6. Линейные и разветвляющиеся алгоритмы

6.1. Блок операторов

И Паскаль, и Си поддерживают **концепцию блока** (блок - группа операторов и локальных переменных, которая может быть помещена в любое место программы и выполнена как один оператор).

Паскаль	Си

Заголовок блока:
program <имя>;
или procedure <имя>;
или function <имя>;
Var
описания локальных переменных
(для данной подпрограммы)
begin
оператор1;
оператор2;
end;
	{
	описания локальных (для блока) переменных
	
	оператор1;
	оператор2;
	}

Эти две формы похожи, но имеют **важные отличия**:

1) в Паскале можно не ставить точку с запятой за последним оператором (**перед end**), а в Си надо обязательно;

2) в Си не надо ставить точку с запятой **после фигурной скобки** (}) в конце блока; а в Паскале надо ставить после end точку с запятой;

1 3) в Паскале между begin и end нельзя размещать **описания переменных**, а в Си - можно. Эти переменные размещаются временно (до выхода из блока) в стеке (и тем самым позволяют экономить память в программе).

4) в Паскале допускаются только именованные блоки (функции, процедуры), а в Си - и именованные и неименованные.

6.2. Средства записи разветвляющихся алгоритмов

Как Паскаль, так и Си поддерживают две конструкции выполнения по условию: оператор **if** и оператор **case** (**switch** в языке Си).

Оператор if очень похож в обоих языках:

Паскаль	Си
if <логическое выражение> then <оператор1> else <оператор2>	if (<выражение>) <оператор1>; else <оператор2>;

И в Паскале, и в Си конструкция else (иначе) – необязательная часть, а <оператор1> и <оператор2> могут быть блоками. Однако имеется несколько важных **различий**:

1) в Си <выражение> после слова *if* **не должно быть булевским**, оно может иметь любое нулевое или ненулевое значение, причем ноль рассматривается как **false (ложь)**, а не ноль - как **true (истина)**;

2) в Си <выражение> после слова *if* должно быть заключено в круглые скобки;

3) в Си **отсутствует** слово **then**;

4) в Си после <оператор1> **перед else** обязательно **требуется точка с запятой**, естественно, за исключением случая, когда там (перед else) стоит блок.

Оператор множественного выбора также реализован в обоих языках (в Си он назван switch):

Паскаль	Си
case <выражение> of 1) <список1> :<оператор1>; <список2> :<оператор2>; ... <списокN-1>: <оператор N-1>; else <операторы N> end; 3)	switch (<выражение>) { 2) 5) 4) case <конст.1> : <операторы1> break; case <конст.2> : <операторы2> ... case <конст.N-1>: <операторыN-1> default : <операторы N> } 3)

Имеются **различия** между реализациями этих конструкций в Паскале и Си:

1) в Паскале <список> может быть списком значений (через запятую) или диапазоном (вида A..Z). В Си ему соответствует <конст> – **всегда одно значение** (символьного, целого или перечисляемого типа);

2) после двоеточия в Паскале мог быть **один оператор**, а в Си несколько;

3) в Паскале в конце оператора case после else двоеточие ставить не нужно, а в Си после default - нужно;

4) в Паскале, после выполнения соответствующих определенному (в соответствии с константой) варианту действий, все следующие операторы пропускаются. В Си после попадания на любой из вариантов и выполнения нужных операторов управление не обязательно передается в конец switch, а **продолжается выполнение операторов** (включая те операторы, которые располагаются после default) до тех пор, пока не будет достигнут конец оператора switch. Это позволяет понимать каждую конструкцию

case <конст>:

как метку, на которую передается управление в случае, когда <выражение> примет значение = <конст> (которая перед двоеточием).

