Информационные технологии

Тема 1. Системы счисления

Системой счисления называется совокупность правил для обозначения (записи) действительных чисел с помощью цифровых знаков. Для записи чисел в конкретных системах счисления используется некоторый конечный алфавит, состоящий из цифр a_1, a_2, a_3,….,a_n. При этом каждой цифре a_i в записи числа ставится в соответствие определенный количественный эквивалент. Различают непозиционные и позиционные системы счисления.

Позиционные и непозиционные системы счисления

- *Непозиционные системы счисления.* В ней количественный эквивалент каждой цифры, входящей в запись данного числа, не зависит от места (позиции) этой цифры в ряду других цифр. Пример: римская система счисления. В ней для записи различных целых чисел используются символы I, V, X, L, C, D, M и т.д., обозначающие соответственно 1, 5, 10, 50, 100, 500, 1000 и т.д. Например, запись MCMLXXXV означает число 1985. Общим недостатком непозиционных систем является сложность представления в них достаточно больших чисел, так как при этом получается чрезвычайно громоздкая запись чисел или требуется очень большой алфавит используемых цифр. В ЭВМ применяют только позиционные системы счисления, в которых количественный эквивалент каждой цифры алфавита зависит не только от вида этой цифры, но и от ее местоположения в записи числа.
- *Позиционные системы счисления.* В позиционной системе счисления запись $a_n a_{n-1}….a_1\ a_0,\ a_{-1}\ ….\ a_{-k}\ …$, где a_i – цифра из конечного алфавита, количество элементов которого является основанием q системы счисления, представляет собой сокращенную запись числа A:

  $A=a_nq^n + a_{n-1}q^{n-1} + … + a_1q^1 + a_0q^0 + a_{-1}q^{-1} +…+ a_{-k}q^{-k} +…$ (1)

  В позиционных системах счисления вес каждой цифры изменяется в зависимости от ее позиции в последовательности цифр, изображающих число. Любая позиционная система характеризуется своим основанием. Основание позиционной системы счисления - это количество различных знаков или символов, используемых для изображения цифр в данной системе. За основание можно принять любое натуральное число - два, три, четыре, десять, шестнадцать и т.д. Следовательно, возможно бесконечное множество позиционных систем. Однако наибольшую ценность для нас имеет индо-арабская (десятичная) система, где имеется ограниченное число значащих цифр - всего 9, а также символ 0 (нуль)

  В целом числе предполагается, что точка (запятая) находится справа от правой крайней цифры. Возможные нули в правых левых и крайних позициях числа не влияют на величину числа и поэтому не отображаются. Действительно, число *432.15* равно числу *000423.150*. Такие нули называются незначащими. Крайняя левая цифра в числе называется цифрой старшего разряда, а крайняя правая - цифрой младшего разряда.

- *Десятичная система счисления.* Пришла в Европу из Индии, где она появилась не позднее VI века н.э. В этой системе 10 цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, но информацию несет не только цифра, но и место, на котором цифра стоит (то есть ее позиция). В десятичной системе счисления особую роль играют число 10 и его степени: 10, 100, 1000 и т.д. Самая правая цифра числа показывает число единиц, вторая справа - число десятков, следующая - число сотен и т.д. Позиции цифр в записи числа называют его разрядами. В десятичной системе счисления вес каждого разряда в 10 раз больше веса предыдущего. Всякое число в десятичной системе счисления можно представить в виде суммы различных целых степеней десятки с соответствующими коэффициентами a_i (0-9), взятыми из алфавита данной системы счисления.

  Пример

  $245{,}83 = 2 * 10^2 + 4 * 10^1 + 5 * 10^0 + 8 * 10^{-1} + 3 * 10^{-2}$.
- *Двоичная система счисления.* Столь привычная для нас десятичная система оказалась неудобной для ЭВМ . Если в механических вычислительных устройствах, использующих десятичную систему, достаточно просто применить элемент со множеством состояний (колесо с девятью зубьями), то в электронных машинах надо было бы иметь 10 различных потенциалов в цепях . Наиболее просто реализуется элементы с двумя состояниями. Поэтому естественным был переход на двоичную систему, т.е. системы по основанию q=2.

