Структуры данных.

Массивы

Структура (структурированный тип данных) – некоторый составной тип данных, составленный из базовых (скалярных) типов.

A

4	55
3	- 45
2	0
1	29
0	- 5

A[1]=29
A[3]= - 45

Одномерный массив (вектор)

B

	0	1	2	3
0	3	7	- 5	- 4
1	8	6	- 2	15
2	- 3	- 24	9	55
3	- 1	14	- 27	12
4	95	36	0	11

B[0,0]=3
B[2,1]= - 24

Прямоугольный двумерный массив (матрица)

Массив – это набор однотипных переменных, на которые ссылаются по общему имени.

Массивы можно создавать из элементов любого (но одного и того же) типа.

Массивы могут иметь одно или несколько измерений. По каждому из измерений массив имеет определенную длину (количество элементов).

К определенному элементу в массиве обращаются по его индексу (номеру), если массив многомерный, то по нескольким индексам (номерам в измерениях).

Такой способ доступа называется прямым или произвольным (в отличие от последовательного, который применяется в таких структурах данных, как списки – будут рассмотрены позже).

Массивы предполагают удобные средства группировки и обработки связанной информации. Элементы массива расположены в памяти непосредственно друг за другом. Если индекс массива изменять в цикле, можно получить доступ к каждому из элементов массива при использовании минимального числа операторов языка программирования.

Одномерные массивы в Pascal

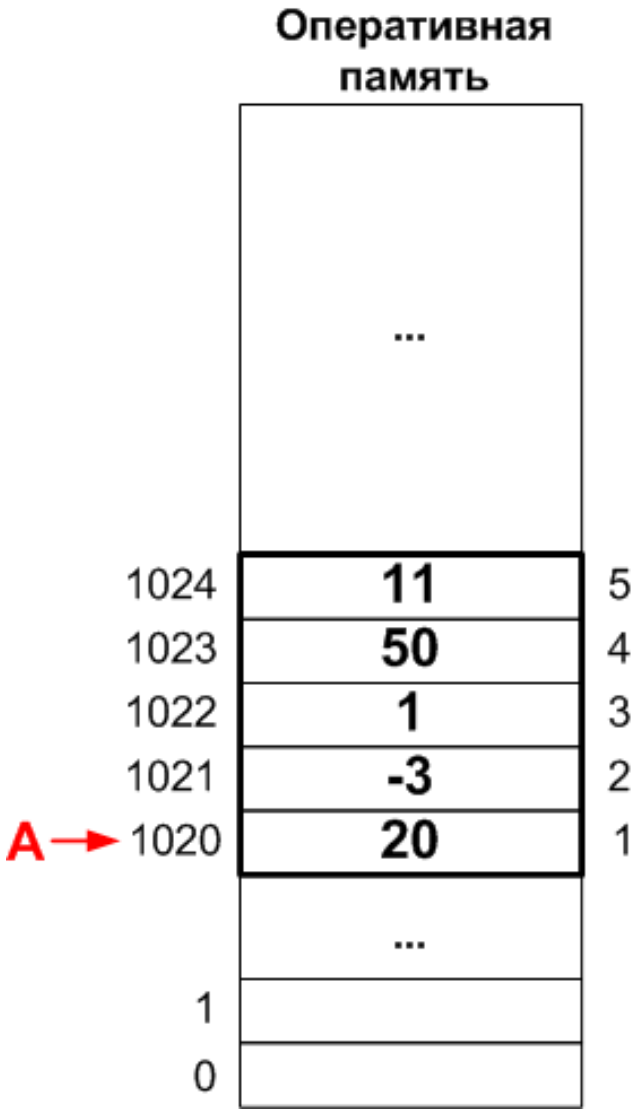
В Pascal массив из пяти элементов типа integer можно задать так:

```
var A: array [0..4] of  integer;
…
A[0]:=20; A[1]:=-3; A[2]:=1; A[3]:=50; A[4]:=11;
```

Индекс (номер элемента) массива может принимать значение от 0 до 4.

Некоторые программисты предпочитают начинать значения индекса с 1. Тогда массив можно задать так:

```
var A: array [1..5] of  integer;
…
A[1]:=20; A[2]:=-3; A[3]:=1; A[4]:=50; A[5]:=11;
```



Элементы одномерного массива располагаются в памяти компьютера последовательно в соседних ячейках (словах).

Компилятор переводит имена переменных в адреса ОП. Адреса слов ОП на рисунке обозначены условно(для примера). Размер слова ОП здесь − 2 байта. Поскольку одна переменная типа integer имеет размер 2 байта, то весь массив займет в памяти 5*2=10 байтов.

В приведенном выше фрагменте программы значения элементов массива задавались с помощью операции присваивания.
Более универсальным является способ задания значений элементов массива путем их ввода с терминала во время выполнения программы.

