Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Филиал

«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**

по выполнению лабораторной работы

«Реализация множества и основных алгоритмов его обработки»

Минск

2018

**Лабораторная работа № 7**

**Тема работы: «Реализация множества и основных алгоритмов его обработки»**

**1. Цель работы**

Изучение принципов работы со структурой данных множество, формирование умений использования множеств для решения задач.

**2. Задание**

1. Дана строка. Сохранить в ней только первые вхождения символов, удалив все остальные.

2. В городе имеется n медицинских учреждений, которые производят закупку медицинской техники. Имеется шесть фирм, которые осуществляют продажу соответствующего оборудования: «БЕЛМЕДСНАБ», «Медтехника Восток Плюс», «БелИнтерн», «Висма-Планар», «Инкомед», «Ратон-МедТех». Ответить на следующие вопросы:

1) в каких фирмах закупка производилась каждым из учреждений?

2) в каких фирмах закупка производилась хотя бы одним из учреждений?

3) в каких фирмах ни одно из учреждений не закупало медтехнику?

**3. Оснащение работы**

ПК, Pascal.

**4. Основные теоретические сведения**

Объявление множеств

В математике для обозначения множества используют фигурные скобки (например, {4, 7, 12}), в Паскаль – квадратные (например, [1, 3, 5]). Порядок элементов во множестве не имеет значения. Так, записав [3, 6, 9] или [9, 3, 6], мы будем иметь дело с одним и тем же множеством. Более того, многократное повторение одного и того же элемента не меняет множество. Например, [4, 7, 3] и [3, 7, 4, 4] – это одно и то же множество.

По форме записи объявление переменной типа множество сходно с объявлением одномерного массива:

**var**

**имя: set of тип;**

*Например, объявление переменной* ***ch****, рассматриваемой как множество с базовым типом* ***char****, имеет вид:*

**var**

**ch: set of char;**

В отличие от элементов массива, элементы множества не упорядочены и не имеют индексов.

Можно сначала объявить тип множества, а потом использовать его для объявления переменных. Множественные типы описываются в разделе типов следующим образом :

**Type < имя типа > = Set of < базовый тип>**

**type**

**t\_ch = set of char;**

**var**

**ch1, ch2: t\_ch;**

Довольно часто в качестве базового типа множества используется тип перечисления или некоторый его диапазон:

**type**

**week\_days = (Mon, Tue, Wed, Thu, Fri);**

**var**

**work\_days: set of week\_days;**

**lett: set of 'A'..'Z';**

*Например:*

**а) Type Beta = Set of 100..200;**

**б) Type Glas = Set of char ; {Vowel}**

**в) Type Color = (red, orange, yellow, green, light\_blue, blue, violet);**

**Paint = Set of Color;**

**г) Type TwoDigNum = Set of 10..99;**

**Var A, B: Beta;**

**llet, flet: Glas;**

**last, first: Paint;**

**Sinit: TwoDigNum;**

Объявление переменной-множества не дает ей определенного значения.

Построение множества

Чтобы во множестве появились элементы, необходимо выполнить оператор присваивания, в левой части которого стоит имя переменной-множества, а в правой – конструктор множества или некоторое выражение над множествами.

**Конструктор множества** – это заключенный в квадратные скобки перечень элементов, разделенных запятыми. В качестве элементов могут использоваться диапазоны значений:

**type**

**week\_days = (Mon, Tue, Wed, Thu, Fri);**

**var**

**work\_days: set of week\_days;**

**lett: set of 'A'..'Z';**

**begin**

**work\_days := [Mon, Wed, Thu];**

**lett := ['C', 'E'..'M', 'Z']**

**end.**

*Т.е. данные типа* ***set*** *задаются путем перечисления значений, разделенных запятыми и заключенных в квадратные скобки.*

Общий вид:

**[expr1, expr2, …exprn];**

*здесь expri – выражение базового типа.*

Порядок следования выражений несущественен. Непустой набор может быть также выражением вида:

**[expr1];**

**[expr1..exprk];**

**[expr1, exprk..exprn];**

Данные вида **[expr1..exprk]** соответствуют набору всех элементов базового типа от значения **expr1** до **exprk**.

*Пример:*

**[3 \* 6 – 7..15 + 4]** соответствует набору **[11..19]**, т.е. **[11, 12, 13, 14, 15, 16, 17, 18, 19]**.

Если окажется, что для **[i..j]**, **i > j**, то такое множество интерпретируется как пустое, а в случае **i = j** – как множество, содержащее один элемент – **i**.

*Пример:*

**[3 \* 6 – 7..5 + 6]** эквивалентно **[11]**

**['A'..'Z', 'O'..'9']** – множество, состоящее из всех прописных латинских букв и цифр

**[i + j .. i + 2\*j]** – множество, состоящее из всех целых чисел между i + j и i + 2j

*Пример:*

**type color = (red, yellow, green, blue);**

**var mix: set of color;**

**……………**

**mix := [red, blue];**

*Пример:*

**type n = (1, 3, 5, 7, 9);**

**var k: set of n;**

**…………..**

**k := [3..9];**

*В этом случае в* ***k*** *запишется комбинация [3, 5, 7, 9];*

Следует помнить, что при задании множества порядок его элементов безразличен, но при задании диапазона такой порядок важен.

Множество, в котором нет элементов, называется **пустым** (или нуль-множеством*). В языке программирования Паскаль обозначается квадратными скобками, между которыми нет элементов:*

**work\_days := [ ];**

Множество может быть объявлено типизированной константой, для чего в описании после знака равенства следует указать конструктор множества.

*Например:*

**const lett: set of ['а'..'я'] = ['а', 'е', 'и', 'о', 'у', 'ы', 'э', 'ю', 'я'];**

*В данном случае оператор описывает множество, элементами которого могут быть буквы русского алфавита, с записью в него начального значения, которое представляет собой множество гласных букв.*

Конструируя множества, можно использовать и переменные при условии, что их текущие значения попадают в диапазон базового типа множества.