Если ни с одной из констант значение выражения не совпадает, то выполняются только те действия, которые записаны после default;

5) в операторе switch можно (и обычно нужно) указать (с помощью оператора **break)**, что после выполнения действий, соответствующих определенному варианту, дальнейшее выполнение оператора switch надо прекратить. Для этого надо последним действием для данного варианта сделать оператор break:

```
switch (выражение)
{
    case конст1:оператор1;  оператор2; break;  //с этого места выполнение switch прервется
        .....
}//конец switch
следующий за switch оператор
```

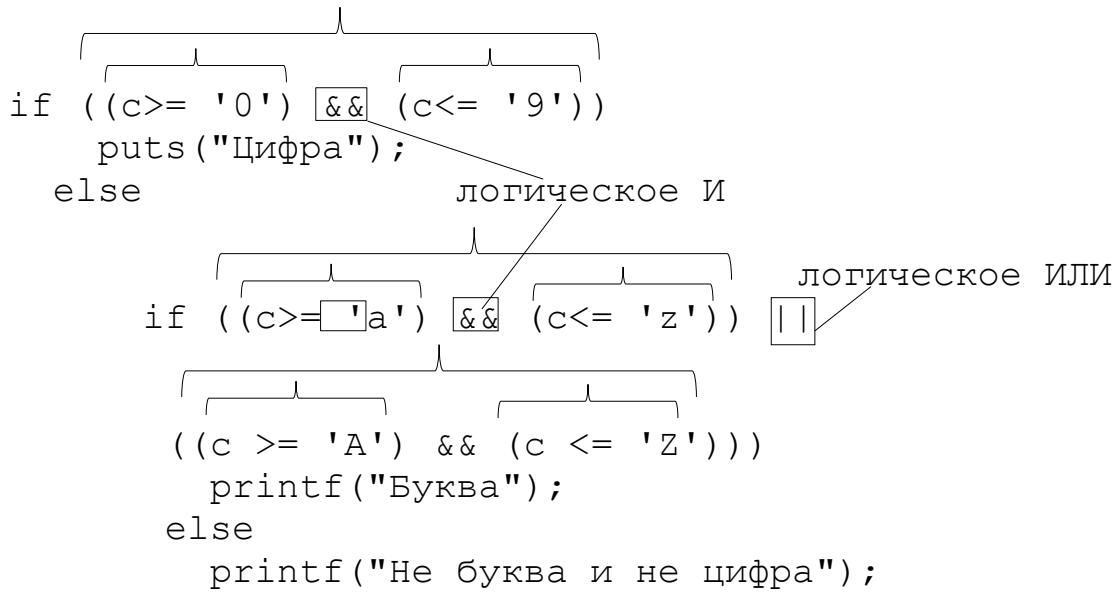
Пример: необходимо считать несколько символов с клавиатуры и каждый раз вывести информацию о считанном символе – буква это или цифра. Предполагается, что будут вводиться буквы только английского алфавита, т.к char c ≡ signed char c (диапазон от 0 до 127).

Паскаль | Си

```
uses crt;
 var
   c:char;
begin
 clrscr;
 c:=readkey; --- считываем код символа
 while c<> #27 do --- цикл, пока
                      не нажмем
                      клавишу ESC
 begin
              код клавиши Esc

  case c of --- Анализ кода клавиши
  '0'..'9': writeln('Цифра');

                нужна всего 1 строка
                на каждый диапазон символов
  'a'..'z',
  'A'..'Z' :

          writeln('Буква');
   else    writeln("Не буква и не цифра");
  end;
  c:=readkey;--- считываем код следующей клавиши
 end;
end.
```

```
#include <stdio.h>
#include <process.h> или <stdlib.h>
 char c;
 void main(void)
 {
    system("cls");
   while ((c = _getch()) != 27)
            -------------
 {
          на Си в том месте, где может
       располагаться переменная, можно
располагать оператор присваивания ей значения
    switch (c)
    { case '0':                нужно 10 строк
      case '1':                на этот диапазон
      ....
      case '9': puts("Цифра"); break;
      case 'a':                нужно 26 строк
       .....                   на этот диапазон
      case 'z':
      case 'A':
       ....
      case 'Z':
      puts("Буква"); break;
      default : puts("Не буква и не цифра");
     } // конец switch

 }  // конец тела цикла while
}    // конец main()
```

Этот пример вроде бы показывает, что из-за отсутствия в Си типа-диапазона привычные в Паскале действия в Си **с помощью switch лучше не делать** (получается хуже, чем в Паскале). Но ту же самую задачу на Си **лучше решить, используя вместо оператора switch обычный оператор if**:



Может показаться, что на Паскале опять было бы лучше (с использованием множеств):

```
if c in ['a'..'z']
  then writeln('буква')
  else
   if c in['0'..'9']
      then writeln('Цифра')
```