Для этого можно использовать следующий фрагмент программы:

```
...
for  i:=1 to 5 do
   begin
      write ('A[', i, ']=');
      readln (A[i]);
   end
...
```

Следующая программа вычисляет сумму элементов массива. В массиве из 100 элементов можно использовать (например, для удобства отладки программы, чтобы поменьше элементов вводить) всего m элементов (1<=m<=100)

Файл aip4_1.pas

```
program sum;
 const n=100;
 type tarr1=array[1..n] of real;
 var
  ar1:tarr1;
  i, m:integer;
  S:real;
 begin
  write('Vvedite chislo elementov menshe ili ravno ',n,':');
  readln (m);
  for i:=1 to m do
     begin
       write('ar1[',i,']=');
       readln(ar1[i]);
     end;
  S:=0;
  for i:=1 to m do S:=S+ar1[i];
  writeln ('S=',S:5:2);
end.
```

В массиве типа tarr1 100 элементов типа real

Переменная-массив типа tarr1

Из 100 элементов для упрощения работы будем использовать m элементов

Переменная для суммы элементов массива

Ввод используемого числа элементов массива

Цикл по вводу элементов массива

Присвоение начального значения сумме

Цикл по вычислению суммы элементов

Вывод суммы

Программа работает при любом m от 1 до 100. Проверьте ее работу, например, при m=5.

Структурирование программ.

Подпрограммы-процедуры и подпрограммы-функции

Подпрограмма - это отдельная функционально независимая часть программы. Любая подпрограмма обладает той же структурой, которой обладает и вся программа.

Подпрограммы решают три важные задачи:

1) избавляют от необходимости многократно повторять в тексте программы аналогичные фрагменты;

2) улучшают структуру программы, облегчая ее понимание;

3) повышают устойчивость к ошибкам программирования и непредвиденным последствиям при модификациях программы.

Очень важно понимать, что в подпрограмму желательно выделять любой законченный фрагмент программы. В качестве ориентиров просмотрите следующие рекомендации:

1) Когда Вы несколько раз перепишите в программе одни и те же последовательности команд, необходимость введения подпрограммы приобретает характер острой внутренней потребности.

2) Иногда слишком много мелочей закрывают главное. Полезно убрать в подпрограмму подробности, заслоняющие смысл основной программы.

3) Полезно разбить длинную программу на составные части - просто как книгу разбивают на главы. При этом основная программа становится похожей на оглавление.

4) Бывают сложные частные алгоритмы. Полезно отладить их отдельно в небольших тестирующих программах. Включение программ с отлаженными алгоритмами в основную программу будет легким, если они оформлены как подпрограммы.

5) Все, что Вы сделали хорошо в одной программе, Вам захочется перенести в новые. Для повторного использования таких частей лучше сразу выделять в программе полезные алгоритмы в отдельные подпрограммы.

Подпрограммы могут быть стандартными, т.е. определенными системой, и собственными, т.е. определенными программистом.

Стандартная подпрограмма (процедура или функция) - подпрограмма, включенная в библиотеку программ ЭВМ, доступ к которой обеспечивается средствами языка программирования. Вызывается она по имени с заданием фактических параметров соответствующих типов, заданных в описании данной подпрограммы в библиотеке процедур и функций.

Передача параметров в подпрограммы

Подпрограмма может иметь или не иметь параметры.

Формальные параметры подпрограммы указываются в описании подпрограммы с указанием соответствующих типов (после имени подпрограммы в круглых скобках). Фактические параметры указываются при вызове подпрограммы (после имени подпрограммы в круглых скобках).

Компилятор проверяет соответствие числа фактических параметров числу формальных параметров, а, также, соответствие типов фактических параметров типам формальных параметров.

Параметры могут передаваться в подпрограмму по значению или по ссылке (адресу).

Передача по значению: передается только значение переменной, указанной в качестве фактического параметра. Это значение присваивается внутренней (локальной) переменной подпрограммы (т.е. создается локальная копия фактического параметра), и все действия затем производятся с этой внутренней переменной, никак не отражаясь на значении переменной, использованной в качестве фактического параметра. Т.е. подпрограмма не может изменить значение фактического параметра, переданного по значению.

Передача по ссылке: в подпрограмму передается адрес переменной, имя которой указано в качестве фактического параметра, и все действия в подпрограмме производятся именно над этой переменной. Т.е. метод может изменить значение фактического параметра, переданного по ссылке (адресу).

В Turbo Pascal переменная любого типа передается по ссылке, если перед соответствующим формальным параметром в описании подпрограммы стоит слово var , а если этого слова нет – по значению.

Как уже говорилось, подпрограмма может вообще не иметь параметров.

Подпрограммы-процедуры и подпрограммы-функции

Если подпрограмма возвращает значение, ее называют подпрограммой- функцией. Примером могут служить библиотечные функции sin(x), ln (x), exp(x) и другие.

Если подпрограмма просто выполняет последовательность действий, но не возвращает значения, то ее называют подпрограммой-процедурой.

Задача. Ввести два массива вещественных чисел A и B, состоящих из m элементов (1<=m<=100). Из суммы элементов массива A вычесть сумму элементов массива B. Ввод массива оформить как процедуру, вычисление суммы элементов массива оформить как функцию.