  В этой системе всего две цифры - 0 и 1. Каждая цифра называется двоичной (от английского binary digit - двоичная цифра). Сокращение от этого выражения (`binary digi`t, bit) привело к появлению термина бит, ставшего названием разряда двоичного числа . Веса разрядов в двоичной системе изменяется по степеням двойки . Поскольку вес каждого разряда умножается либо на 1, либо на 0, то в результате значение числа определяется как сумма соответствующих значений степеней двойки . Ниже в таблице показаны значения весов для 8-разрядного числа (1 байт).

  Таблица 1 – Таблица весов для числа размером 1 байт

Номер разряда	7	6	5	4	3	2	1	0
Степень двойки	7	6	5	4	3	2	1	0
Значение позиции	128	64	32	16	8	4	2	1

  Нетрудно догадаться, что максимальное значение двоичного числа ограничено числом его разрядов и определяется по формуле $M=2^n-1$, где n-число разрядов.

  В вычислительной технике эти числа имеют фиксированные значения 4, 8,16, 32, а соответствующие им числа будут иметь максимальные значения, представленные в таблице 2.

  Таблица 2 – Соотношение числа разрядов с максимальным значением числа

Число разрядов	Максимальное значение числа
4	15(полубайт)

8	255(байт)
16	65535(слово)
32	4294967295 (двойное слово)

- *Восьмиричная и шестнадцатиричная системы счисления.* В восьмеричной системе счисления 8 цифр: 0, 1, 2, 3, 4, 5, 6, 7. Цифра 1, указанная в самом младшем разряде, означает - как и в десятичном числе - просто единицу. Та же цифра 1 в следующем разряде означает 8, в следующем 64 и т.д. Число 100 (восьмеричное) есть не что иное, как 64 (десятичное).

  Запись числа в восьмеричной системе счисления достаточно компактна, но еще компактнее она получается в шестнадцатеричной системе. В качестве первых 10 из 16 шестнадцатеричных цифр взяты привычные цифры 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, а вот в качестве остальных 6 цифр используют первые буквы латинского алфавита: A, B, C, D, E, F. Цифра 1, записанная в самом младшем разряде, означат просто единицу. Та же цифра 1 в следующем - 16 (десятичное), в следующем - 256 (десятичное) и т.д. Цифра F, указанная в самом младшем разряде, означает 15 (десятичное).

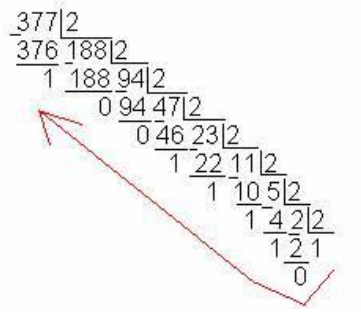
  Вообще, системы бывают любой разрядности (не только двоичные, десятичные, восьмеричные и шестнадцатеричные). Например, двенадцатеричная, на которой раньше производились все расчеты, и как память об этом мы живем по календарю, в котором 12 месяцев. Троичные, пятеричные и т.п. также возможны. Троичная система, например, используется при адресации троичных деревьев. Операции перевода из одной системы счисления в другую универсальны для всех систем счисления.

Перевод из одной системы счисления в другую

Перевод целых чисел из десятичной системы счисления в другую

Для перевода целых чисел из десятичной системы счисления в другую с основанием *q,* надо это число последовательно делить на основание *q* новой системы счисления до тех пор, пока не получится частное меньше *q*. Число в новой системе запишется в виде остатков деления, начиная с последнего. Это последнее частое дает цифру старшего разряда в новой системе счисления. Деление выполняют в исходной системе счисления.

Пример



$377_{10}=101111001_2$

Перевод правильных дробей из десятичной системы счисления в другую

Для перевода правильной дроби из десятичной системы счисления в другую необходимо эту дробь последовательно умножать на основание той системы, в которую она переводится, перемножаются только дробные части. Дробь в новой системе записывается в виде целых частей получающихся произведений, начиная с первого.