*Так, если* ***ch1*** *и* ***ch2*** *имеют тип* ***char****, то допустима следующая последовательность операторов:*

**ch1 := 'A';**

**ch2 := 'K';**

**chs := [ch1, ch2, 'M'];**

*В результате получится множество ['A', 'K', 'M'].*

*Пример:*

Пусть базовый тип **int** и тип **a** заданы так:

**type**

**int = 1..3;**

**a = set of int;**

*Переменная a в этом случае может принимать восемь значений:*

*[1], [2], [3], [1, 2], [1, 3], [2, 3], [1, 2, 3], [ ].*

*Например, если переменная* ***b*** *имеет тип* ***a****, то можно присвоить ей одно из перечисленных выше значений: b := [1, 3]; b := [1, 3, 2]; и т.д.*

*Пример.*

Если базовый тип описывает набор двоичных бит, то можно получить их комбинацию. Пусть

**type**

**bin = (bit1, bit2, bit3);**

**bts = set of bin;**

*Переменная типа* ***bts*** *может принимать восемь значений.*

*Таким образом, используя переменные типа* ***set****, можно работать с битовой информацией.*

*В качестве базового типа может использоваться любой простой тип, кроме* ***real****. Если задача требует использования множества, состоящего из большого числа элементов, то его можно представить как массив множеств, состоящих из допустимого числа элементов.*

Элементы множества нельзя вводить и выводить. *Для организации ввода-вывода элементов множества следует использовать вспомогательные переменные. В то же время можно использовать множества как элементы типизированных файлов.*

Действия над множествами

*Над множествами допустимо выполнять следующие действия:*

*- объединение (+);*

*- пересечение (\*);*

*- разность (-);*

*- четыре операции сравнения: =, <>, >=, <=.*

*Операции: + объединение, \* пересечение, дополнение применимы к переменным типа* ***set****, относящимся к одному и тому же конкретному типу.*

**Объединение** двух множеств A и B (A + B) – это новое множество, состоящее из элементов, принадлежащих множеству A или B, либо тому и другому одновременно.



**var**

**chs1, chs2, chs3: set of char;**

**begin**

**chs1 := ['a', 'b', 'd'];**

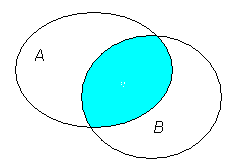
**chs2 := ['m', 'd', 'e'];**

**chs3 := chs1 + chs2 + ['k', 'n'];**

**end.**

*Результат:* ***chs3 = ['a', 'b', 'd', 'm', 'e', 'k', 'n'].***

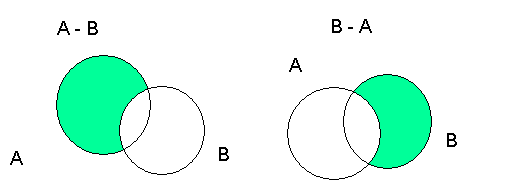
**Пересечение** двух множеств A и B (A \* B) – это множество, состоящее из элементов, одновременно принадлежащих множествам A и B.



**chs3 := chs1 \* chs2;**

*Результат:* ***chs3 = ['d']****.*

**Разность** двух множеств A и B (A – B) – это новое множество, состоящее из элементов множества A, не вошедших в множество B.



**chs1 := ['a', 'e', 't'];**

**chs2 := chs1 – ['e'] //{ ['a', 't'] }**

**chs3 := ['m', 'n', 't'] – chs2 //{ ['m', 'n'] }**

Манипулируя операциями над множествами, можно добавлять элементы к множествам или удалять их.

*Пример:*

**[1, 3] + [1, 4] = [1, 3, 4];**

**[1, 3] \* [1, 4] = [1];**

**[1, 3] - [1, 4] = [3].**

*Операция* ***a := a + x*** *добавляет элемент* ***x*** *к множеству* ***a****. Если* ***x*** *уже имелся в* ***a****, то множество* ***a*** *не меняется.* ***a:= a – x*** *исключает* ***x*** *из* ***a****. Если* ***x*** *отсутствовал в* ***a****, то множество* ***a*** *не меняется.*

Для вставки и удаления элементов при работе с множествами в Pascal введены две процедуры:

**include (имя\_множества, элемент) – позволяет выполнить добавление одного элемента в указанное множество;**

**exclude (имя\_множества, элемент) - позволяет выполнить удаление одного элемента из указанного множества**

*Например:*

**include (chs1, 'g'); { аналогично chs1 + ['g'] }**

**exclude (chs2, 'a'); { аналогично chs2 - ['a'] }**

*Два множества A и B равны (A = B), если каждый элемент множества A является элементом множества B и наоборот.*

*Два множества A и B не равны (A <> B), если они отличаются хотя бы одним элементом.*

Операции **=** и **<>** используются для проверки эквивалентности: два значения переменной типа **set** считаются равными, если они состоят из одних и тех же элементов.

*Пример:*

**[1, 3] = [3, 1] возвращает true,**

**[1..3] = [1, 2, 3] возвращает true,**

**[1] <> [2] возвращает true,**

**[1, 2, 3] = [1, 4, 3] возвращает false,**

**[red, blue] = [red, yellow] возвращает false.**

*Множество A является подмножеством множества B (A <= B, или B >= A), если каждый элемент из A присутствует в B.*

Операции **>=** и **<=** используются для проверки принадлежности одного множества другому:

так, если множество **a** содержится во множестве **b**, то **a** **<= b** дает **true**.

*Пример.*

**[1, 2] <= [1, 2, 3] // дает true**

Пустое множество **[ ]** содержится во всех множествах,

т.е. всегда **[ ] <= [b]** дает **true**.

Операция **in** используется для установления наличия определенного элемента в величине типа **set**. Так, если **x** есть элемент множества **b**, то **(x in b)** дает **true**.

*Пример:*

**red in [red, yellow] возвращает true;**

**red in [blue, green] возвращает false.**

Чтобы проверить, является ли значение **n** цифрой, удобно использовать операцию **in** следующим образом:

**if n in [0..9] then …**

*Все операции* ***сравнения множеств****, а также операция* ***in*** *возвращают логическое значение* ***true*** *или* ***false****.*

|  |  |
| --- | --- |
| Операции над множествами:  - присвоение;  - объединение;  - пересечение;  - дополнение;  - тождественность;  - нетождественность;  - содержится во множестве;  - содержит множество;  - принадлежность элемента множеству. | К **операндам - однотипным множествам** **А** и **В** применимы следующие операции :  А + В - объединение А ∪ В;  А \* В - пересечение А ∩ В;  А - В - разность А \ В.  Между **А** и **В** определены также отношения порядка и равенства:  А = В, А <> В, А < В, А <= В, А > В, А >= В;  *Отношения порядка интерпретируются как теоретико-множественные включения.*  Если **А** - множество и **х** - элемент базового типа, то определено отношение принадлежности **х in A** - **x** принадлежит **A** ( x ∈ A ). |

Каждое из отношений, описанных выше, по-существу, является операцией, результат которой имеет тип Boolean. Таким образом, если **Init** – переменная типа Boolean, возможно присваивание **Init := A < B**. Возможны такие сравнения **( А = В ) = ( С = D )**.

