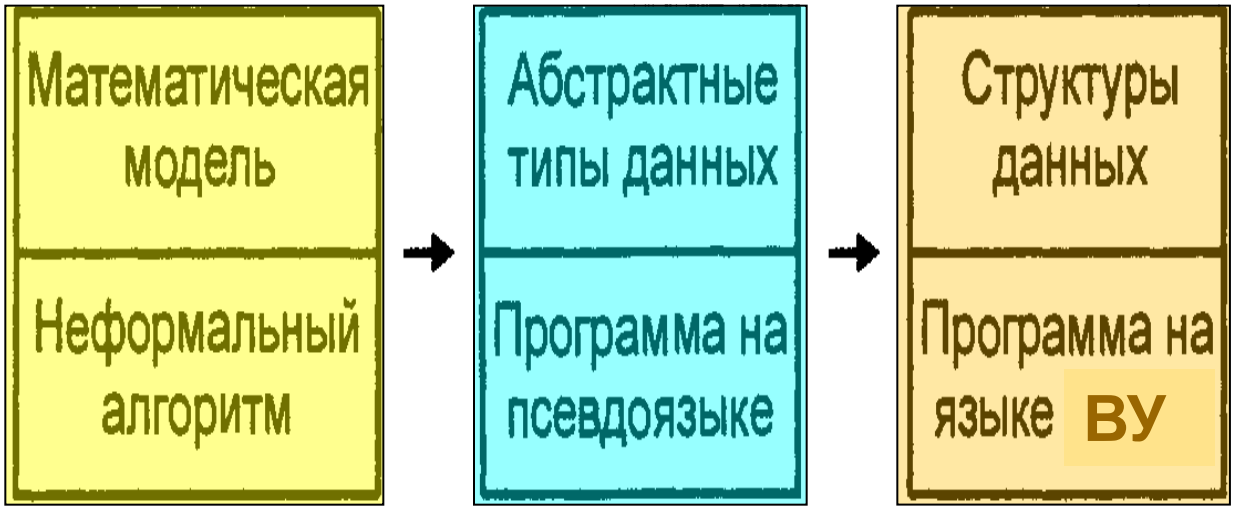
Основные абстрактные типы данных

Схема процесса создания программ для решения прикладных задач



Основные абстрактные типы данных

Определение1:

Абстрактный тип данных (АТД) - некоторая математическая или информационная модель с совокупностью операторов, определенных в рамках этой модели.

Например: множество целых чисел с операторами объединения, пересечения и разности множеств.

Определение 2:

Абстрактный тип данных (АТД) — это тип данных (набор значений и совокупность операций для этих значений), доступ к которому осуществляется только через интерфейс.

Основные абстрактные типы данных: списки

В математике *список* представляет собой последовательность элементов определенного типа.

Список представляется в виде последовательности элементов: $a_1, a_2, a_3, \ldots, a_n$ все a_i имеют один тип.

Количество элементов *n* будем называть *длиной списка*. Если $n \geq 1$, то a_1 называется *первым элементом*, а a_n — *последним элементом* списка.

В случае n=0 имеем *пустой список*, который не содержит элементов.

Основные абстрактные типы данных: списки

insert(x, *p*, *L*)

Этот оператор вставляет объект *x* в позицию *p* в списке L, перемещая элементы от позиции *p* и далее в следующую, более высокую позицию. Если список *L* состоит из элементов

$$a_1, a_2, a_3, \ldots, a_n$$

то после выполнения этого оператора он будет иметь вид $a_1, a_2, a_3, a_{p-1}, x, a_p \ldots, a_n$

Если *p* принимает значение n+1, то будем иметь $a_1, a_2, a_3, \ldots, a_n, x$.

Если в списке *L* нет позиции *p*, то результат выполнения этого оператора не определен.

Основные абстрактные типы данных: списки

locate(x, *L*)

Функция возвращает позицию объекта *x* в списке L.

Если в списке объект *x* встречается несколько раз, то возвращается позиция первого от начала списка объекта *x*.