Пример перевода числа 0,6875

0,6875
* 2
1,3750
* 2
0,7500
* 2
1,5000
* 2
1,0000.

Получается, $0{,}6875_{10}=0{,}1011_2$

Количество умножений на основание систем зависит от требуемой точности представления числа. Если требуется перевести с точностью до N знаков после запятой ,

Надо получить N+1 цифр дроби, после чего округлить до N знаков.

При переводе неправильных десятичных дробей необходимо пользуясь рассмотренными правилами выполнить отдельно перевод целой и дробной частей.

Правила перевода из любой системы счисления в десятичную систему счисления

- 1 способ

По формуле (1).

Пример:

$10110{,}11_2 = 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0 + 1 * 2^{-1} + 1 * 2^{-2} = 22{,}75_{10}$

- 2 способ

a) Старшую цифру исходного числа умножить на основание старой системы счисления и прибавить следующую цифру исходного числа
b) Результат опять умножить на основание старой системы счисления и прибавить следующую цифру исходного числа
c) Процесс перевода заканчивается после прибавления последней самой младшей цифры исходного числа

Пример:

$10110_2 = 22_{10}$,так как по указанным выше действиям получаем:

1*2+0=2
2*2+1=5
5*2+1=11
11*2+0=22

Примечание: этот способ годится только для перевода целых чисел

Перевод чисел из любой системы счисления в любую

Для перевода чисел из любой системы счисления в любую необходимо исходное число перевести в десятичную систему, полученное десятичное число перевести в искомую систему.

Перевод чисел из систем счисления, которые являются степенью двойки

Для перевода чисел из систем счисления, которые являются степенью двойки необходимо:

А) из 16-ричной в 2-ичную: для перевода 16-ричного числа в двоичную систему необходимо каждую цифру 16-ричного числа заменить 4-х разрядным двоичным значением.

Б) из 8-ричной в 2-ичную: Каждую цифру 8-ричного числа необходимо заменить 3-х разрядным двоичным значением.

Пример:

A	B	5	1_{16}
1010	1011	0101	0001_2

1	7	7	2	0	4_8
1	111	111	010	000	100_2

При обратном переводе двоичное число , начиная с запятой разбивают на группы из 3 или 4 бит (при необходимости слева дописывают нули) и каждую группу заменяют на соответствующее число.

Пример 1:

1100	0011	1101	0110_2	$= C3D6_{16}$
C	3	D	6	

1	100	011	111	010	110_2	$= 141726_8$
1	4	1	7	2	6	

Пример 2: Перевести $11011011{,}11011011_2$ перевести в 8ую и 16ую системы счисления.

Перевод в 8ю систему счисления:

011	011	011,	110	110	110	$=333{,}666_8$
3	3	3,	6	6	6	

Перевод в 16ю систему счисления:

1101	1011,	1101	1011	$= DB{,}DB_{16}$
D	B,	D	B	

Представление чисел в различных системах счисления

Таблица 3 – Представление чисел от 0 до 15 в разных системах счисления

Системы счислений			
Десятичная	Двоичная	Восьмеричная	Шестнадцатеричная
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6

7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Арифметические действия

Арифметические действия, выполняемые в двоичной системе, подчиняются тем же основным правилам, что и в десятичной системе. Только в двоичной системе перенос единиц в старший разряд происходит несравнимо чаще. Вот как выглядит сложение в двоичной системе:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 + 1 - перенос

или 11010
+ 10010

101100

При выполнении математических действий результат может получиться не только положительным, но и отрицательным. Как же представить знак минус в схемах машины, если в них фиксируется лишь два состояния 1 и 0? Договорились знак числа определять самым левым битом. Если число положительное, то этот бит (знаковый) равен 0 (сброшен), если отрицательное - 1 (установлен). Решение о введении знакового разряда сказалось на максимальных величинах представляемых чисел. Максимальное положительное 16-битное число равно +32767, а отрицательное -32768.