Наличие операций над множествами позволяет применять в программах операторы присваивания, в левой части которых стоит переменная типа множества, а в правой - выражение того же типа. Например:

А := А \* [1 .. 10] + B ; B := (А + B)\*['A' .. 'Z'];

В сложных выражениях над множествами операции имеют следующие приоритеты:

1. **\***
2. **+, -**
3. **=, <>, <=, >=, in**

*Пример:*

Пусть в вашем распоряжении имеется множество из трех монет разного достоинства: 1 р, 5 р, 10 р. Из этих монет можно составить следующие подмножества (их число равно 23 = 8):

1. **{1};**
2. **{5};**
3. **{10};**
4. **{1, 5};**
5. **{1, 10};**
6. **{5, 10};**
7. **{1, 5, 10};**
8. **{ }**

*Эти подмножества и будут принадлежать некоторому множеству, тип которого назовем* ***sum****. Сами элементы (монеты), из которых составляется подмножество, пусть принадлежат некоторому базовому типу, который назовем* ***monet****.*

*Опишем типы данных этого примера:*

**type**

**monet = (m1, m5, m10);**

**sum = set of monet;**

*Пример:*

Рассмотрим в качестве элементов базового типа сигналы от 4-х абонентов **(ab1, ab2, ab3, ab4)**, поступающие на телефонную станцию. Обозначим базовый тип через **abonent**:

**type abonent = (ab1, ab2, ab3, ab4),**

*тогда комбинации сигналов можно описать переменной типа* ***множество****. Назовем этот тип* ***sing****:*

**sing = set of abonent;**

*Тип* ***sing*** *описывает 16 комбинаций.*

Нельзя вводить значения во множественную переменную процедурой ввода и выводить процедурой вывода.

Множественная переменная может получить конкретное значение только в результате выполнения оператора присваивания:

**<множественная переменная> := <множественное выражение>;**

Например:

**A : = [50, 100, 150, 200];**

**B : = ['m', 'n', 'k']; C : = [True, False];**

**D : = A;**

**Реализация множеств посредством двоичных векторов**

Если все рассматриваемые множества будут подмножествами небольшого универсального множества целых чисел **1, ..., N** для некоторого фиксированного **N**, тогда можно применить реализацию АТД (абстрактного типа данных) **SET** посредством **двоичного (булева) вектора***. Единственным условием является наличие некоторого внутреннего порядка среди представляемых элементов: без этого невозможно будет их перенумеровать.*

В этой реализации множество представляется двоичным вектором, в котором i-**й** бит равен **1** (или **true**), если **і** является элементом множества.

[1,3,5] [1,0,1,0,1]

Главное преимущество этой реализации состоит в том, что здесь операторы **MEMBER**, **INSERT** и **DELETE** можно выполнить за фиксированное время (независимо от размера множества) путем прямой адресации к соответствующему биту.

Посмотрим теперь, какими способами мы вынуждены будем имитировать операции над «массивными» множествами:

- **проверка множества на пустоту** *может быть осуществлена довольно просто*:

**pusto := True;**

**for i := 1 to N do**

**if set\_arr[i] then**

**begin**

**pusto := False;**

**Break**

**end;**

- **проверка элемента на принадлежность** множеству *также не вызовет никаких затруднений, поскольку соответствующая компонента массива содержит ответ на этот вопрос*:

**is\_in := set\_arr[element];**

- **добавление элемента *в*** *множество нужно записывать так:*

**set\_arr[element] := True;**

- **удаление элемента из множества** *записывается аналогичным образом:*

**set\_arr[element] := False;**

Но операторы **UNION**, **INTERSECTION** и **DIFFERENCE** выполняются за время, пропорциональное размеру универсального множества.

- **построение пересечения множеств** *реализуется как проверка вхождения каждого элемента в оба множества и последующее добавление удовлетворивших этому условию элементов в результирующее множество:*

Обозначим через **i** номер текущего рассматриваемого элемента в множестве **A**, через **j** – номер текущего рассматриваемого элемента множества **B**. Будем получать множество **P**, представляющее собой пересечение множеств **A** и **B**. Через **k** обозначим мощность множества **P**. Также **k** будет и номером последнего добавленного элемента в **P**.

Алгоритм решения:

1. Положить **i = j = 1** и **k = 0**.

2. Если в **A** и **B** (одновременно) есть ещё непросмотренные элементы, выполнить:

2.1. Если **A[i] = B[j]**, то выполнить:

Добавить **A[i]** в **U**, то есть **k := k + 1** и **U[k] := A[i]**

Перейти к следующим элементам множеств **A, B**, то есть **i := i + 1** и **j := j + 1**

2.2. Если **A[i] < B[j]**, то перейти к следующему элементу множества **A**, то есть **i := i +1**

2.3. В остальных случаях (то есть когда **A[i] > B[j]**) перейти к следующему элементу множества **B**, то есть **j := j + 1**

2.4. Перейти к пункту 2.

- **построение объединения множеств** *аналогичным образом базируется на проверке вхождения элемента хотя бы в одно из объединяемых множеств и дальнейшем добавлении элементов в результирующее множество:*

Обозначим через **i** номер текущего рассматриваемого элемента в множестве **A**, через **j** – номер текущего рассматриваемого элемента множества **B**. Будем получать множество **U**, представляющее собой объединение множеств **A** и **B**. Через **k** обозначим мощность множества **U**. Также **k** будет и номером последнего добавленного элемента в **U**.

Алгоритм решения:

1. Положить i = j =1, k = 0.