Но такие же проверки в языке Си намного лучше выполняются функциями (макро) **классификации символов**:

```
isalpha()
isdigit()  } их определения в ctype.h
. . . . .
```

```
                    Возвращает не 0, если символ c есть буква
if (isalpha(c))                                      (английская)
  puts("буква");
                   Возвращает не 0, если символ c  есть цифра
  else                                               (арабская).
    if (isdigit(c))
     puts("цифра");
```

7. Циклы

7.1. Цикл while

В Си, так же как в Паскале, есть 3 типа циклов: while, do...while и for, которые аналогичны трем конструкциям Паскаля - соответственно while, repeat...until, for. **Особенностью всех циклов в Си** является то, что во всех циклах анализируемое условие является **условием продолжения** и цикл продолжается до тех пор, пока это условие (выражение) является (остается) истинным (истинным в смысле Си, т.е. его арифметическое значение отлично от нуля).

Этот цикл **с предусловием** наиболее похоже реализован в обоих языках:

while <булево выражение >do < оператор>; 3) 1) пока выражение истинно (равно TRUE), цикл повторяется	while (<выражение >) do <оператор>; 2) скобки обязательны 3) 1) пока выражение истинно (>0), цикл повторяется

7.2. Цикл do...while

Цикл **с постусловием** do...while аналогичен циклу repeat...until в Паскале:

repeat <операторы> 4) until<булево выражение>; 1)	do <оператор >; или <блок операторов> 2) while (<выражение >); 1) 2)

Но есть и важные **различия между циклами в Паскале и Си**:

1) цикл do...while выполняется, пока <выражение> истинно, а repeat...until заканчивается, когда <булево выражение> станет истинно (в Паскале это условие завершения);

2) в языке Си выражение после слова while заключается в круглые скобки;

3) на Паскале для цикла while пишется while .....do , а в Си - просто while;

4) оператор repeat...until не требует создания (использования) блока для нескольких операторов в теле цикла, а do...while требует (в фигурных скобках);

Пример: пусть надо в цикле заставить пользователя вводить целое значение до тех пор, пока оно не попадёт в заданный диапазон [B,C]

Паскаль	Си

Const
B = 10;
C = 25;
Var
A:integer;
begin
repeat
 	
write('Введите значение A: ');
readln(A)
условие завершения
 	
until (A>=B) and (A<=C);
end.
	#define B 10
	#define C 25
	#include <stdio.h>
	int a;
	void main(void)
	{
	do
	{
	printf("Введите значение A: ");
	scanf("%d", &a);
	printf("\n"); условие продолжения
	}
	while ((a<B) || (a>C));
	}

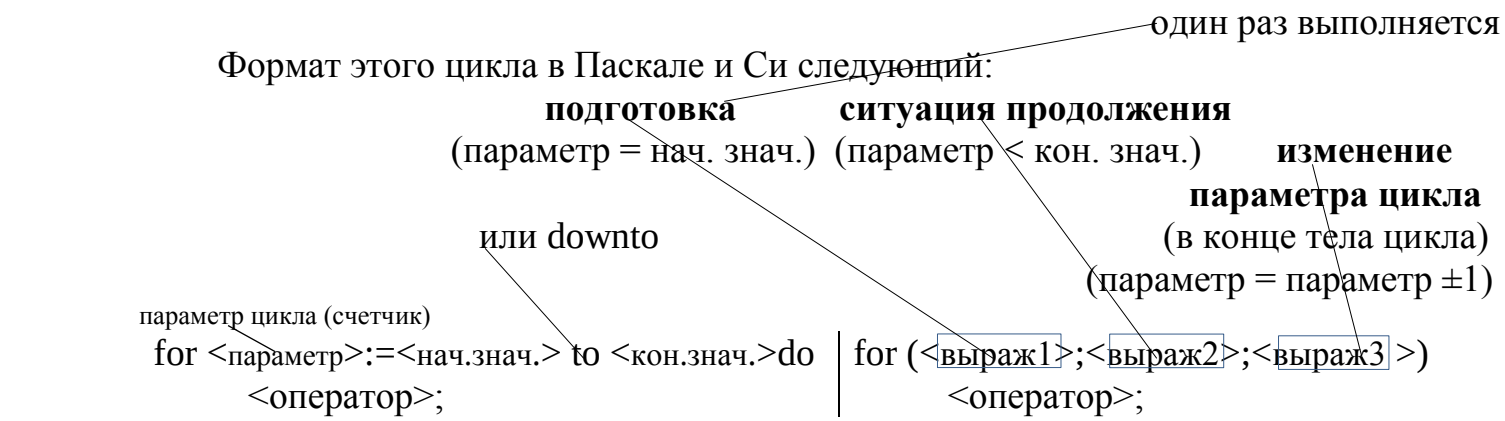
Как и в Паскале, в Си в циклах могут применяться конструкции **break** и **continue. continue** используется для пропуска оставшейся части выполняемого в данный момент тела цикла и перехода к следующему выполнению (повторению) тела цикла. **break** используется для выхода из текущего цикла (внутреннего) в цикл (внешний), его охватывающий.

```
          while (...)        do
           {                  {
             ...                ....
      +---- continue;    +---- continue;
      |      ...         |      ...
      +--->}             +--->}
                                while (...);
          for (...)
            {
             ...
             while (...)
               {
                 ...
                 break;---+
                 ...      |Из внутреннего цикла while вышли,
               }//конец while |но остались в цикле for (внешнем)
             ... <--------+
            }//конец for
```