Положительные числа в ЭВМ хранятся в прямом коде, а отрицательные - в дополнительном. Чтобы получить дополнительный код из прямого следует инвертировать все разряды числа (обратный код) и прибавить единицу к младшему разряду. Для получения прямого кода из дополнительного надо произвести обратные действия: отнять единицу и инвертировать разряды или инвертировать разряды и прибавить единицу.

Для положительных чисел дополнительный, обратный и прямой коды совпадают.

Таблица 4 – Представление прямого, обратного и дополнительного кода для чисел от -8 до 7

Десятичное число	Прямой код	Обратный код	Дополнительный код
-8	-	-	1000
-7	1111	1000	1001
-6	1110	1001	1010
-5	1101	1010	1011

-4	1100	1011	1100
-3	1011	1100	1101
-2	1010	1101	1110
-1	1001	1110	1111
0	1000	1111	0000
	0000	0000	
1	0001	0001	0001
2	0010	0010	0010
3	0011	0011	0011
4	0100	0100	0100
5	0101	0101	0101
6	0110	0110	0110
7	0111	0111	0111

В таблице 4 приведены десятичные числа и их двоичные представления в трех различных формах. Интересно в ней вот что. Если начать счет с числа 1000 (-8) и двигаться вниз по столбцам, то в дополнительном коде каждое последующее число получается прибавлением единицы к предыдущему без учета переноса за пределы четвертого разряда. Так просто эту операцию в прямом и обратном кодах не осуществить. Эта особенность дополнительного кода и явилось причиной предпочтительного применения его в современных микро- и мини- ЭВМ.

Итак, числа, представленные в дополнительном коде, складываются по правилам двоичного сложения, но без учета каких либо переносов за пределы старшего разряда. Затем полученное число, если оно отрицательное, опять необходимо представить в прямой код (инвертировать и прибавить 1).

Рассмотрим это на следующих примерах:

```
 +2    0010              -2    1110
+     +                 +     +
 +5    0101              -6    1010
 ----   -----            ---   --------
 +7    0111              -8    1000   = 0111(обр. код) = 1000₂(прям. код)=8₁₀

 +5    0101              +3    0011
+     +                 +     +
 -4    1100              -7    1001
 ---   ------            ---   ------
 +1    0001              -4    1100   = 0011(обр. код) = 0100₂(прям. код)= 4₁₀
```

Умножение двоичных чисел происходит еще проще, чем сложение. Ведь она обладает рекордно малой таблицей умножения.

Множимое		Множитель		Произведение
0	*	0	=	0
0	*	1	=	0
1	*	0	=	0
1	*	1	=	1

Другими словами, процедура умножения сводится к записи 0, если разряд множителя равен 0, или 1, если разряд =1.

Пример: $12_{10} = 1100_2$ умножить на $6_{10} = 110_2$

```
   1100
  *
    110
   0000
+ 1100
  1100
```

$1001000_2 = 72_{10} = 12_{10} * 6_{10}$

Двоичное деление сводится к выполнению операций умножения и вычитания, как в десятичной системе. Выполнение этой процедуры - выбор числа, кратного делителю, и предназначенному для уменьшения делимого, здесь проще, так как таким числом может быть либо 0, либо сам делитель.

Для деления чисел со знаком в дополнительном коде существует несколько методов. Простейший из них – преобразование чисел в положительные с последующим восстановлением знака результата.

Пример 1: $12_{10} = 1100_2$ разделить на $4_{10} = 100_2$

```
_1100 |100
 100   11₂ = 3₁₀
  100
  100
    0
```

Пример 2: $12_{10} = 1100_2$ разделить на $6_{10} = 110_2$

```
_1100 |110
 110   10₂ = 2₁₀
   0   (деление закончили, так как получили ноль и оставшийся ноль дописали в конец)
```

В ЭВМ вычитание производится как сложение с числом в дополнительном коде. Все положительные числа хранятся в прямом коде, имеют «0» в знаковом разряде. Число в дополнительном коде получают из числа в прямом коде путем инвертирования и сложением с единицей в младшем разряде.

Например:

```
00000101    Прямой код числа 5
11111010    Обратный код числа 5
+       1
11111011    Дополнительный код числа 5 (-5)
```

Приведем примеры.