2. Если ещё не просмотрены все элементы множеств **A**, **B** выполнить:

2.1. Если в **A** ещё есть элементы, и в **B** есть элементы и **A[i] = B[j**], то

Добавить **A[i]** в **U**, то есть **k := k + 1** и **U[k] := A[i]**

Перейти к следующим элементам в **A** и **B,** то есть **i := i + 1** и **j := j + 1**

2.2. Если в **B** уже все элементы были просмотрены (при условии, что в **A** не все элементы были просмотрены) или же **A[i] < B[j]** выполнить:

Добавить **A[i]** в **U**, то есть **k := k + 1** и **U[k] := A[i]**

Перейти к следующему элементу множества **A**, то есть **i := i + 1**

2.3. Во всех остальных случаях (то есть когда в **A** уже все элементы просмотрены или же если **A[i] > B[j]**) выполнить:

Добавить **B[j]** в **U**, то есть **k := k + 1** и **U[k] := B[j**];

Перейти к следующему элементу множества **B**, то есть **j := j + 1**

2.4. Перейти к пункту 2.

Как видно, на каждом шаге мы добавляем в **U** минимальный **элемент из A[i] и B[j] и** переходим к рассмотрению следующего элемента.

Реализация оператора UNION:

**procedure UNION (A, B: SET; var C: SET ) ;**

**var**

**І: integer ;**

**begin**

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

**C[i]:=A[i] or В[i]**

**end**

Для реализации операторов INTERSECTION и DIFFERENCE надо в примере выше заменить логический оператор **or** на операторы **and** и **and not** соответственно.

- **построение разности двух множеств** *также опирается на проверку вхождения элемента в оба множества, причём добавление элемента в создаваемое множество происходит только в том случае, если элемент присутствует в множестве – уменьшаемом и одновременно отсутствует в множестве – вычитаемом:*

Обозначим через **i** номер текущего рассматриваемого элемента в множестве **A**, через **j** – номер текущего рассматриваемого элемента множества **B**. Будем получать множество **D**, представляющее собой множество **A** без элементов множества **B**. Через **k** обозначим мощность множества **D**. Также **k** будет и номером последнего добавленного элемента в **D**.

Алгоритм решения:

1. Положить i = j = 1 и k = 0.

2. Если в A и B (одновременно) ещё есть непросмотренные элементы, выполнить:

2.1. Если A[i] = B[j], то переходим к следующим элементам множеств A и B, так как равные элементы вычлись и в D ничего добавлять не надо. Выполняем i := i + 1 и j := j + 1.

2.2. Если A[i] < B[j], то, в силу упорядоченности, в множестве B уже точно нет элемента, равного A[i], поэтому ничто не вычитается. Добавляем A[i] в D, то есть k := k + 1 и D[k] := A[i], и переходим к следующему элементу в A, то есть i := i + 1.

2.3. Если A[i] > B[j], то берём следующий элемент из B (так как из A исключить элемент B[i] ввиду того, что в A нет такого элемента), то есть j := j + 1.

2.4. Переходим к пункту 2.

Если универсальное множество так мало, что двоичный вектор занимает не более одного машинного слова, то операторы **UNION**, **INTERSECTION** и **DIFFERENCE** можно выполнить с помощью простых логических операций (конъюнкции (and) и дизъюнкции (or)), выполняемых над двоичными векторами.

- **проверка двух множеств на равенство** *не требует особых пояснений*:

**equal := True;**

**for i :=1 to N do**

**if set1[i] <> set2[i] then**

**begin**

**equal := False;**

**Break**

**end;**

**- проверка двух множеств на включение** *(set1 < set2)( проверка вхождения* ***A*** *в* ***B****) тоже не потребует больших усилий:*

**subset := True;**

**for i := 1 to N do**

**if set1[i] and not set2[i] then**

**begin**

**subset := False;**

**Break**

**end;**

Обозначим через **i** номер текущего рассматриваемого элемента в множестве **A**, через **j** – номер текущего рассматриваемого элемента множества **B**.

Алгоритм решения:

1. Если мощность A больше мощности B, то, очевидно, что A в B не входит. Завершить работу.

2. Положить i = j = 1.

3. Если в A и B (одновременно) есть ещё непросмотренные элементы, выполнить:

3.1. Если A[i] > B[i], то перейти к следующему элементу в B, то есть j := j + 1.

3.2. Если A[i] = B[j], то перейти к следующим элементам в A и B, то есть i := i + 1 и j := j + 1.

3.3. Перейти к пункту 3.

4. Если i - 1 равно N (то есть мы перебрали все элементы из A, а это в нашем алгоритме возможно лишь тогда, когда для каждого элемента из A имеется такой же элемент в B), то A входит в B, иначе не входит.

Не существует никакой процедуры, позволяющей распечатать содержимое множества. Это приходится делать следующим образом:

**{ s : set of type1; k : type1 }**

**for k := min\_type1 to max\_type1 do**

**if k in s then**

**Write(k, ' ');**

**Задача.**

***Описать множество М(1..50). Сделать его пустым. Вводя целые числа с клавиатуры, заполнить множество 10-ю элементами.***

В разделе описания переменных опишем множество целых чисел от 1 до 50, переменную **Х** целого типа будем использовать для считывания числа-кандидата в множество, целую переменную **i** используем для подсчета количества введенных чисел.

В начале программы применим операцию инициализации множества **М:=[ ]**. Так множество в правой части оператора присваивания не имеет элементов, множество **М** будет пустым.

Заполнение множества организуем в цикле **Repeat**, параметр которого **i** будет указывать порядковый номер вводимого элемента. Операцию добавления элемента в множество запишем оператором присваивания **М:=M+[X]**. Перед добавлением элемента выполним его проверку на допустимость **(0 < X < 50)**. При выполнении условия **X in M** выведем сообщение о том, что число **Х** уже принадлежит в множеству.

*Текст программы описания и заполнения множества будет таким:*

**Program InputMno;**

**Var**

**M : set of 1..50;**

**X, i : integer;**

**Begin**

**M := [ ];**

**i :=1;**

**repeat**

**write('Введите ',i,'-й элемент множества');**

**readln(X);**

**if (X in M)**

**then**

**writeln(Х, ' уже содержится в множестве')**

**else**

**if (X < 1) or (X > 50)**

**then**

**writeln('Недопустимое значение ', Х)**

**else**

**begin**

**writeln(Х, ' помещен в множество');**

**M := M+[X];**

**i := i+1;**

**end;**

**until i>10;**

**End.**

Представление множеств в виде двоичных векторов можно применить не только тогда, когда универсальное множество состоит из последовательных целых чисел, но и в более общей ситуации, когда универсальное множество конечно. Для этого достаточно установить взаимно однозначное соответствие между элементами этого множества и целыми числами 1, ..., N.