В качестве параметров подпрограмм (процедур или функций) целесообразно задавать те переменные, значения которых будут меняться при каждом вызове подпрограммы. Например, для нашей задачи процедура ввода массива должна быть вызвана дважды: один раз для ввода элементов массива A, второй раз – для ввода элементов массива B. Поэтому имя массива должно передаваться в процедуру как параметр (*параметр – то, что изменяется*).

Мы не всегда будем вводить в массив все 100 элементов (максимальное количество элементов в массиве). Например, чтобы проверить правильность работы программы, достаточно ввести в массив 5 (или даже 2 элемента) и оценить результат работы с пятью элементами. Число вводимых элементов массива зададим в процедуре в качестве второго параметра.

Функцию, подсчитывающую и возвращающую сумму элементов массива, мы должны использовать дважды: для массива A и для массива B. Значит, массив, с которым работает функция, нужно задавать в качестве параметра. Функция будет более универсальной, если мы зададим в качестве параметра и число элементов массива, которые необходимо просуммировать. Тогда можно вычислять сумму не всех элементов массива, а только m первых.

Поскольку, функция возвращает значение, ее вызов можно использовать в выражении, например,

```
r := sum(A,m) − sum(B,m);
```

Файл aip4_2.pas

```
program prog1;
  const  n=100;
  type  tarr1=array[1..n] of real;
  var
    A, B: tarr1;
    m: integer;
    r: real;
    procedure inputarr(var X:tarr1; k:integer);
      var i: integer;
      begin
        for i:=1 to k do
          begin
            write('element ',i,': ');
            readln(X[i]);
          end
      end;
    function sum(var X:tarr1; k:integer):real;
      var i: integer; S: real;
      begin
        S:=0;
        for i:=1 to k do S:=S+X[i];
        sum:=S;
      end;
```

Глобальные переменные – определены в основной программе

Локальные переменные процедуры

Процедура ввода k ‘элементов в массив типа tarr1 (описание)

Локальные переменные функции

Функция, возвращающая сумму k элементов массива типа tarr1 (описание)

```
begin
   write('Vvedite chislo elementov menshe ili ravno ',n,': ');
   readln (m);
   writeln('Vvedite massiv A');
   inputarr(A,m);
   writeln('Vvedite massiv B');
   inputarr(B,m);
   r:=sum(A,m)-sum(B,m);
   writeln ('r=',r:5:2);
end.
```

