Разбор заданий контрольной работы

Задание 1

```
program my;
var
z,c1,c2,c3:integer;

begin
readln(z);
c1:=z div 100;
c2:=(z mod 100) div 10;
c3:=z mod 10;
writeln(‘1-я цифра=',c1,‘2-я цифра=',c2,‘3-я цифра=',c3);
z:=c3*100+c2*10+c1;
writeln(z);
end.
```

Задание 3

```
program my_prog;
var
k,i,j:integer;
begin
k:=0;
for i:=100 to 200  do
begin
j:=i mod 3;
if j=0 then
k:=k+1;
end;
Writeln(‘Число значений=',k);
end.
```

Задание 4

```
program my;
var
i,n,s,k:integer;
begin
s:=0; {summa}
i:=0; {counter}
readln(n);   {vvod}
   repeat
   s:=s+i;
   i:=i+1;
   until  s>n;
Writeln('Число, которое введено с клавиатуры=',n);
Writeln('Сумма=',s);
writeln('Всего чисел=',i);
end.
```

Задание 5

```
program my;
var
mas:array[1..31] of Integer;
i:integer;
begin
For i:=1 to 31 do
Begin
mas[i]:=random(25)+3;
if mas[i]<=9 then
writeln(mas[i],’ осадков выпало',i,' января');
End;
end.
```

Задание 6.

```
program my;
var
i,s:integer;
mas:array[1..10] of integer;
begin
s:=0;
for i:=1 to 10 do
begin
mas[i]:=random(201)-100;
writeln(mas[i]);
s:=s+mas[i];
end;
writeln('summa=',s);
i:=1;
s:=0;
while mas[i]>0 do
begin
s:=s+mas[i];
end;
writeln('summa2=',s);
end.
```

Задание 7

```
program my;
var
i,buf,maxi:integer;
mas:array[1..10] of integer;
begin
buf:=0;
for i:=1 to 10 do
begin
mas[i]:=random(21);
writeln(mas[i]);
end;
for i:=1 to 10 do
begin
if mas[i]>buf then
buf:=mas[i];
maxi:=i;
end;
```

```
mas[maxi]:=mas[1];
mas[1]:=buf;

for i:=1 to 10 do
writeln(mas[i]);
end.
```

Процедуры и функции

Процедуры и функции

Разновидностью программ являются **подпрограммы**.

Подпрограмма – как любая программа может иметь все полагающиеся программе атрибуты: имя, разделы описания меток **(label),** констант **(const),** типов **(type),** переменных **(var)** и может содержать вложенные функции и процедуры.

В языке Pascal имеется два вида подпрограмм:

процедуры и функции.

Упоминание в тексте программы имени процедуры (или функции) является ее вызовом.

Объявление и описание подпрограммы.

Подпрограммы объявляются и описываются в начале программы, до ключевого слова **begin,** означающего начало тела программы.

Объявление функции

function <имя_функции>
[(<список_параметров>)]:<тип_результата>;

Обьявление процедуры

Procedure <имя процедуры> [(< список_параметров >)]

В отличие от констант и переменных, объявление подпрограммы может быть оторвано от ее описания. В этом случае после объявления нужно указать ключевое слово forward:

function <имя_функции>
[(<параметры>)]:<тип_результата>; forward;

Описание подпрограммы

Если описание подпрограммы имеется, оно идет после ее объявления по следующей схеме (единой для процедур и функций):

[uses <имена_подключаемых_модулей>;][label <список_меток>;][const <имя_константы> = <значение_константы>;][type <имя_типа> = <определение_типа>;][var <имя_переменной> : <тип_переменной>;] [procedure <имя_процедуры> <описание_процедуры>][function <имя_функции> <описание_функции>;] begin {начало тела подпрограммы} <операторы>end; (* конец тела подпрограммы *)

Если объявление подпрограммы было оторвано от ее описания, то описание начинается дополнительной строкой с указанием только имени подпрограммы:

function <имя_подпрограммы>; или

procedure <имя_подпрограммы>;

Описания двух различных подпрограмм не могут пересекаться. Но внутри любой подпрограммы могут быть описаны другие процедуры или функции - **вложенные**.