**Реализация множеств посредством связанных списков**

*Очевиден способ представления множеств посредством связанных списков, когда элементы списка являются элементами множества. В отличие от представления множеств посредством двоичных векторов, в данном представлении занимаемое множеством пространство пропорционально размеру представляемого множества, а не размеру универсального множества. Кроме того, представление посредством связанных списков является более общим, поскольку здесь множества не обязаны быть подмножествами некоторого конечного универсального множества.*

При реализации оператора **INTERSECTION** (Пересечение) в рамках представления множеств посредством связанных списков есть несколько альтернатив:

1. Если универсальное множество линейно упорядочено, то в этом случае множество можно представить в виде сортированного списка, т.е. предполагая, что все элементы множества сравнимы посредством отношения **"<"**, можно ожидать, что эти элементы в списке будут находиться в порядке **е1, е2, ..., еn**, когда **е1 < е2 < e3< … < еn**. Преимущество отсортированного списка заключается в том, что для нахождения конкретного элемента в списке нет необходимости просматривать весь список.

**Элемент будет принадлежать пересечению списков L1 и L2 тогда и только тогда, когда он содержится в обоих списках:**

1. В случае несортированных списков мы должны сравнить каждый элемент списка **L1** с каждым элементом списка **L2**, т.е. сделать порядка **О(n2)** операций при работе со списками длины **n**.

2. Для сортированных списков операторы пересечения и некоторые другие выполняются сравнительно просто: если надо сравнить элемент **е** списка **L1** с элементами списка **L2**, то надо просматривать список **L2** только до тех пор, пока не встретится элемент **е** или больший, чем **е**. В первом случае будет совпадение элементов, второй случай показывает, что элемента **е** нет в списке **L2**. Более интересна ситуация, когда мы знаем элемент **d**, который в списке **L1** непосредственно предшествует элементу **е**. Тогда для поиска элемента, совпадающего с элементом **е**, в списке **L2** можно сначала найти такой элемент **f**, что **d≤f**, и начать просмотр списка **L2** с этого элемента. Используя этот прием, можно найти совпадающие элементы в списках **L1** и **L2** за один проход этих списков, продвигаясь вперед по спискам в прямом порядке, начиная с наименьшего элемента.

Пример реализации оператора INTERSECTION:

Здесь множества представлены связанными списками ячеек, чей тип определяется следующим образом (описание множества):

**type**

**се11type = record**

**element:   elementtype;**

**next:   ↑celltype**

**end**

*предполагается, что* ***elementtype*** *— это тип целых чисел, которые можно упорядочить посредством обычного оператора сравнения* ***<****. Если* ***elementtype*** *представлен другим типом данных, то надо написать функцию, которая будет определять, какой из двух заданных элементов предшествует другому элементу.*

**procedure INTERSECTION ( ahead, bhead: ↑celltype;**

**var pc: ↑celltype );**

**{ Вычисление пересечения сортированных списков А и В с**

**ячейками заголовков ahead и bhead, результат —**

**сортированный список, на чей заголовок указывает рс }**

**var**

**acurrent, bcurrent, ccurrent: ↑celltype;**

**{ текущие ячейки списков A и В и последняя ячейка**

**дополнительного списка С }**

**begin**

**new(pc); { создание заголовка для списка С }**

**acurrent:= aheadf↑ .next;**

**bcurrent:= bnead↑.next;**

**ccurrent:= pc;**

**while   (acurrent <> nil)   and  (bcurrent <> nil)  do begin**

**{  сравнение текущих элементов списков А и В }**

**if acurrent↑.element = bcurrent↑. element then begin {  добавление элемента в пересечение  }**

**nev(ccurrent↑.next);**

**ccurrent:=  ccurrent↑.next;**

**ccurrent↑.element:= acurrent↑.element;**

**acurrent:=  acurrent↑.next;**

**bcurrent:= bcurrent↑.next**

**end**

**else  {  элементы неравны }**

**if acurrent↑.element < bcurrent↑.element then**

**acurrent:=  acurrent↑.next;**

**else**

**bcurrent:= bcurrent↑.next**

**end**

**ccurrent↑.next:= nil**

**end;    {   INTERSECTION   }**

*Связанные списки в данной реализации в качестве заголовков имеют пустые ячейки, которые служат как указатели входа списков.*

*Данную процедуру* ***INTERSECTION*** *можно легко приспособить для реализации операторов* ***UNION*** *и* ***DIFFERENCE****.*

*Для выполнения оператора* ***UNION*** *надо все элементы из списков* ***А*** *и* ***В*** *записать в список* ***С****. Поэтому, когда элементы не равны, наименьший из них заносится в список* ***С****, так же, как и в случае их равенства. Элементы заносятся в список* ***С*** *до тех пор, пока не исчерпаются оба списка, т.е. пока логическое выражение в строке 5 не примет значение* ***false****.*

*В процедуре* ***DIFFERENCE*** *в случае равенства элементов они не заносятся в список* ***С****. Если текущий элемент списка* ***А*** *меньше текущего элемента списка* ***В,*** *то он (текущий элемент списка* ***А****) заносится в список* ***С****. Элементы заносятся в список* ***С*** *до тех пор, пока не исчерпается список А.*

Оператор **ASSIGN(A, В)** копирует список **А** в список **В**. Отметим, что этот оператор нельзя реализовать простым переопределением заголовка списка **В** на заголовок списка **А**, поскольку при последующих изменениях в списке **В** надо будет делать аналогичные изменения в списке **А**, что, естественно, может привести к нежелательным коллизиям.

Оператор **MIN** реализуется легко – просто возвращается первый элемент списка.

Операторы **DELETE** и **FIND** можно реализовать, применив общие методы поиска заданного элемента в списках, в случае оператора **DELETE** найденный элемент (точнее, ячейка, в которой он находится) удаляется.

Реализовать оператор **insert** (вставки) нового элемента в список также несложно, но он должен стоять не в произвольной позиции в списке, а в "правильной" позиции, учитывающей взаимный порядок элементов.