Вызов процедуры

Вызов процедуры

Вызовы функции

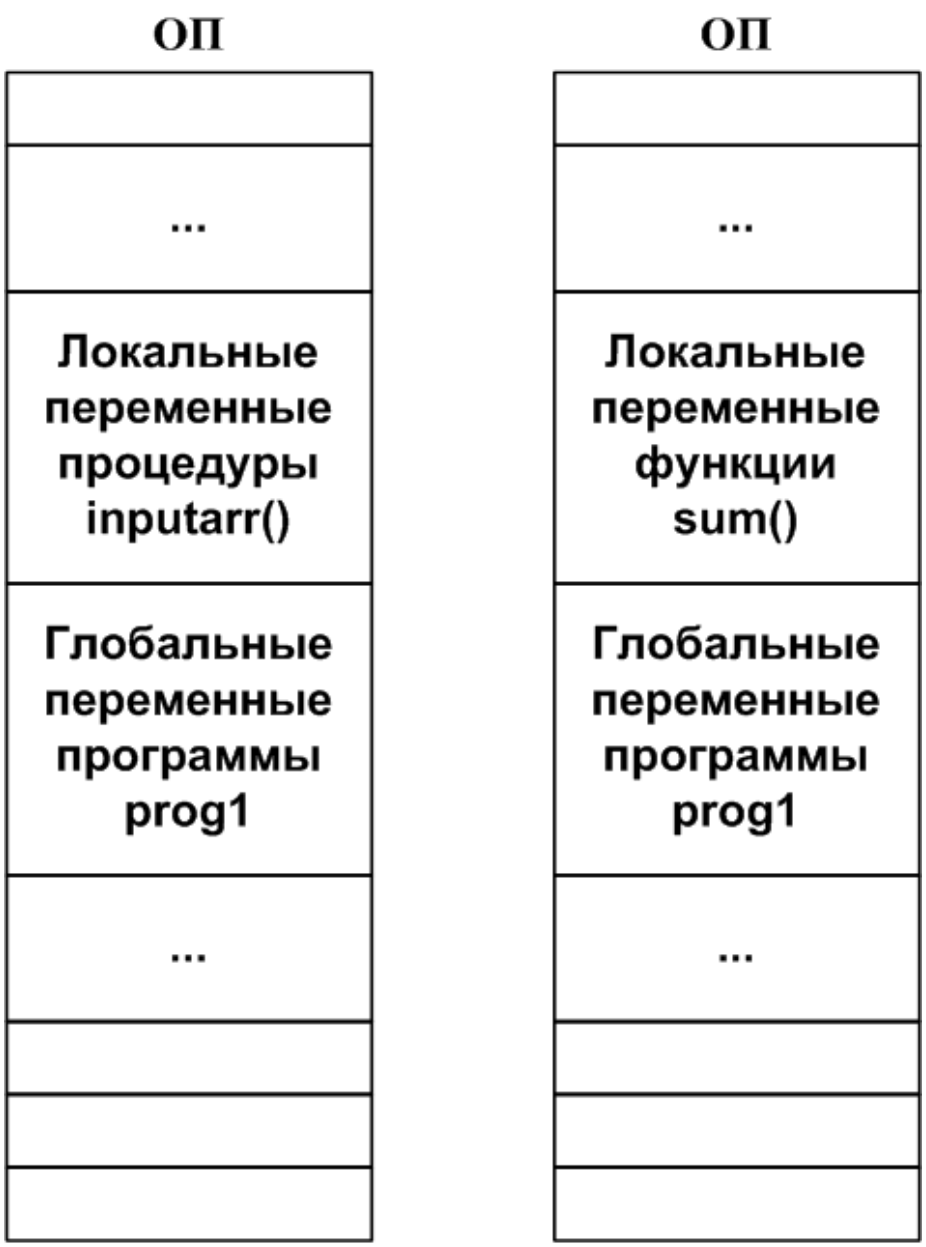
Основная (головная) программа

Проверьте работу программы, например, при m=5.

Программе для ее работы выделяется определенная область ОП. Одна часть этой области отводится под хранение глобальных переменных программы (назовем ее областью глобальных переменных). Другая часть выделенной области памяти отводится под хранение локальных переменных (назовем ее областью локальных переменных).

Глобальные переменные программы создаются в области глобальных переменных в начале работы программы и существуют до момента завершения программы. Доступ к ним возможен из любого места программы, в том числе, и из подпрограмм (они видны отовсюду). Т.е. подпрограмма может изменить значение глобальной переменной.

Локальные переменные создаются в области локальных переменных при входе в подпрограмму и удаляются при выходе из нее. Локальные переменные подпрограммы видны только внутри этой подпрограммы во время ее работы и доступ к ним возможен только из этой подпрограммы. После выхода из подпрограммы получить значения локальных переменных этой подпрограммы невозможно, т.к. соответствующая область памяти уже считается свободной (переменные разрушены).



Вид ОП во время работы процедуры inputarr()

Вид ОП во время работы функции sum()

Следует отличать определение подпрограммы (процедуры или функции) и вызов подпрограммы. В нашей программе prog1:

определение процедуры (это как бы описание действий в общем виде):

```
procedure inputarr(var X:tarr1; k:integer);
    var i: integer;
    begin
      for i:=1 to k do
        begin
          write('element ',i,': ');
          readln(X[i]);
        end
    end;
```

Формальные параметры

вызов процедуры (осуществляется в основной программе и передает процедуре конкретные значения, с которыми нужно выполнять описанные в определении процедуры действия):

```
inputarr(A,m);
```

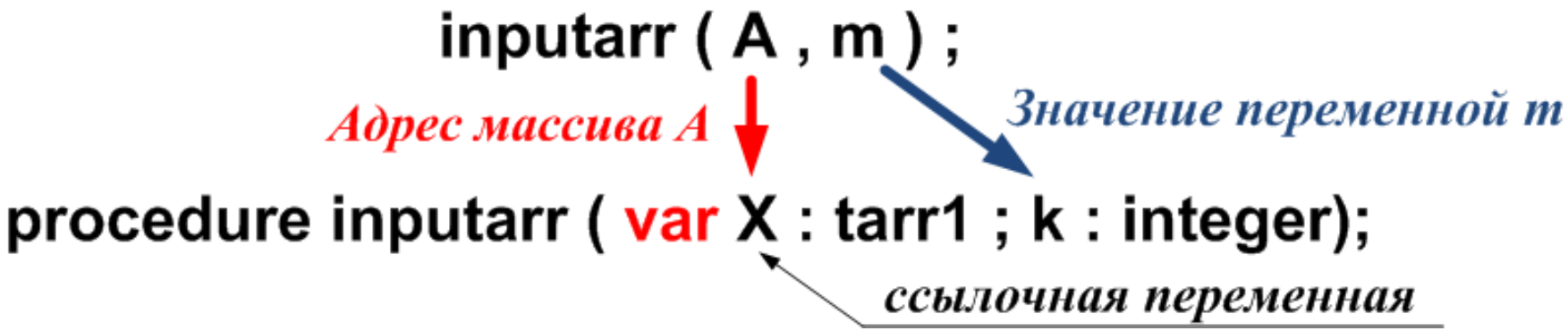
Фактические параметры

Компилятор проверяет соответствие числа фактических параметров числу формальных параметров, а, также, соответствие типов фактических параметров типам формальных параметров.