Пример подпрограммы-процедуры:

```
Program pr1;
procedure err(c:byte; s:string);
var zz: byte;
begin
if c = 0 then
writeln(s)
else
writeln('Ошибка!');
end;
var
n:byte;
begin
readln(n);
err(n,’krona’);
end.
```

Пример подпрограммы-функции

```
Function Power(a,b:real):real;
Begin
If a>0 then
Power:=exp(b*ln(a))
Else if a<0 then
Power:=exp(b*ln(abs(a)))
Else if b=0 then
Power:=1
Else
Power:=0
End;
```

Список параметров

В заголовке подпрограммы (в ее объявлении) указывается список **формальных параметров** - переменных, которые принимают значения, передаваемые в подпрограмму извне во время ее вызова.

Procedure <имя> [(<список_формальных_параметров>)];

Function <имя> [(<список_формальных_параметров>)]:<тип>;

Список параметров может и вовсе отсутствовать:

```
procedure proc1;
function func1: boolean;
```

В этом случае подпрограмма не получает никаких переменных "извне".

Отсутствие параметров и, как следствие, передаваемых извне значений не означает, что при каждом вызове подпрограмма будет выполнять абсолютно одинаковые действия. Поскольку глобальные переменные видны изнутри любой подпрограммы, их значения могут неявно изменять внутреннее состояние подпрограмм.

Если же параметры имеются, то каждый из них описывается по следующему шаблону:

[<способ_подстановки>]<имя_параметра>:<тип>;

Если способ подстановки и тип нескольких параметров совпадают, описание этих параметров можно объединить:

[<способ_подстановки>]<имя1>,...,<имяN>: <тип>;

Имена локальных переменных, описываемые в разделе var внутреннем для подпрограммы, не могут совпадать с именами параметров этой же подпрограммы.

Пример:

function func2(a,b:byte; var x,y,z:real; const c:char);

В заголовке подпрограммы можно указывать **только простые (не составные)** типы данных. **Нельзя записать:**

procedure proc2(a: array[1..100]of char);

Чтобы обойти это ограничение, составной тип данных нужно описать в разделе **type**, а при объявлении подпрограммы воспользоваться именем этого типа:

type arr = array[1..100] of char;

procedure proc2(a: arr);

function func2(var x: string): arr;

Возвращаемые значения

Основное различие между функциями и процедурами состоит в количестве возвращаемых ими значений.

Любая функция, завершив свою работу, должна вернуть основной программе (или другой вызвавшей ее подпрограмме) **ровно одно значение**, причем его тип нужно явным образом указать уже при объявлении функции.

Для возвращения результата применяется специальная **"переменная", имеющая имя, совпадающее с именем самой функции.** Оператор присваивания значения этой "переменной" обязательно должен встречаться в теле функции хотя бы один раз.

Например:

```
function min(a,b: integer): integer;
begin
if a>b  then
min:= b
else
min:= a
end;
```

В отличие от функций, процедуры явно не возвращают никаких значений.

Вызов подпрограмм

Любая подпрограмма может быть вызвана не только из основного тела программы, но и из любой другой подпрограммы, **объявленной позже нее**.

При вызове в подпрограмму передаются **фактические** параметры или аргументы (в круглых скобках после имени подпрограммы, разделенные запятыми):

<имя_подпрограммы>(<список_аргументов>)

Аргументами могут быть переменные, константы и выражения, включающие в себя вызовы функций.

Количество и типы передаваемых в подпрограмму аргументов должны соответствовать количеству и типам ее параметров.

Тип каждого аргумента должен учитывать способ подстановки, указанный для соответствующего параметра. Если у подпрограммы вообще нет объявленных параметров, то при вызове список передаваемых аргументов будет отсутствовать вместе с обрамляющими его скобками.