Пример 1: X = 7 - 5

```
    00000111        Прямой код 7
+   11111010        Дополнительный код 5 (-5)
    --------
    00000010        X = 2
```

Пример 2: X = 3 – 5

```
    00000011        Прямой код 3
+   11111011        Дополнительный код 5 (-5)
    --------
    11111110        Дополнительный код X, т.к. в знаковом разряде стоит
```

единица. Чтобы определить, какова абсолютная величина числа X, требуется найти его прямой код. Перевод осуществляется по тому же правилу – инверсия и инкремент (добавление единицы к младшему разряду).

	11111110	Дополнительный код X
+	00000001	Инверсия
	00000001	Инкремент
	00000010	abs (X) = 2, значит, X = -2

Контрольная работа по системам счисления

Вариант 1.

1. Перевести в десятичную систему счисления из восьмеричной системы 345,67
2. Перевести в шестнадцатеричную систему счисления из восьмеричной системы 345,67
3. Перевести в двоичную систему счисления из десятичной системы с точностью до двух знаков после запятой 5,67
4. Произвести вычитание в разрядной сетке ЭВМ 88-128

Вариант 2.

1. Перевести в десятичную систему счисления из шестнадцатеричной системы 1EFA,78B
2. Перевести в восьмеричную систему счисления из шестнадцатеричной системы 1EFA,78B
3. Перевести в двоичную систему счисления из десятичной системы с точностью до двух знаков после запятой 3,78
4. Произвести вычитание в разрядной сетке ЭВМ 29-129

Вариант 3.

1. Перевести в десятичную систему счисления из восьмеричной системы 347,652
2. Перевести в шестнадцатеричную систему счисления из восьмеричной системы 347,652
3. Перевести в двоичную систему счисления из десятичной системы с точностью до двух знаков после запятой 4,45
4. Произвести вычитание в разрядной сетке ЭВМ 67-132

Вариант 4.

1. Перевести в десятичную систему счисления из шестнадцатеричной системы 235,FCD
2. Перевести в восьмеричную систему счисления из шестнадцатеричной системы 235,FCD
3. Перевести в двоичную систему счисления из десятичной системы с точностью до двух знаков после запятой 6,66
4. Произвести вычитание в разрядной сетке ЭВМ 125-155

Вариант 5.

1. Перевести в десятичную систему счисления из восьмеричной системы 567,765
2. Перевести в шестнадцатеричную систему счисления из восьмеричной системы 456,654
3. Перевести в двоичную систему счисления из десятичной системы с точностью до двух знаков после запятой 7,54
4. Произвести вычитание в разрядной сетке ЭВМ 67-69

Вариант 6.

1. Перевести в десятичную систему счисления из шестнадцатеричной системы 12AC,D8
2. Перевести в восьмеричную систему счисления из шестнадцатеричной системы 12AC,D8
3. Перевести в двоичную систему счисления из десятичной системы с точностью до двух знаков после запятой 8,72
4. Произвести вычитание в разрядной сетке ЭВМ 75-95

Вариант 7.

1. Перевести в десятичную систему счисления из восьмеричной системы 548,765
2. Перевести в шестнадцатеричную систему счисления из восьмеричной системы 548,765
3. Перевести в двоичную систему счисления из десятичной системы с точностью до двух знаков после запятой 9,61
4. Произвести вычитание в разрядной сетке ЭВМ 90-129

Вариант 8.

1. Перевести в десятичную систему счисления из шестнадцатеричной системы 3A4B,C5
2. Перевести в восьмеричную систему счисления из шестнадцатеричной системы 3A4B,C5
3. Перевести в двоичную систему счисления из десятичной системы с точностью до двух знаков после запятой 2,81
4. Произвести вычитание в разрядной сетке ЭВМ 145-45

Вариант 9.

1. Перевести в десятичную систему счисления из восьмеричной системы 176,543
2. Перевести в шестнадцатеричную систему счисления из восьмеричной системы 176,543
3. Перевести в двоичную систему счисления из десятичной системы с точностью до двух знаков после запятой 1,43
4. Произвести вычитание в разрядной сетке ЭВМ 90-160

Вариант 10.