В данном случае в процессе компиляции проверяется, что в вызове указано два параметра (столько же, сколько и в определении заголовка процедуры). Первый передаваемый в процедуру фактический параметр A должен иметь тип tarr1, второй фактический параметр m должен иметь тип integer (как в описании).

Что происходит в процессе выполнения программы prog1 при вызове процедуры inputarr()?

Схема передачи параметров в подпрограмму:



Слово “var” перед именем локальной переменной X в описании формальных параметров говорит о том, что переменная X, будет иметь ссылочный тип (другое название – указатель), и ей при передаче фактического параметра A будет присвоен адрес этого параметра в ОП (т.е. адрес первого элемента массива A). Так работает передача параметра по ссылке. Процедура будет работать именно с глобальным массивом A, обращаясь к нему через ссылочную переменную X. Все изменения, вносимые процедурой в глобальный массив A, останутся в этом массиве после выхода из процедуры, т.е. данные будут успешно введены в массив A.

Перед локальной переменной k в описании формальных параметров нет слова var, поэтому передача параметра будет осуществляться по значению. Другими словами, локальной переменной k будет присвоено значение глобальной переменной m, указанной в вызове процедуры в качестве фактического параметра. Все изменения, производимые с локальной переменной k в процедуре никак не будут влиять на значение глобальной переменной m.

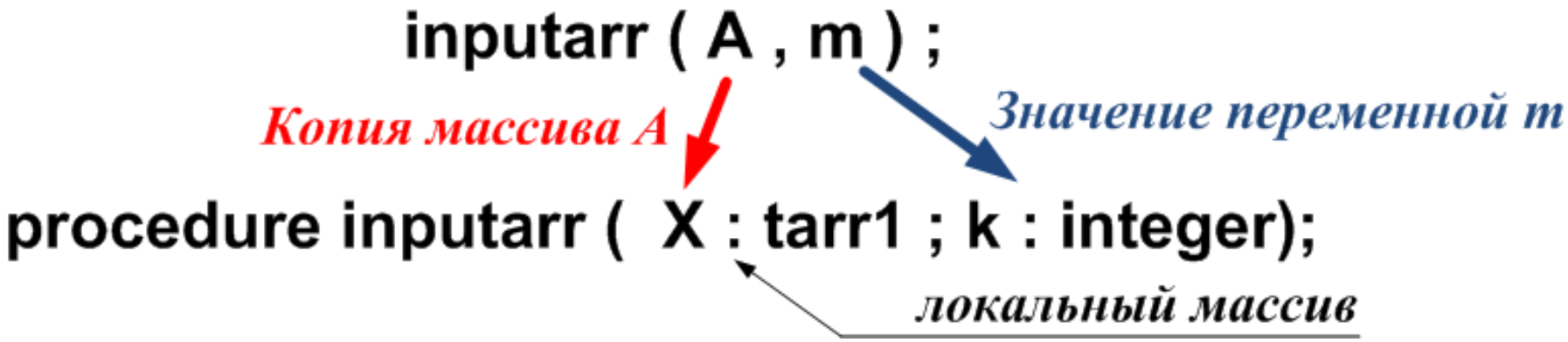
Значения адресов ОП взяты условно (для примера).

Схема построена с учетом того, что переменные различных типов занимают соответствующее число байтов:
integer – 2;
real – 6;
ссылка – 4.