Если объекта *x* нет в списке L, то возвращается -1

Основные абстрактные типы данных: списки

retrieve(p, *L*)

Эта функция возвращает элемент, который стоит в позиции *p* в списке *L*.

Результат не определен, если *p* = n+1

или в списке L нет позиции *p*.

Отметим, что элементы должны быть того типа, который в принципе может возвращать функция. Однако на практике мы всегда можем изменить эту функцию так, что она будет возвращать указатель на объект типа elementtype.

delete(p, L)

Этот оператор удаляет элемент в позиции p списка L.

если список L состоит из элементов $a_1, a_2, a_3, \ldots, a_n$ то после выполнения этого оператора он будет иметь вид

$$a_1, a_2, a_3, \ldots a_{p-1}, a_{p+1} \ldots, a_n$$

Результат не определен, если в списке L нет позиции p или $p = n-1$

Основные абстрактные типы данных: списки

next(p, *L*) и previous(p, *L*).

Функции возвращают соответственно следующую и предыдущую позиции от позиции *p* в списке L. Если *p* — последняя позиция в списке *L*, то next(p, L) не определена

Функция previous не определена, если *p* = 1.

Обе функции не определены, если в списке L нет позиции *p*.

makenull(L)

Эта функция делает список L пустым и возвращает позицию n

first(L)

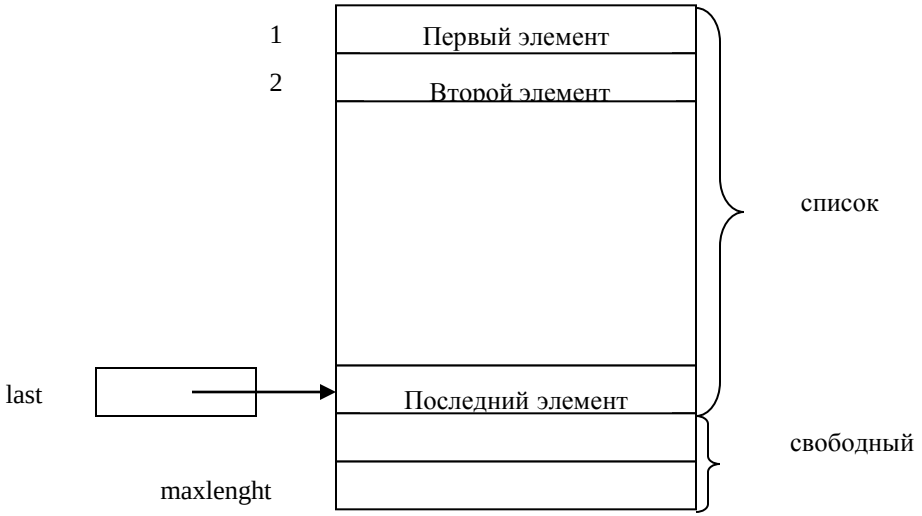
Эта функция возвращает первую позицию в списке *L*.

Если список пустой, то возвращается значение 0

printlist(L)

Печатает элементы списка *L* в порядке из расположения.

Реализация списков (линейное представление)



Представление списка с помощью массива

Реализация списков

```
Const maxl  = 1000;
Type
  List = record
           elements: array[1..maxl] of eltype;
           last: integer
         end;
  position = integer;
```

```
Function EndL (var L:List) : position;
    begin
        EndL := L.last+1
    end;
```

```
procedure INSERTL (x: eltype; p: position; var L:
      LIST );
  { вставляет элемент x в позицию p в списке L }
  var
    q  : position;
    begin
if L.last >= maxlength
then Writeln (’Список полон')
else if (p > L.last + 1) or (p < 1)
       then
        Writeln ('Такой позиции не существует')
       else begin
```

```
    for q:= L.last downto p do
          {перемещение элементов     на
              одну позицию к концу списка }
        L.elements[q+l]:= L.elements[q];
    L.last:= L.last + 1;
    L.elements[p]:= x
  end
end; { INSERT }
```

```
procedure DELETEL ( p: position;     var L: LIST ) ;
{удаляет элемент в позиции p списка L }
var
    q: position;
  begin
if (p > L.last) or (p < 1)
then
    Writeln ('Такой позиции не существует')
 else
```

```
begin
  L.last:= L.last – 1;
  for q:= p to L.last do
  { перемещение элементов из позиций
   p+1, p+2, ...
      на одну позицию к началу списка }
   L.elements[q] := L.elements [q+1]
end
 end; { DELETE }
```

```
function LOCATE ( x:eltype; L: LIST ):  position;
 { возвращает позицию элемента x в списке L }
 var
   q: position;
   begin
    for q:= 1 to L.last do
     if L.elements[q] = x
     then begin Locate := q; Exit (LOCATE)
               end
      else  Locate := L.last+1 {x не найден }
    end; { LOCATE }
```