1. Перевести в десятичную систему счисления из шестнадцатеричной системы 3BE,CA8
2. Перевести в восьмеричную систему счисления из шестнадцатеричной системы 3BE,CA8
3. Перевести в двоичную систему счисления из десятичной системы с точностью до двух знаков после запятой 1,74
4. Произвести вычитание в разрядной сетке ЭВМ 25-125

Вариант 11.

1. Перевести в десятичную систему счисления из восьмеричной системы 6347,1234
2. Перевести в шестнадцатеричную систему счисления из восьмеричной системы 6347,1234
3. Перевести в двоичную систему счисления из десятичной системы с точностью до двух знаков после запятой 5,91
4. Произвести вычитание в разрядной сетке ЭВМ 125-25

Вариант 12.

1. Перевести в десятичную систему счисления из шестнадцатеричной системы 23A5,C67
2. Перевести в восьмеричную систему счисления из шестнадцатеричной системы 23A5,C67
3. Перевести в двоичную систему счисления из десятичной системы с точностью до двух знаков после запятой 1,63
4. Произвести вычитание в разрядной сетке ЭВМ 12-130

Вариант 13.

1. Перевести в десятичную систему счисления из восьмеричной системы 545,4556
2. Перевести в шестнадцатеричную систему счисления из восьмеричной системы 545,4556
3. Перевести в двоичную систему счисления из десятичной системы с точностью до двух знаков после запятой 2,59
4. Произвести вычитание в разрядной сетке ЭВМ 15-98

Тема 2. Схемы алгоритмов. ГОСТ

Единая система программной документации (ЕСПД)
Государственный стандарт (ГОСТ) 19.701-90
Схемы алгоритмов, программ данных и систем
Условные обозначения и правила выполнения

Способы описания алгоритмов

1. Словесно
   Например, рецепты, правила по покупке и пр.
2. Формулой
   Например, $S = v_0*t + (a*t^2)/2$, где старшинство операций задают последовательность действий
3. На псевдоязыке
   Описание алгоритмов на псевдоязыке – это словесное описание с помощью структур, близких к языку программирования.
   Пример: поиск корней многочлена вида $ax^2 + bx + c = 0/$
   1) Ввод a, b, c
   2) Если a = 0, то печать сообщения «Нет квадратного уравнения», иначе п.3
   3) $D = b^2 – 4ac$
   4) Если D < 0, то печать сообщения «Нет вещественных корней», иначе…
   5) И т.д.
4. Графический

Графический способ описания алгоритмов

Стандарт распространяется на условные обозначения (символы) в схемах алгоритмов, программ, данных и систем и устанавливает правила выполнения схем, используемых для отображения различных видов задач обработки данных и средств их решения.

Общие положения

Схемы алгоритмов, программ, данных и систем (далее схемы) состоят из имеющих заданное значение символов, краткого пояснительного текста и соединяющих линий.

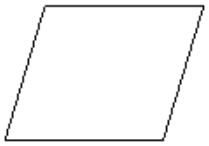
Схемы могут использоваться на различных уровнях детализации, причем число уровней зависит от размеров и сложности задачи обработки данных. Уровень детализации должен быть таким, чтобы различные части и взаимосвязь между ними были понятны в целом.

В настоящем стандарте определены символы, предназначенные для использования в документации по обработке данных, и приведено руководство по условным обозначенным для применения их в:

1) схемах данных;
2) схемах программ;
3) схемах работы системы;
4) схемах взаимодействия программ;
5) схемах ресурсов системы.

Описание символов

1. Ввод и вывод данных



2. Символы процесса

2.1. Процесс

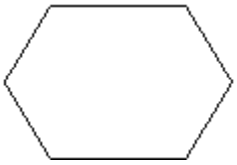
Символ отображает функцию обработки данных любого вида (выполнение определенной операции или группы операций, приводящее к изменению значения, формы или размещения информации или к определению, по которому из нескольких направлений потока следует двигаться). Направления движения задают стрелки.