Описание переменных программы prog1

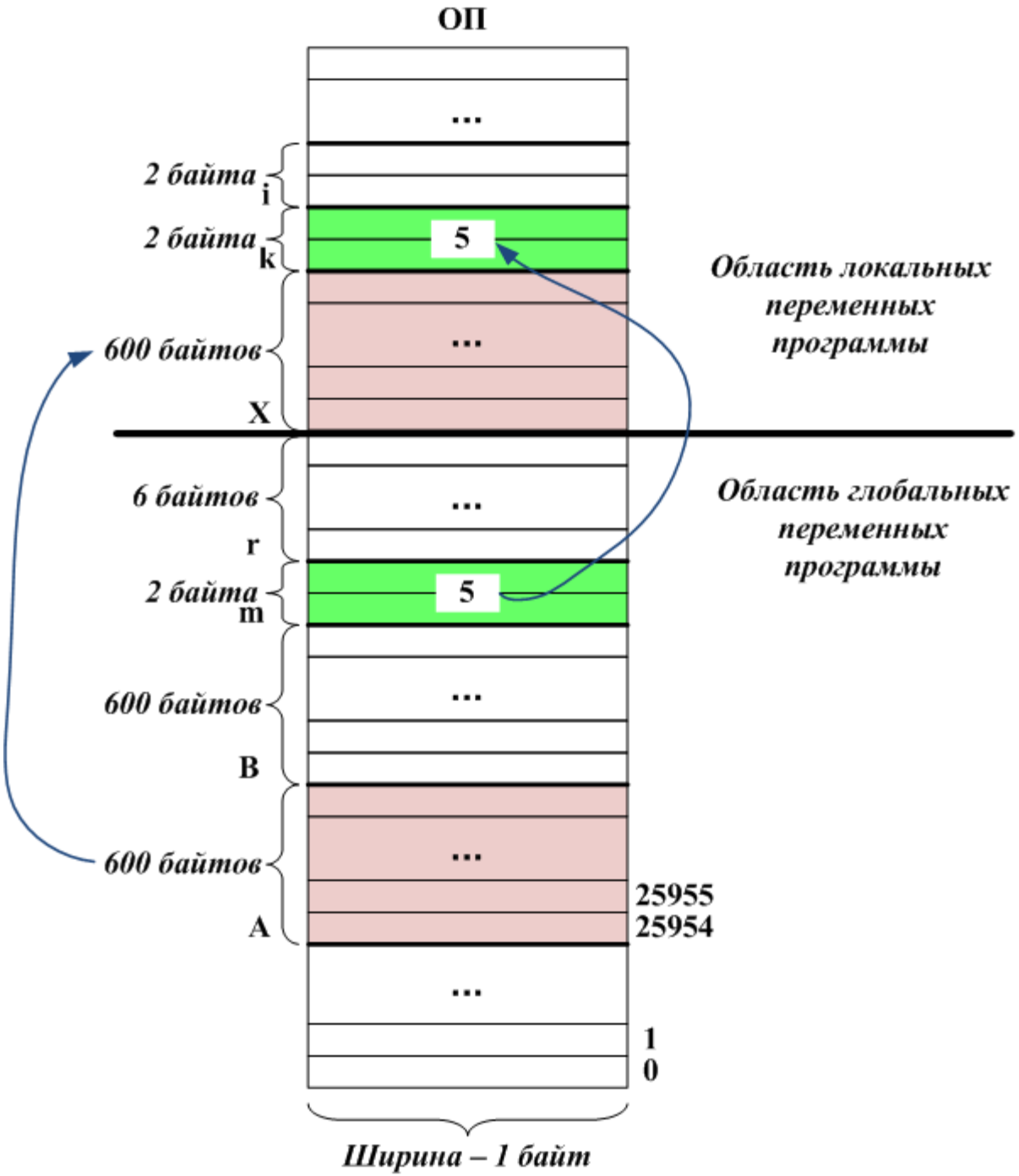
№ п/п	Имя переменной	Тип	Размер в байтах	Область видимости	Период существования
Глобальные переменные программы					
1.	A	tarr1=array[1..n] of real; (n=100)	6*100=600	вся программа prog1	все время работы программы prog1
2.	B	tarr1=array[1..n] of real; (n=100)	6*100=600		
3.	m	integer	2		
4.	r	real	6		
Локальные переменные процедуры inputarr()					
5.	X	pointer (ссылка)	4	процедура inputarr()	время работы процедуры inputarr()
6.	k	integer	2		
7.	i	integer	2		
Локальные переменные функции sum()					
8.	X	pointer (ссылка)	4	функция sum()	время работы функции sum()
9.	k	integer	2		
10.	i	integer	2		
11.	S	real	6		

А если мы в заголовке процедуры не поставим слово var перед параметром X?



Тогда первый параметр, как и второй, будет передаваться по значению. Т.е. будет создан локальный массив X из 100 элементов типа real, и значениям элементов массива X будут присвоены значения соответствующих элементов массива A. Все действия, предусмотренные в описании процедуры, будут производиться не с самим глобальным массивом A, а с его локальной копией X, которая будет удалена после выхода из процедуры. Для некоторых задач создание локальной копии может быть и удобно, но для нашей задачи это приведет к пренеприятнейшим последствиям: значения элементов введутся не в глобальный массив A, а в локальный массив X и уничтожатся после выхода из процедуры ввода.

Запустите программу с указанными выше изменениями (без var в описании формальных параметров) − файл aip4_3.pas − и оцените результат.


ОП
...
2 байта
i
2 байта
k
5
Область локальных переменных программы
600 байтов
...
X
6 байтов
...
r
Область глобальных переменных программы
2 байта
m
5
600 байтов
...
B
600 байтов
...
25955
25954
A
...
1
0
Ширина – 1 байт

При задании формальных параметров функции sum() наличие слова “var” перед именем массива не играет такой решающей роли, т.к. функция не меняет значения самих элементов массива, а только вычисляет и возвращает их сумму. В принципе, эти вычисления можно сделать и над локальной копией, но зачем тратить на ее создание ресурсы памяти и времени компьютера?

Целесообразно все-таки слово “var” оставить:

```
function sum(var X:tarr1; k:integer):real;
```

Использование глобальных переменных в подпрограмме

Подпрограмма может работать с глобальными переменными программы, т.к. они видны из любого места программы. У некоторых подпрограмм может вообще не быть параметров.

В нашей задаче в головной программе вводится число элементов массива m, с которыми осуществляется работа. Поэтому из заголовков процедуры и функции второй параметр в принципе можно убрать, а в теле процедуры вместо локальной переменной k использовать глобальную переменную m. Подпрограммы от этого станут менее универсальными, но логика работы программы для рассматриваемой задачи от этого не пострадает.