Процедура INSERT, *которая в качестве параметров имеет вставляемый элемент и указатель на ячейку заголовка списка, куда вставляется элемент:*

**procedure  INSERT   (   х:   elementtype;   p:   ↑celltype   );**

**var**

**current,   newcell:   ↑celltype;**

**begin**

**current:=  p;**

**while current↑.next <> nil do begin**

**if current↑.next↑.element = x then**

**return;   {  элемент x уже есть  в списке  }**

**if current↑.next↑.element > x then**

**goto add; { далее останов процедуры }**

**current:= current↑.next**

**end;**

**add: {здесь current — ячейка, после которой надо вставить х}**

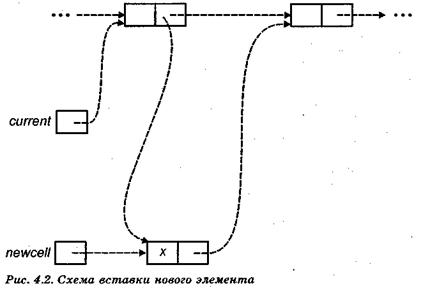
**new(newcell) ;**

**newcell↑.element := х;**

**newcell↑.next:= current↑.next;**

**current↑.next:= newcell**

**end; { INSERT }**



*На рисунке показаны ключевые ячейки и указатели до и после вставки элемента (старые указатели обозначены сплошными линиями, а новые — пунктирными).*

**Словари**

*Применение множеств при разработке алгоритмов не всегда требует таких мощных операторов, как операторы* ***объединения*** *и* ***пересечения****. Часто достаточно только хранить в множестве "текущие" объекты с периодической вставкой или удалением некоторых из них. Время от времени также возникает необходимость узнать, присутствует ли конкретный элемент в данном множестве.* ***Абстрактный тип множеств*** *с операторами* ***INSERT****,* ***DELETE*** *и* ***MEMBER*** *называется* ***DICTIONARY*** *(Словарь). Мы также включим оператор* ***MAKENULL*** *в набор операторов словаря – он потребуется при реализации АТД для инициализации структур данных. Далее в этом разделе мы приведем пример использования словарей, а в следующем разделе рассмотрим реализации, подходящие для представления словарей.*

***Пример:***

Общество защиты тунцов (ОЗТ) имеет базу данных с записями результатов самого последнего голосования законодателей по законопроектам об охране тунцов. База данных состоит из двух списков (множеств) имен законодателей, которые названы **goodguys** (хорошие парни) и **badguys** (плохие парни). ОЗТ прощает законодателям их прошлые "ошибки", но имеет тенденцию забывать своих "друзей", которые ранее голосовали "правильно". Например, после голосования по законопроекту об ограничении вылова тунца в озере Эри все законодатели, проголосовавшие за этот законопроект, заносятся в список **goodguys** и удаляются из списка **badguys**, тогда как над оппонентами этого законопроекта совершается обратная процедура. Законодатели, не принимавшие участие в голосовании, остаются в тех списках, в которых они были ранее.

*Для управления описываемой базы данных при вводе имен законодателей будем применять односимвольные команды, за символом команды будет следовать 10 символов с именем законодателя. Каждая команда располагается в отдельной строке. Используем следующие односимвольные команды.*

*1. F (законодатель голосовал "правильно").*

*2. U (законодатель голосовал "неправильно").*

*3. ? (надо определить статус законодателя).*

*Мы также будем использовать символ 'Е' для обозначения окончания процесса ввода списка законодателей.*

**program  tuna   (   input,   output  ) ;**

**{  База данных законодателей   (legislator)   }**

**type**

**nametype = array[1..10]   of char;**

**var**

**command:   char;**

**legislator:   nametype;**

**goodguys,   badguys:   DICTIONARY;**

**procedure  favor   (   friend:   nametype   );**

**{  заносит имя friend  (друг)   в список goodguys**

**и вычеркивает из списка badguys }**

**begin**

**INSERT(friend,   goodguys);**

**DELETE(frіend,  badguys)**

**end;   {   favor }**

**procedure  unfavor   (   foe:   nametype  );**

**{   заносит имя  foe   (враг)   в список badguys**

**и вычеркивает из списка goodguys  }**

**begin**

**INSERT(foe,   badguys);**

**DELETE(foe, goodguys)**

**end; { unfavor }**

**procedure report ( subject: nametype );**

**{печать имени subject с соответствующей характеристикой}**

**begin**

**if MEMBER(subject, goodguys) then**

**writeln(subject, ' — это друг')**

**else if MEMBER(subject, badguys) then**

**writeln(subject, ' — это враг')**

**else**

**writeln('Heт данных о ', subject,)**

**end; { report }**

**begin { основная программа }**

**MAKENULL(goodguys);**

**MAKENULL(badguys);**

**read(command) ;**

**while command <> 'E' do begin**

**readln(legislator);**

**if command = 'F' then**

**favor(legislator)**

**else if command = 'U' then**

**unfavor(legislator)**

**else if command = '?' then**

**report(legislator)**

**else**

**report('Неизвестная команда')**

**read(command)**

**end**

**end; { tuna }**

**const**

**maxsize = { некое число, максимальный размер массива }**

**type**

**DICTIONARY = record**

**last: integer;**

**data: array[1..maxsize] of nametype**

**end;**

**procedure MAKENULL ( var A: DICTIONARY );**

**begin**

**A.last:= 0**

**end; { MAKENULL }**

**function MEMBER ( x: nametype; var A: DICTIONARY ): boolean;**

**var**

**i: integer;**

**begin**

**for i:= 1 to A.last do**

**if A.data[і] = x then return(true);**

**return(fale) { элемент x не найден }**

**end; { MEMBER }**

**procedure INSERT ( x: nametype; var A: DICTIONARY );**

**begin**

**if not MEMBER(x, A) then**

**if A.last < maxsize then begin**

**A.last:= A.last + 1;**

**A. data [A. last] := x**

**end**

**else error('База данных заполнена')**

**end; { INSERT }**

**procedure DELETE ( x: nametype; var A: DICTIONARY );**

**var**

**i:= integer;**

**begin**

**if A.last > 0 then begin**

**i:= 1;**

**while (A.data[i] <> x) and (i < A.last) do**

**i:= і + 1;**

**if A.data[i] - x then begin**

**A.data[i] = A.data[A.last];**

**{ перемещение последнего элемента на место**

**элемента х; если і = A.last, то удаление х**

**происходит на следующем шаге }**

**A.last:= A.last - 1**

**end**

**end**

**end; { DELETE }**

В узком смысле **комбинаторика** – это подсчёт различных комбинаций, которые можно составить из некоторого множества **дискретных**объектов.

