**

**else**

**if** a[i]>b[j] **then**

**begin**

c[k]:=b[j];

inc(k);

inc(j);

**end**

**else**

**begin**

c[k]:=a[i];

inc(k);

inc(i);

**end**;

**if** i<=n **then**

**for** f:=i **to** n **do**

**begin**

c[k]:=a[f];

inc(k);

**end**;

**if** j<=m **then**

**for** l:=j **to** m **do**

**begin**

c[k]:=b[l];

inc(k);

**end**;

symmetric\_difference:=k;

**end**;

**function** complement: byte;

**var** k, i, j: byte;

**begin**

k := 1;

i := 1;

j := 1;

**while** i <= n **do**

**if** j = a[i] **then**

**begin**

inc(i);

inc(j);

**end**

**else**

**if** j > a[i] **then**

inc(i)

**else**

**begin**

c[k] := j;

inc(k);

inc(j);

**end**;

**while** j<=10 **do**

**begin**

c[k]:= j;

inc(k);

inc(j);

**end**;

complement:= k;

**end**;

**end**.

**program** zd\_3\_b;

**uses** or\_arr;

**var** a, b, c: t\_arr;

b, m, t, p, count, q: byte;

**begin**

writeln('Введите количество элементов множества А');

read(n);

writeln('Введите элементы множества А');

input(a, n);

**while not** sort\_arr(a, n) **do**

**begin**

writeln('Элементы множества не упорядочены, повторите ввод');

input(a, n);

**end**;

writeln('Введите количество элементов множества В');

read(m);

writeln('Введите элементы множества В');

input(b, m);

**while not** sort\_arr(b, a) **do**

**begin**

writeln('Элементы множества не упорядочены, повторите ввод');

input(b, m);

**end**;

k:=0;

writeln('Введите число операций');

readln(count);

**for** q := 1 **to** count **do**

**begin**

writeln('Введите число от 1 до 8 (1-включение, 2-равенство, 3-строгое включение, 4-объединение, 5-пересечение, 6-разность, 7-симметрическая разность, 8-дополнение');

readln(p);

**case** p **of**

1: **if** inclusion(a, b, n, m) **then**

writeln('Множество А включено в множество В')

**else**

writeln('Множество А не включено в множество В');

2: **if** equality(a, b, n, m) **then**

writeln('Множества А и В равны')

**else**

writeln('Множества А и В неравны');

3: **if** strict\_inclusion(a, b, n, m) **then**

writeln('Множество А строго включено в множество В')

**else**

writeln('Множество А не включено строго в множество В');

4: t:=unification(a, b, n, m, c);

5: t:=intersection(a, b, n, m, c);

6: t:=difference(a, b, n, m, c);

7: t:=symmetric\_difference(a, b, n, m, c);

8: t:=complement(a, n, c);

**end**;

**if** p>= 4 **then**

output(c, t);

**end**;

**end**.

в) **unit** bool\_arr;

**interface**

**const** maxlen=10;

**type** t\_arr = **array** [1..maxlen] **of** boolean;

**procedure** input(**var** a: t\_arr; n:byte);

**procedure** output(a: t\_arr);

**function** inclusion(a, b: t\_arr; n, m: byte): boolean;

**function** equality(a, b: t\_arr; n, m: byte): boolean;

**function** strict\_inclusion(a, b: t\_arr; n, m:byte): boolean;

**procedure** unification(a, b: t\_arr; **var** c: t\_arr);

**procedure** intersection(a, b: t\_arr; **var** c: t\_arr);

**procedure** difference(a, b: t\_arr; **var** c: t\_arr);

**procedure** symmetric\_difference(a, b: t\_arr; **var** c: t\_arr);

**procedure** complement(a: t\_arr; **var** c: t\_arr);

**implementation**

**procedure** input;

**var** i, j: byte;

**begin**

**for** i:=1 **to** high(a) **do**

a[i]:=false;

**for** j:=1 **to** N **do**

**begin**

read(i);

a[i]:=true;

**end**;

**end**;

**procedure** output;

**var** i: byte;

**begin**

**for** i:=1 **to** high(a) **do**

**if** a[i] **then** write(i, ‘ ‘);

writeln;

**end**;

**function** inclusion: boolean;

**var** i, j: byte;

f: boolean;

**begin**

i:=1;

f:=n<=m;

**while** (i<=high(a)) **and** f **do**

**begin**

j:=1;

**while** (a[i]<>b[j]) **and** (j<=m) **do**

inc(j);

f:=j<=m;

inc(i);

**end**;

inclusion:=f;

**end**;

**function** equality: boolean;

**var** i: byte;

f: boolean;

**begin**

i:=1;

f:=n=m;