Файл aip4_4.pas

```
program prog1;
  const n=100;
  type tarr1=array[1..n] of real;
  var
    A,B:tarr1;
    m:integer;
    r:real;
    procedure inputarr(var X:tarr1);
       var i:integer;
       begin
         for i:=1 to m do
           begin
             write('element ',i,': ');
             readln(X[i]);
           end

       end;
    function sum(var X:tarr1):real;
       var i:integer; S:real;
       begin
         S:=0;
         for i:=1 to m do S:=S+X[i];
         sum:=S;
       end;
```

```
begin
  write('Vvedite chislo elementov menshe ili ravno ',n,': ');
  readln (m);
  writeln('Vvedite massiv A');
  inputarr(A);
  writeln('Vvedite massiv B');
  inputarr(B);
  r:=sum(A)-sum(B);
  writeln ('r=',r:5:2);
end.
```

Красным цветом выделены изменения в программе. Результат при тех же исходных данных тот же (проверьте).

Вывод: В качестве параметров в подпрограммах нужно задавать те элементы, которые от вызова к вызову могут меняться. Например, если сумму элементов массива в программе нужно вычислять для нескольких массивов, то целесообразно задать имя обрабатываемого массива как параметр.

Если же разбиение программы на подпрограммы используется только для улучшения ее структуры, то подпрограммы могут вообще не иметь параметров, а работать с глобальными переменными программы.

Умозаключения об области действия имен переменных:

1) **В различных подпрограммах имена локальных переменных могут совпадать (это здорово, т.к. не надо изощряться в придумывании новых имен). Хоть имена и совпадают, но это разные переменные, т.к. время их жизни не совпадает (локальные переменные создаются при входе в подпрограмму и разрушаются (освобождаются) при выходе из нее.**

2) **Имена локальных переменных подпрограммы и глобальных переменных программы также могут совпадать. Хотя имена и совпадают, но это разные переменные, т.к. они находятся в разных областях памяти, выделенной программе (глобальные – в области глобальных переменных, локальные – в области локальных переменных).**

3) **Локальная переменная подпрограммы делает недоступной (скрывает) для этой подпрограммы глобальную переменную с тем же именем (забыв об этом, многие программисты «набили себе шишки»).**

Вопрос (на засыпку):

Если внутри подпрограммы в свою очередь определена подпрограмма, какова будет область видимости локальных переменных последней?
Будут ли видны локальные переменные внутренней подпрограммы (подпрограммы второго уровня) из внешней (вызывающей) подпрограммы (подпрограммы первого уровня)? Будут ли из подпрограммы второго уровня видны локальные переменные подпрограммы первого уровня? Будут ли из подпрограммы второго уровня видны глобальные переменные программы?

(Вопрос справедлив для Pascal, в Java используется всего один уровень вложенности подпрограмм).

Учебная практика (задание 1)

Пример. Разработать программу обработки матриц.
Тип элементов матрицы – real.
Максимальная размерность матрицы 50x100.
Подпрограммы:
1) процедура ввода матрицы из текстового файла;
2) процедура вывода матрицы в окно терминала;
3) процедура вывода одномерного массива (вектора) в окно терминала;
4) процедура, заполняющая вектор, j-ый элемент которого равен максимальному элементу j-го столбца матрицы.

Головная программа должна демонстрировать применение вышеуказанных подпрограмм для двух матриц различной размерности.

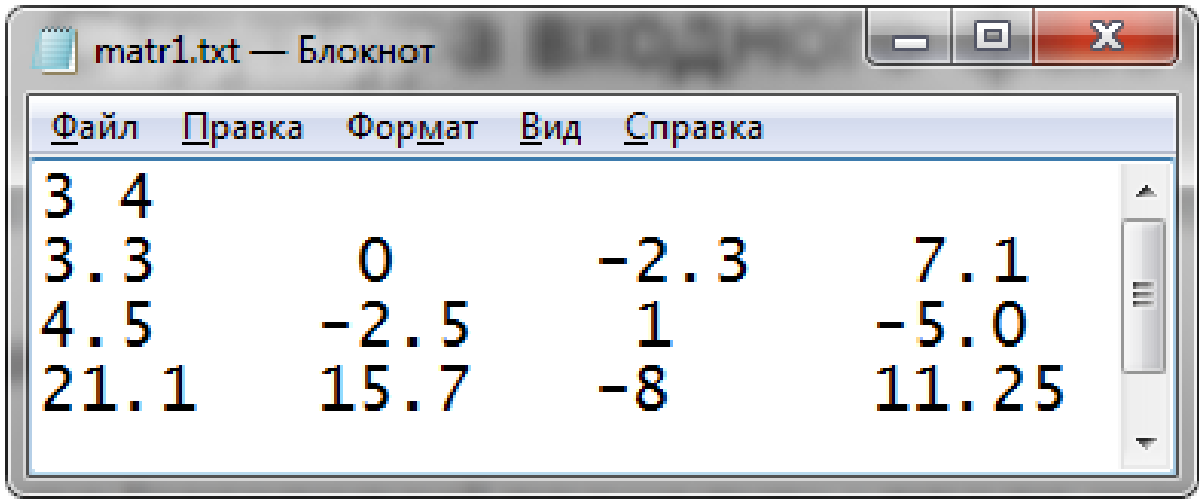
Структура входного файла:

Число строк, число столбцов, элементы матрицы.