Вызов функции не может быть самостоятельным оператором, потому что возвращаемое значение нужно куда-то записывать. Но может быть частью выражения. Например:

```
c:= min(a,a*2);if min(z, min(x,y))= 0 then...;
```

Вызов процедуры является отдельным оператором в программе, так как процедура не возвращает значения. Например:

```
err(res,'Привет!');
```

После того как вызванная подпрограмма завершит свою работу, **управление передается оператору, следующему за оператором, вызвавшим эту подпрограмму.**

Способы подстановки аргументов

- **<пустой>; параметр-значение**
- **var; параметр-переменная**
- **const. параметр-константа**

1-й способ подстановки. Параметр-значение

Описание

В списке параметров подпрограммы перед параметром-значением служебное слово отсутствует. Например, функция func3 имеет три параметра-значения:

```
function func3(x:real; k:integer; flag:boolean):real;
```

При вызове подпрограммы параметру-значению может соответствовать аргумент, являющийся выражением, переменной или константой, например:

```
dlina:= func3(shirina/2, min(a shl 1,ord('y')), true)+0.5;
```

Для типов данных здесь не обязательно строгое совпадение (эквивалентность), достаточно и совместимости по присваиванию

Механизм передачи значения

В области памяти, выделяемой для работы вызываемой подпрограммы, создается переменная с именем **<имя_подпрограммы>.<имя_параметра>,** и в эту переменную записывается значение переданного в соответствующий параметр аргумента. Все действия в подпрограмме выполняются именно над этой новой переменной. **Значение же входного аргумента не затрагивается.** Поэтому после окончания работы подпрограммы, **значение аргумента останется точно таким же**, каким оно было на момент вызова подпрограммы.

Например:

function func3(x:real; k:integer; flag:boolean):real;

При вызове **func3(1+a/2,a,true)** будут выполнены следующие действия:

1. создать временные переменные **func3.x, func3.k, func3.flag;**
2. вычислить значение выражения **1+a/2** и записать его в переменную **func3.x;**
3. записать в переменную **func3.k** значение переменной **a;**
4. записать в переменную **func3.flag** значение константы **true;**
5. произвести действия, описанные в теле функции;
6. уничтожить все временные переменные, в том числе **func3.x, func3.k, func3.flag.**

Значения аргументов не изменятся.

2-й способ подстановки. Параметр-переменная

Описание

В списке параметров подпрограммы перед параметром-переменной ставится служебное слово **var.** Например, процедура **proc3** имеет три параметра-переменные и один параметр-значение:

```
procedure proc3(var x,y:real; var
    k:integer;flag:boolean);
```

При вызове подпрограммы параметру-переменной может соответствовать только аргумент-переменная; **константы и выражения запрещены.**

Кроме того, тип аргумента и тип параметра-переменной должны быть эквивалентными

Механизм передачи значения

В отличие от параметра-значения, **для параметра-переменной не создается копии при вызове подпрограммы.** В работе подпрограммы участвует та самая переменная, которая послужила аргументом. Если ее значение изменится в процессе работы подпрограммы, то это изменение сохранится и после того, как будет уничтожен контекст подпрограммы. Поэтому аргументы должны соответствовать параметрам-переменным: ни константа, ни выражение не смогут сохранить изменения, внесенные в процессе работы подпрограммы.

Параметры-переменные позволяют получать результаты работы процедур, а также увеличивать количество результатов, возвращаемых функциями.

3-й способ подстановки. Параметр-константа

Описание

В списке параметров подпрограммы перед параметром-константой ставится служебное слово **const.** Например, процедура **proc4** имеет один параметр-переменную и один параметр-константу:

```
procedure proc4(var k:integer; const flag:boolean);
```

При вызове подпрограммы параметру-константе может соответствовать аргумент, являющийся выражением, переменной или константой. Во время выполнения подпрограммы соответствующая переменная считается обычной константой. Ограничением является то, что при вызове другой подпрограммы из тела текущей параметр-константа не может быть подставлен в качестве аргумента в параметр-переменную.