Все разнообразие комбинаторных формул может быть выведено из двух основных утверждений, касающихся конечных множеств – **правило суммы** и **правило произведения**.

Самыми распространёнными видами комбинаций являются **перестановки** объектов, их **выборка** из множества (**сочетание**) и **распределение** (**размещение**).

**1. Правило суммы**: пусть имеется **n** попарно непересекающихся множеств **A1, A2, …, An**, содержащих **m1, m2, …, mn** элементов соответственно. Число способов, которыми можно выбрать один элемент из всех этих множеств, равно **m1 + m2 + … + mn**.

*Пример:*

Если на первой полке стоит **X** книг, а на второй **Y**, то выбрать книгу с первой или второй полки, можно **X+Y** способами.

**2. Правило произведения:** пусть имеется **n** множеств **A1, A2, …, An** содержащих **m1, m2, …, mn** элементов соответственно. Число способов, которыми можно выбрать по одному элементу из каждого множества, т. е. построить кортеж **(а1, а2, ..., аn)**, где **аi ∈ А i1** (i = 1, 2, …, n), равно **m1\* m2\*…\*mn**.

*Пример:*

Если на первой полке стоит 5 книг, а на второй 10, то выбрать одну книгу с первой полки и одну со второй можно 5\*10=50 способами.

**3.** **Выборки.** Если из множества предметов выбирается некоторое подмножество, то его называют **выборкой**. Выборки бывают **упорядоченные** и **неупорядоченные**.

В упорядоченной выборке существенен порядок, в котором следуют ее элементы, другими словами, изменив порядок элементов, мы получим другую выборку.

*Пример:*

Из цифр 1, 2, 3, 4, 5 можно составить следующие трехзначные числа 123, 431, 524, ...и т.д. Это упорядоченные трехэлементные выборки, так как 123 и 132 – разные числа.

**4. Размещения.** Размещениями из **n** элементов по **m** элементов (**m < n**) называются комбинации, составленные из данных **n** элементов по **m** элементов, которые отличаются либо самими элементами, либо порядком элементов.

**Число размещений** без повторений из **n** по **m** (**n** различных элементов) вычисляется по формуле:

|  |  |
| --- | --- |
|  | **(3.1)** |

**Размещениями с повторениями** из **n** элементов по **m** называются упорядоченные **m**-элементные выборки, в которых элементы могут повторяться *(важен порядок следования элементов)*.

**Число размещений с повторениями** вычисляется по формуле:

|  |  |
| --- | --- |
|  | **(3.2)** |

*Пример:*

Возьмем буквы **Б, А, Р**. Какие размещения из этих букв, взятых по две, можно получить? Сколько таких наборов получиться, если: 1) буквы в наборе не повторяются; 2) буквы могут повторяться?

Решение:

1) получатся следующие наборы: **БА, БР, АР, АБ, РБ, РА.**

По формуле (3.1) получаем: http://umk.portal.kemsu.ru/uch-mathematics/papers/posobie/image016.png наборов.

2) получатся наборы: **ББ, БА, БР, АА, АБ, АР, РР, РБ, РА.**

По формуле (3.2) получаем: наборов.

*Пример:*

Вдоль дороги стоят 6 светофоров. Сколько может быть различных комбинаций их сигналов, если каждый светофор имеет 3 состояния: "красный", "желтый", "зеленый"?

Решение:

Выпишем несколько комбинаций: КККЖЗЗ, ЗЗЗЗЗЗ, КЖЗКЖЗ... Мы видим, что состав выборки меняется и порядок элементов существенен (ведь если, например, в выборке КЖЗКЖЗ поменять местами К и Ж, ситуация на дороге будет другой). Поэтому применяем формулу (3.2) и вычисляем число размещений с повторениями из 3 по 6, получаем http://umk.portal.kemsu.ru/uch-mathematics/papers/posobie/image28.gif комбинаций.

**5. Перестановки.** Перестановками из **n** элементов называются размещения из этих **n** элементов по **n** (Перестановки – частный случай размещений).

**Число перестановок без повторений** (**n** различных элементов) вычисляется по формуле:

|  |  |
| --- | --- |
|  | **(3.3)** |

**Число перестановок c повторениями** (**k** различных элементов, где элементы могут повторяться **m1, m2, …, mk** раз и **m1+ m2+… + mk= n**, где **n** - общее количество элементов) вычисляется по формуле:

|  |  |
| --- | --- |
|  | **(3.4)** |

*Пример:*

Возьмем буквы **Б, А, Р**. Какие перестановки из этих букв можно получить? Сколько таких наборов получится, если: 1) буквы в наборе не повторяются; 2) буква **А** повторяется два раза?

Решение:

1. Получатся наборы: **БАР**, **БРА**, **АРБ**, **АБР**, **РАБ**, **РБА**.

По формуле (3.3) получаем: наборов.

2. Получатся наборы: **БАРА**, **БРАА**, **БААР**, **ААРБ**, **ААБР**, **АБАР**, **АРАБ**, **АРБА**, **АБРА**, **РАБА**, **РААБ**, **РБАА**.

По формуле (3.4) получаем: наборов.

**6. Сочетания.** **Сочетаниями** из **n** элементов по **m** элементов называются комбинации, составленные из данных **n** элементов по **m** элементов, которые различаются хотя бы одним элементом (отличие сочетаний от размещений в том, что в сочетаниях не учитывается порядок элементов).

*Сочетание – это выбор из N предметов нескольких (M), причем порядок не важен.*

*Из курса комбинаторики известно, что число сочетаний из N по M равно N!/(M! \* (N - M)!)*

**Число сочетаний без повторений** (**n** различных элементов, взятых по **m**) вычисляется по формуле:

|  |  |
| --- | --- |
|  | **(3.5)** |

Для примера, всего 10 сочетаний из 5 по 3: (1, 2, 3), (1, 2, 4), (1, 2, 5), (1, 3, 4), (1, 3, 5), (1, 4, 5), (2, 3, 4), (2, 3, 5), (2, 4, 5), (3, 4, 5)

**Число сочетаний c повторениями** (**n** элементов, взятых по **m**, где элементы в наборе могут повторяться) вычисляется по формуле:

|  |  |
| --- | --- |
|  | **(3.6)** |

*Пример:*

Возьмем буквы **Б, А, Р**. Какие сочетания из этих букв, взятых по две, можно получить? Сколько таких наборов получится, если: 1) буквы в наборе не повторяются; 2) можно брать по два одинаковые буквы.