В каждом файле находится одна матрица.

Сигнатура (заголовок) процедуры ввода матрицы из текстового файла:

```
procedure inputMatr(fileName: string; var X:tmatr; var
n:integer; var m:integer);
```



Процедура №4.

Сформировать массив X, содержащий максимальные элементы столбцов матрицы Y.

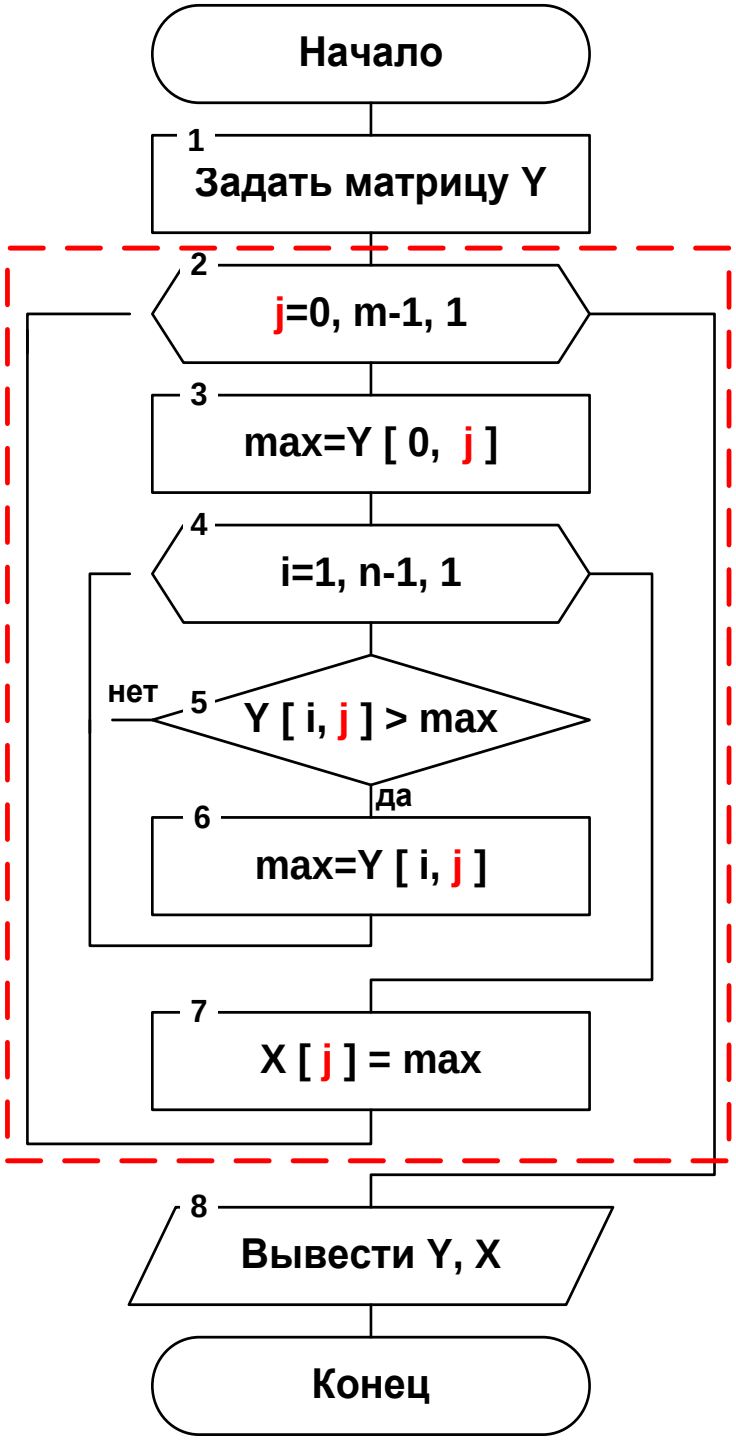
Y

	0	1	2	3
0	3	72	- 5	- 4
1	8	6	- 2	- 15
2	- 3	- 24	9	- 55
3	- 1	14	- 27	- 1
4	95	36	0	- 11

X

95	72	9	- 1

n – число строк в матрице Y,
m – число столбцов в матрице Y и элементов в векторе Y.



Задача интересна тем, что внешний цикл организован по столбцам, а внутренний по строкам.

На каждой итерации внешнего цикла фиксируется номер столбца. Внутренний цикл «пробегает» по всем номерам строк, т.е. перебираются все элементы в зафиксированном столбце.

Выделенную красным часть нужно оформить в виде процедуры (зачем?). Какие параметры будете передавать и каким способом?