Для типов данных здесь достаточно совместимости по присваиванию

Механизм передачи значения

Есть разные точки зрения на то создается ли для параметра-константы, как и для параметра-переменной, копия в момент вызова подпрограммы.

Выполнение в качестве проверки примера

```
var a: byte;
procedure prob(const c:byte);
begin
writeln(longint(addr(c)));          {физ.адрес параметра c}
end;
Begin
a:=0;
 writeln(longint(addr(a)));   {физ.адрес переменной a}
prob(a);
end.
```

показывает различие физических адресов переменной a
и параметра c. Отсюда вывод: копия создается.

Области действия имен. Разграничение контекстов

Глобальные объекты - это типы данных, константы и переменные, объявленные в начале программы до объявления любых *подпрограмм*. Эти объекты будут видны во всей программе, в том числе и во всех ее *подпрограммах*. Глобальные объекты существуют на протяжении всего времени работы программы.

Локальные объекты объявляются внутри какой-нибудь *подпрограммы* и "видны" только этой *подпрограмме* и тем *подпрограммам*, которые были объявлены как внутренние для нее. Локальные объекты не существуют, пока не вызвана *подпрограмма*, в которой они объявлены, а также после завершения ее работы.

```
program prog;
var a:byte;
procedure pr1 (p:byte);
var b:byte;   {первый уровень вложенности}
function f (pp:byte);
var c:byte;   {второй уровень вложенности}
begin   {здесь "видны" переменные a, b, c, p, pp}
end;
begin   {здесь "видны" переменные a, b, p}
end;
var g:byte;
procedure pr2;
var d:byte;     {первый уровень вложенности}
begin   {здесь видны переменные a, d, g}
end;
begin   {тело программы; здесь "видны" переменные a, g}
end;
```

Побочный эффект

Поскольку *глобальные переменные* видны в *контекстах* всех блоков, то их значение может быть изменено изнутри любой *подпрограммы*. Этот эффект называется ***побочным***, а его использование очень нежелательно, потому что может стать источником непонятных ошибок в программе.

Чтобы избежать *побочного эффекта*, необходимо строго следить за тем, чтобы *подпрограммы* изменяли только свои *локальные переменные* (в том числе и *параметры-переменные*).

Совпадение имен

Совпадение глобальных и локальных имен не вызывает коллизий, поскольку к каждому локальному имени неявно приписано имя той *подпрограммы*, в которой оно объявлено. Тем не менее, рекомендуется этого избегать.

Если имеются *глобальная* и *локальная переменные* с одинаковым именем, то изнутри *подпрограммы* к *глобальной переменной* можно обратиться, приписав к ней спереди имя программы:

<имя_программы>.<имя_глобальной переменной> Например

a:= prog.a; (*локальной переменной* здесь присваивается значение *глобальной*):

Нетипизированные параметры

В объявлении *подпрограммы* можно не указывать тип *параметра-переменной*:

```
procedure proc5(var x);
```

Такой *параметр* будет называться ***нетипизированным***. В этот *параметр* можно передать *аргумент*, относящийся к любому типу данных.

Для того чтобы внутри самой *подпрограммы* корректно обрабатывать значения, поступившие через *нетипизированный параметр*, существует два различных способа.

Явное преобразование типа

В *процедуре* `proc5` значение одного и того же *параметра* `x` интерпретируется тремя разными способами: как целое число, как вещественное число и как массив:

```
procedure proc5(var x);
type arr = array[1..10] of byte;
var x: integer;
z: real;
m: arr;
begin
...
y:= integer(x);
z:= real(x);
m:= arr(x);
 ...
end;
```

Совмещение в памяти

Второй способ: описать внутри *подпрограммы локальную переменную*, которая будет физически совпадать с переменной, передаваемой через *нетипизированный параметр*:

<локальная_переменная>: <тип> absolute <нетипизир_параметр>;