Решение:

1. Получатся наборы: **БА** (БА и АБ – один и тот же набор), **АР** и **РБ**.

По формуле (3.5) получаем: наборов.

2. Получатся наборы: **ББ, БА, БР, АА, АР, РР**.

По формуле (3.6) получаем: наборов.

***Схема определения вида комбинации:***



**Все подмножества данного множества**

Допустим, что у нас есть множество **S**, состоящее из элементов **a1, a2, …, aN**, т.е.

**S = {a1, a2, …, aN}**

Для простоты можно считать, что **a1, .. aN** – это различные целые числа от 1 до N.

Подмножеством данного множества S называется множество S', которое содержит некоторые элементы из S (не обязательно все). У множества из N элементов будет ровно 2N различных подмножеств. Получить их все можно как минимум тремя различными способами.

#### 1-й способ

Для примера возьмем N = 3. Запишем все числа от 0 до 2N- 1 = 7 в двоичной системе счисления:

**0 - 000**

**1 - 001**

**2 - 010**

**3 - 011**

**4 - 100**

**5 - 101**

**6 - 110**

**7 - 111**

*Если на* ***i****-той позиции в двоичной записи стоит* ***1****, то* ***i****-тый элемент входит в подмножество, иначе - не входит. Поэтому данный алгоритм можно реализовать так:*

**Var N : Longint;**

**Count : Longint;**

**I : Longint;**

**Procedure PrintSet(X : Longint);**

**Var P : Integer;**

**Begin**

**Write('{ ');**

**P:=N;**

**While X <> 0 Do**

**Begin**

**If X Mod 2 = 1 Then Write(P, ' ');**

**Dec(P);**

**X:=X Div 2;**

**End;**

**Writeln('}');**

**End;**

**Begin**

**N:=5;**

**Count:=1 Shl N; {2^N}**

**For I:=0 To Count - 1 Do**

**PrintSet(I);**

**End.**

#### 2-й способ

Допустим, что у нас уже есть множество S, причем про *N* элементов мы уже знаем, входят они в множество или нет. Из него можно получить еще 2 множества – то, которое содержит (*N + 1*)-й элемент, и то, которое (*N + 1*)-й элемент не содержит.

Получается рекурсивный алгоритм:

**Function IntToStr(X : Longint) : String;**

**Var S : String;**

**Begin**

**Str(S, X);**

**IntToStr:=S;**

**End;**

**Procedure PrintSet(N : Integer; S : String);**

**Begin**

**If (N = 0) Then**

**Writeln('{ ', S, '}')**

**Else**

**Begin**

**PrintSet(N - 1, S + ' ' + IntToStr(N));**

**PrintSet(N - 1, S);**

**End;**

**End;**

**Begin**

**PrintSet(5);**

**End.**

#### 3-й способ

На самом деле, это вариация первого способа, но такой метод очень полезный.

Пусть у нас уже есть какое-то подмножество. Подумаем, как из него получить следующее (все подмножества будем записывать в определенном порядке).

Для определенности заведем массив, в котором будем хранить, какие элементы входят в данное подмножество. Начнем генерировать подмножества с пустого (т.е. в множестве нет ни одного элемента).

Допустим, что N = 5 и текущее подмножество {1, 2, 5} или в битовом представлении (1, 1, 0, 0, 1). Найдем первый 0 слева (если такого элемента не найдем, значит у нас «полное» множество, которое будет последним). Заменим этот 0 на 1, а все единицы слева - на нули.

**Const MaxN = 100; {Максимальное число элементов в множестве}**

**Var Bits : Array[1..MaxN] Of Byte; {Массив битов}**

**N : Longint;**

**I : Longint;**

**Begin**

**ReadLn(N); {Считываем, какого размера множество}**

**For I:=1 To N + 1 Do Bits[I]:=0; {Сначала у нас пустое подмножество}**

**While (Bits[N + 1] = 0) Do {Пока не все множество состоит из 1}**

**Begin**

**Write('{ '); {Выводим текущее подмножество}**

**For I:=1 To N Do**

**If Bits[I] = 1 Then Write(I, ' ');**

**Writeln('}');**

**I:=1; {Ищем следующее подмножество}**

**While (Bits[I] = 1) Do**

**Begin**

**Bits[I]:=0;**

**Inc(I);**

**End;**

**Bits[I]:=1;**

**End;**

**End.**

**5. Порядок выполнения работы**

1. Решить задачи с использованием множеств.

2. Ответы на поставленные во втором задании вопросы получить с помощью операций объединения, пересечения и разности множеств.

3. Результат вывести на экран.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы и задания**

1. Что такое множество?

2. Почему множество является структурированным типом данных?

3. Какие операции можно выполнять над множествами?

4. Как добавить элемент в множество?

5. Как исключить элемент из множества?

6. Как вывести элементы множества? Как подсчитать количество элементов в множестве?

7. Как может быть использована операция вхождения?

**8. Рекомендуемая литература**

1.Ахо, А.В.Структуры данных и алгоритмы / А.В. Ахо, Дж.Хопкрофт, Дж. Д.Ульман. – пер. с англ. – М.: Издательский дом "Вильямс", 2007.-400 с.

2. Вирт,Н. Алгоритмы и структуры данных / Н. Вирт. –СПб.:Невский диалект, 2008. – 352с.

3. Вирт, Н. Алгоритмы и структуры данных с примерами на Паскале / Н. Вирт[перевод с английского Д. Б. Подшивалова]; – 2-е изд., испр., – СПб.: Невский Диалект, 2005. – 352с.

4. Гагарина, Л.Г. Алгоритмы и структуры данных / Л.Г. Гагарина, В.Д. Колдаев. – учеб.пособие – М: Финансы и статистика, 2009. – 304с.

5. Котов, В.М. Алгоритмы и структуры данных: учеб.пособие / В.М. Котов, Е.П. Соболевская, А.А. Толстиков – Минск: БГУ, 2011. – 267с.

6. Макконнелл, Дж. Основы современных алгоритмов / Дж. Макконнелл – 2-е дополненное издание – М.:Техносфера, 2006. – 368с.

7. Окулов, С.М. Программирование в алгоритмах / С.М. Окулов. – 3-е изд. – М.: БИНОМ. Лаборатория знаний, 2007. – 383с.