АТД вектор

АТД «вектор» расширяет структуру данных список.

- **содержит методы доступа к обычным элементам последовательности, также методы обновления, обеспечивающие удаление и добавление элементов с определенным индексом.**

Для отличия от процесса доступа к элементам конкретной структуры данных массива при обозначении индекса элемента вектора используется термин «разряд».

АТД вектор

Пусть S -линейная последовательность из n элементов.

Каждый элемент e последовательности имеет уникальный индекс, выраженный целым числом в интервале [0, n-1], равный числу элементов, предшествующих e.

Разряд элемента e равен количеству элементов, находящихся перед e,

разряд первого элемента равен 0,

а последнего - n-1

АТД вектор

Линейная последовательность элементов, в которой доступ к элементам осуществляется по их разряду, называется вектором.

- **АТД вектор поддерживает следующие основные методы:**

1. elemAtRank (r): возвращает элемент S с разрядом r;

если r < 0 или r > n – 1 - сообщение об ошибке.

Input: целое число; Output: объект.

2. **replaceAtRank (r,e): замещает объектом e элемент с разрядом r и возвращает замещаемый объект.**

Контроль (r < 0 или r > n – 1),

Input: целое число r и объект e; Output: объект.

3. **insertAtRank (r,e): добавляет в S новый элемент e**

Input: целое число r и объект e Output: нет.

4. **removeAtRank (r): удаляет из S элемент с разрядом r;**

Input: целое число; Output: объект.

5. **АТД вектор поддерживает стандартные методы size() и isEmpty().**

АТД вектор

Алгоритм insertAtRank(r,e)

for $i = n - 1, n - 2, \ldots, r$ **do**

$A[i + 1] \leftarrow A[i]$ (создает место для нового элемента)

$A[r] \leftarrow e$

$n \leftarrow n + 1$

Алгоритм removeAtRank(r)

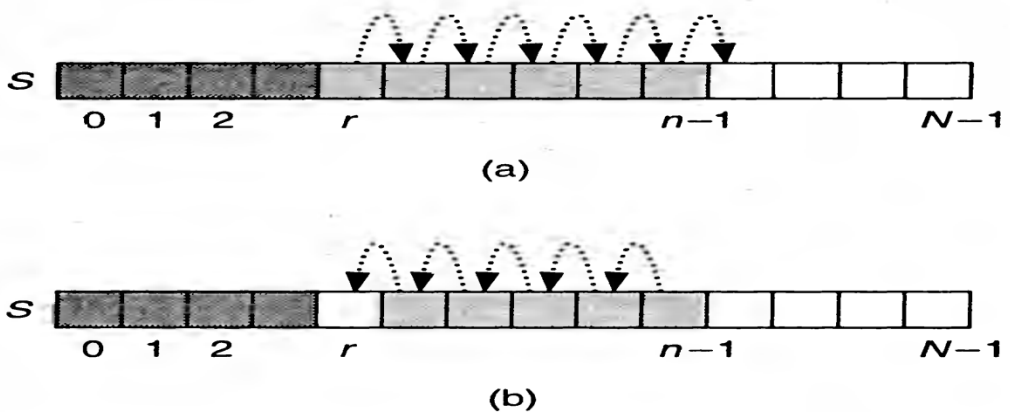
$e \leftarrow A[r]$ (e — временная переменная)

for $i = r, r + 1, n - 2$ **do**

$A[i] \leftarrow A[i + 1]$ (замещает удаленный элемент)