2.2. Предопределенный процесс (обращение к подпрограмме)

Символ отображает предопределенный процесс, состоящий из одной или нескольких операций или шагов программы, которые определены в другом месте (в подпрограмме, модуле).

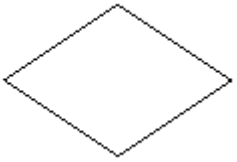
2.3. Подготовка (модификатор)

Символ отображает модификацию команды или группы команд с целью воздействия на некоторую последующую функцию (используется для реализации цикла с параметром).



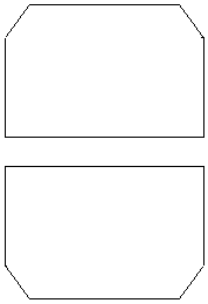
2.4. Решение

Символ отображает решение или функцию переключательного типа, имеющую один вход и ряд альтернативных выходов, один и только один из которых может быть активизирован после вычисления условий, определенных внутри этого символа.



2.5. Граница цикла

Символ, состоящий из двух частей, отображает начало и конец цикла. Обе части символа имеют один и тот же идентификатор. Условия для инициализации, приращения, завершения и т. д. помещаются внутри символа в начале или в конце в зависимости от расположения операции, проверяющей условие (т. е. в зависимости от вида цикла «до» и «пока»).



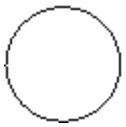
2.6. Линия

Символ отображает поток данных или управления. При необходимости и для удобочитаемости могут быть добавлены стрелки – указатели.

3. Специальные символы

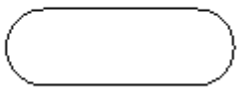
3.1. Соединитель

Символ отображает выход в часть схемы и вход из другой части этой схемы и используется для обрыва линии и продолжения ее в другом месте. Соответствующие символы – соединители должны содержать одно и то же уникальное обозначение.



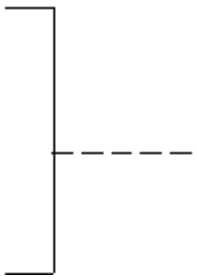
3.2. Терминатор

Символ отображает выход во внешнюю среду вход из внешней среды (начало или конец программы).



3.3. Комментарий

Символ используют для добавления описательных комментариев или пояснительных записей в целях объяснения или примечаний. Пунктирные линии в символе комментария связаны с соответствующим символом или могут обводить группу символов. Текст комментариев или примечаний должен быть помещен около ограничивающей фигуры. {В Word фигура создается из пунктирной соединяющей линии и а) черного прямоугольника с 3 белыми скрывающими ненужные линии б) 3 аккуратно соединенных линий (без пересечений)}



3.4. Пропуск

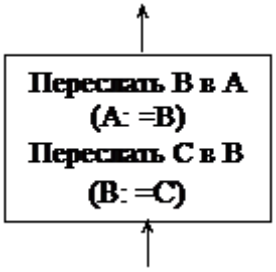
Символ (три точки) используют в схемах для отображения пропуска символа или группы символов, в которых не определены ни тип, ни число символов. Символ используют только в символах линии или между ними. Он применяется в основном в схемах, изображающих общие решения с неизвестным числом повторений.

Правила применения символов и выполнения схем

1. Правила применения символов

Большинство символов задумано так, чтобы дать возможность включения текста внутри символа. Формы символов, установленные стандартом, должны служить руководством для фактически используемых символов. Не должны изменятся углы и другие параметры, влияющие на соответствующую форму символов. Символы должны быть по возможности одного размера. При этом должно соблюдаться соотношение сторон 2:3 (2 – высота, 3 – длина), для терминаторов – 1:3, длина одинакова для всех предписаний рисунка.

Минимальное количество текста необходимое для понимания функции данного символа, следует помещать внутри данного символа. Текст для чтения должен записываться слева направо и сверху вниз, независимо от направления основного потока.