В этом случае будут совмещены значения, физически записанные в этих переменных, в точности так же, как это происходит при подстановке *аргумента* в *параметр-переменную*, однако без контроля за совпадением типов данных. Поэтому вполне возможна, например, ситуация, когда первые четыре байта строки (*аргумента*, переданного в *нетипизированный параметр*) будут восприниматься как `longint`-число:

```
function func5(var x):real;
var xxx: longint absolute x;
begin
  {здесь с началом любой переменной,
   поступившей в параметр x,   ... можно
   обращаться как с longint-числом:       при
   помощи локальной переменной xxx}
end;
```

Открытые параметры

Открытые параметры - это массивы и строки неопределенной длины. *Открытым параметром* может стать только *параметр-переменная*. Возможность работать с *открытыми параметрами* в *подпрограммах* появилась в версии Turbo Pascal 7.0.

Открытые массивы

В *параметр*, который является **открытым массивом**, можно передавать как *аргумент* массив любой длины и размерности. Единственное ограничение: типы компонент у этих двух массивов должны совпадать.

В заголовке *подпрограммы* открытый параметр-массив описывается по следующему шаблону:

```
var <имя_параметра>: array of
    <тип_компонентов_массива>
```

Например, если описано

```
procedure proc6 (var a: array of byte);
```

то *аргументом* могут стать такие массивы:

```
a1: array[1..100] of byte;
a2: array[-10..10] of byte;
a3: array[1..2,1..3] of byte;
```

Компоненты открытого параметра-массива нумеруются, начиная с нуля - этим достигается единообразие обращения к массивам переменной длины.

Если в качестве *аргумента* поступил многомерный массив, его компоненты "вытягиваются" в одну строку: сначала все компоненты первой строки массива, затем - второй строки и т.д. Например, если массив `a3` имеет значения

```
1 2 3
4 5 6
```

то внутри *процедуры* proc6 *параметр* a будет иметь уже следующий вид: `1 2 3 4 5 6` причем компоненты двух массивов будут соотноситься так:

a[0]	**a[1]**	**a[2]**	**a[3]**	**a[4]**	**a[5]**
a3[1,1]	a3[1,2]	a3[1,3]	a3[2,1]	a3[2,2]	a3[2,3]

Открытые строки

Поскольку строки - это массивы символов, то они тоже могут стать *открытыми параметрами*. Описывается это следующим образом:

var <имя_параметра>: string

Например:

function func6 (var s: string): byte;

Длина такого *параметра* будет автоматически скорректирована в соответствии с длиной строки-аргумента.

Процедурный тип данных

Имена *подпрограмм* могут выступать в роли *аргументов* для других *подпрограмм*.

Описание

В разделе `type` *процедурный тип данных* задается одним из следующих способов:

<имя_типа> = function[(<список_параметров>)]:<тип_результата>; или

<имя_типа> = procedure[(<список_параметров>)];

Например:

```
type func = function(a,b:integer):integer;
```

Аргументы

Аргументами, которые можно передать в *параметр процедурного типа*, могут быть только *подпрограммы* первого уровня вложенности, чье объявление полностью соответствует этому типу. Кроме того, объявления *подпрограмм*, которые могут стать *аргументами*, необходимо снабдить ключевым словом **far**, означающим, что программа будет использовать не только основной сегмент данных.

Например, для *параметра*, имеющего описанный выше тип **func**, *аргументами* могут послужить такие *функции*:

```
function min(a,b: integer): integer; far;
Begin
if a>b  then
min:= b
else
min:= a
end;
```

```
function max(a,b: integer): integer; far;
begin
if a<b    then
max:= b
else
max:= a
end;
```

Вызов

Пример *подпрограммы*, имеющей *параметр процедурного типа*:

```
procedure count(i,j:integer; f:func);
var c: integer;
Begin
...
c:= f(i,j);
 ...
end;
```

Если будет осуществлен *вызов* `count(x,y,min)`, то в *локальную переменную* с запишется минимум из `x` и `y`. Если же вызвана будет `count(x,y,max)`, то в *локальную переменную* с запишется максимум из `x` и `y`.