7.3. Цикл for

Оператор цикла for в Си более гибкий, разрешающий программистские приемы (трюки), с которыми он теряет всякое сходство с циклом for из Паскаля.

Формат этого цикла в Паскале и Си следующий:



В Си оператор for - это просто специальный случай оператора while (он тоже как и цикл FOR - цикл с предусловием):

```
<выраж1>
while (<выраж2>)
  {
    <оператор>;
    <выраж3>;
  }
```

отсюда виден порядок выполнения оператора цикла for

где <выраж1> выполняется только один раз и используется для подготовки цикла (например, для инициализации параметра цикла);

<выраж2> используется для определения необходимости продолжения (наступления конца) цикла (выполняется перед очередным повторением тела цикла);

<выраж3> используется для изменения параметра(ов) цикла (выполняется в конце текущего выполнения тела цикла).

Особенности записи цикла for в Си:

1) обязательно **наличие скобок** (после for);

2) хотя **выраж1,выраж2 и выраж3 могут быть пропущены** по отдельности или все вместе, однако соответствующие им **точки с запятой должны оставаться**. В пределе заголовок оператора for может принять следующий вид:

for (; ;).

В общем случае:

- если пропущено выраж1, то выполнение цикла начнется с текущего значения параметра цикла;

- если выраж2 пропущено, то получится бесконечный цикл;

- если же пропущено выраж3, то цикл будет выполняться при неизменном параметре цикла;

3) **шаг изменения параметра цикла** может быть любым (целым).

Пример: посимвольный вывод строки.

```
#include  <stdio.h>
char  s[81] = “Строка”;
int  i;
void  main(void)
  {
   for (  i = 0; s[i] != ‘\0’ ;  i++)
   putchar(s[i]); ≡ _putch(s[i]);
  }
```

≡ s[i]

≡

```
void  main(void)
  {
     for  (int i = 0; s[i]; _putch(s[i++);
  }
```

Рассмотрим несколько **примеров использования циклов** в Паскале и Си:

Паскаль	Си

var
  i, j, k : integer;
  x,d : real;
begin
 for i := 1 to 10 do
  begin
  write('i = ', i:2);
  write(' i*i = ', (i*i):4);
  writeln(' i**3 = ', (i*i*i):6)
  end;
	int так лучше чем float x ,y;
	  i, j, k; (у м.б. int по умолчанию)
	float x, float d;
	void main (void) {
	  for (i = 1; i<=10; i++) то же самое
	  { вычисляет
	  printf("i = %2d ",i); функция powf( )
	  printf("i*i = %4d ",i*i);
	  printf("i**3=%6d\n", i*i*i);
	  }
	  powf (i, 3) ≡ i^3

была подготовка цикла,
а стала выраж1
i := 17; k := i;
 while (i>-450) do
  begin
  k := k + i;
  writeln('k=',k,' i=',i);
  i := i – 15;
  end;
	for (i=17, k=i; i >-450; k+=i , i-=15)
	
	printf("k=%d i=%d\n",k,i)

X := D/2.0; while(Abs(X*X-D)>0.01) do X := (X + D/X)/2.0; end.	for(x=d/2; fabs(x*x-d)>0.01; x = (x+d/x)/2) ; /* тело цикла - пустой оператор*/ } /* конец main ( ) */