$n \leftarrow n - 1$

return e



АТД вектор

Метод	Время
size	$O(1)$
isEmpty()	$O(1)$
elemAtRank(*r*)	$O(1)$
replaceAtrank(*r*,*e*)	$O(1)$
insertAtRank(*r*,*e*)	$O(n)$
removeAtRank(*r*)	$O(n)$

АТД вектор

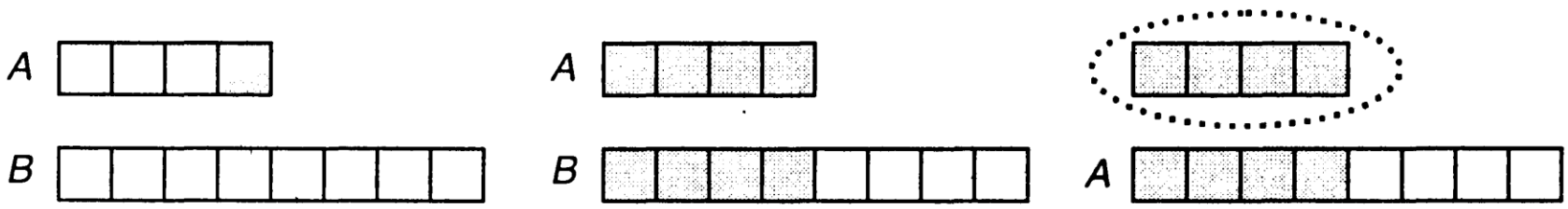
при возникновении переполнения (n = N),

и вызове метода insertAtRank выполняются следующие операции:

1) создается новый массив B длиной 2N;

2) копируется A[i] в B[i], где i= 0, ..., N- 1;

3) присваивается A = B, то есть используется B как массив, содержащий S.



АТД вектор

Классы java.util.ArrayList и java.util.Vector являются встроенными классами Java, которые реализуют АТД «вектор», включая расширяемые массивы

Метод АТД вектора	Метод java util Vector
size(),isEmpty()	size(),isEmpty()
elemAtRank(*r*)	get (*r*)
replaceAtrank(*r*,*e*)	set (*r*,*e*)
insertAtRank(*r*,*e*)	add (*r*,*e*)
removeAtRank(*r*)	remove (*r*)

Коллекции

Класс ArrayList – динамический массив объектов (ссылок).

Расширяет класс AbstractList и реализует интерфейс List.

Класс имеет конструкторы:

ArrayList()

ArrayList(Collection c)

ArrayList(int capacity)

Коллекции

Плюсы:

Быстрый доступ по индексу

Быстрая вставка и удаление краевых элементов

Минусы:

Медленная вставка и удаление элементов

Коллекции

Методы класса ArrayList

void add(int index, Object obj) – вставляет obj в позицию, указанную в index;

void addAll(int index, Collection c) – вставляет в вызывающий список все элементы коллекции c, начиная с позиции index;

Object get(int index) – возвращает элемент в виде объекта из позиции index;

int indexOf(Object ob) – возвращает индекс указанного объекта;

Object remove(int index) – удаляет объект из позиции index.

Коллекции

```
/* пример: работа со списком : DemoList1.java */
import java.util.*;
public class DemoList1 {
        public static void main(String[] args) {
                List c = new ArrayList();
         //Collection c = new ArrayList();
                int i = 2, j = 5;
                c.add(new Integer(i));
                c.add(new Boolean("True"));
```

Коллекции

```
        c.add("<STRING>");
            c.add(2, Integer.toString(j) + "X");
        //добавить во 2 позицию  5
            System.out.println(c);
            if (c.contains("5X"))
                c.remove(c.indexOf("5X"));
            System.out.println(c);
    }
}
```

Коллекции

на консоль будет выведено:

[2, true, 5X, <STRING>]

[2, true, <STRING>]