**while** (i<=high(a)) **and** f **do**

**begin**

f:=a[i]=b[i];

inc(i);

**end**;

equality:=f;

**end**;

**function** strict\_inclusion: boolean;

**var** i, j: byte;

f: boolean;

**begin**

i:=1;

f:=n<m;

**while** (i<=high(a)) **and** f **do**

**begin**

j:=1;

**while** (a[i]<>b[j]) **and** (j<=m) **do**

inc(j);

f:=j<=m;

inc(i);

**end**;

strict\_inclusion:=f;

**end**;

**procedure** unifiation;

**var** i: byte;

**begin**

**for** i:=1 **to** high(a) **do**

c[i]:=a[i] **or** b[i];

**end**;

**procedure** intersection;

**var** i: byte;

**begin**

**for** i:=1 **to** high(a) **do**

c[i]:=a[i] **and** b[i];

**end**;

**procedure** difference;

**var** i:byte;

**begin**

**for** i:=1 **to** high(a)

**do** c[i]:=a[i]>b[i];

**end**;

**procedure** symmetric\_difference;

**var** i: byte;

**begin**

**for** i:=1 **to** high(a) **do**

c[i]:=a[i]<>b[i];

**end**;

**procedure** complement;

**var** i:byte;

**begin**

**for** i:=1 **to** high(a) **do**

c[i]:=**not** a[i];

**end**;

**end**.

**program** zd3\_c;

**uses** bool\_arr;

**var** a, b, c: t\_arr;

count, q, n, m, p: byte;

**begin**

writeln('Введите количество элементов множества А');

readln(n);

writeln('Введите элементы множества А');

input(a, n);

writeln('Введите количество элементов множества В');

readln(m);

writeln('Введите элементы множества B');

input(b, m);

writeln('Введите число операций');

readln(count);

**for** q:= 1 **to** count **do**

**begin**

writeln('Введите число от 1 до 8 (1 - Включение; 2 - Равенство; 3 - Строгое включение; 4 – Объединение, 5 - Пересечение; 6 - Разность; 7 - Симметрическая разность; 8 - Дополнение');

read(p);

**case** p **of**

1: **if** inclusion(a, b, n, m) **then**

writeln('Множество А включено в множество В')

**else**

writeln('Множество А не включено в множество В');

2: **if** equality(a, b, n, m) **then**

writeln('Множества А и В равны')

**else**

writeln('Множества А и В не равны');

3: **if** strict\_inclusion(a, b, n, m) **then**

writeln('Множество А строго включено в множество В')

**else**

writeln('Множество А не включено строго в множество В');

4: unification(a, b, c);

5: intersection(a, b, c);

6: difference(a, b, c);

7: symmetric\_difference(a, b, c);

8: complement(a, c);

**end**;

**if** p>=4 **then**

output(c);

**end**;

**end**.

Приложение 2

Программа для вычисления значения выражения п. 1:

**program** z\_1;

**uses** or\_arr;

**var** a, b, c, d, e: t\_arr;

n, m, k, p, l: byte;

**begin**

writeln ('Введите количество элементов множества A');

read (n);

writeln ('Введите элементы множества А');

input (a, n);

writeln ('Введите количество элементов множества B');

read (m);

writeln ('Введите элементы множества B');

input (b, m);

writeln ('Введите количество элементов множества C');

read (k);

writeln ('Введите элементы множества C');

input (c, k);

p:=intersection (a, c, n, k, d);

l:=intersection (b, c, m, k, e);

p:=unification (d, e, p, l, d);

p:=difference (c, d, k, p, d);

write ('D = C – (A ∩ C ⋃ B ∩ C) = { ');

output (d, p);

write ('}');

**end**.

Программа для вычисления значения выражения п. 2:

**program** z\_2;

**uses** or\_arr;

**var** a, b, c, d, e: t\_arr;

n, m, k, p, l: byte;

**begin**

writeln ('Введите количество элементов множества A');

read (n);

writeln ('Введите элементы множества А');

input (a, n);

writeln ('Введите количество элементов множества B');

read (m);

writeln ('Введите элементы множества B');

input (b, m);

writeln ('Введите количество элементов множества C');

read (k);

writeln ('Введите элементы множества C');

input (c, k);

p:=intersection (a, c, n, k, d);

p:=difference (d, b, p, m, d);

p:=difference (c, d, k, p, d);

write ('D = C – (A ∩ C – B) = { ');

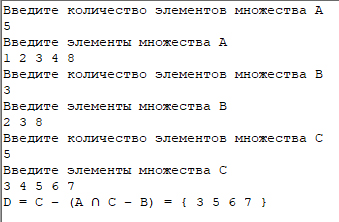
output (d, p);

write ('}');

**end**.

Приложение 3

Задача 1.



Задача 2.