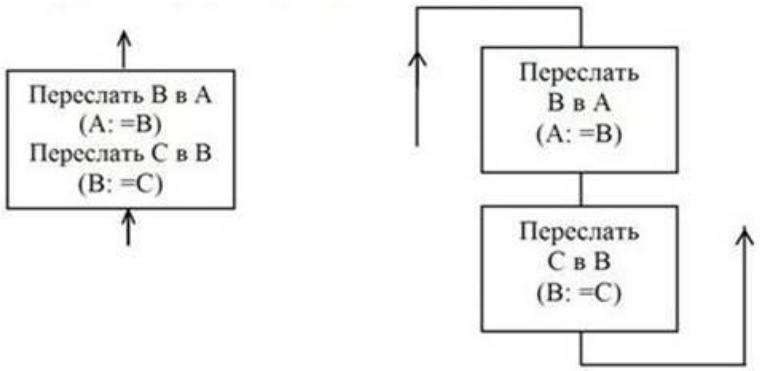
Если объем текста, помещаемого внутри символа, превышает его размеры, следует использовать символ комментария.

Если использование символов может запутать или разрушить ход схемы, текст следует помещать на отдельном листе и давать перекрестную ссылку на символ.

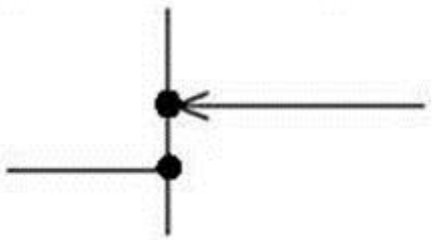
2. Правила выполнения соединений

Потоки данных или управления в схемах показывается линиями. Направление потока слева направо сверху вниз считается стандартным. В случаях, когда необходимо внести большую ясность в схему (например, при соединениях), на линиях ставятся стрелки. Если поток имеет направление отличное от стандартного, стрелки должны указывать это направление.

В схемах следует избегать пересечения линий. Пересекающиеся линии не имеют логической связи между собой, поэтому изменения направления в точках пересечения не допускаются.



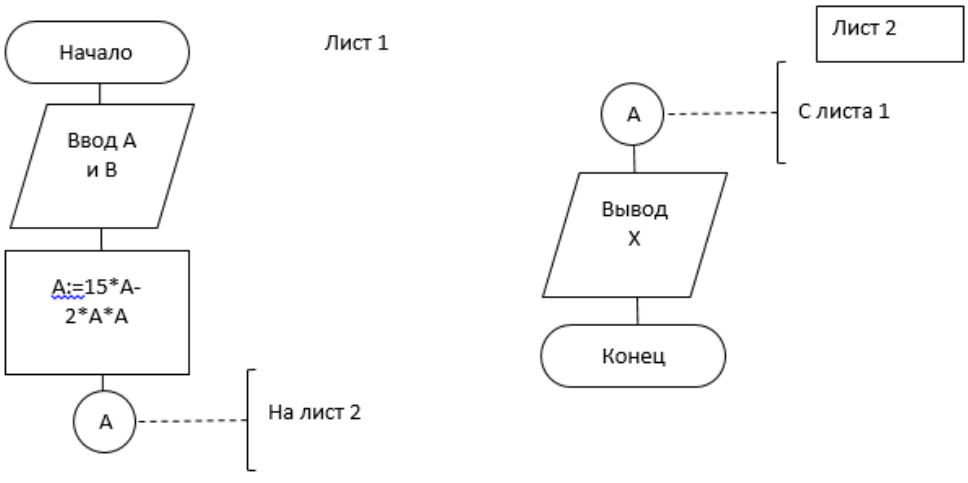
Две или более входящие линии могут объединиться в одну исходящую линию. Если две или более линии объединяются в одну линию, место объединения должно быть смещено.



Линии в схемах должны подходить к символу либо слева, либо сверху, а исходить либо справа, либо снизу. Линии должны быть направлены к центру символа.

При необходимости линии в схемах следует разрывать для избежания лишних пересечений или слишком длинных линий, а также если схема состоит из нескольких страниц. Соединитель в начале разрыва называется внешним соединителем, а соединитель в конце разрыва – внутренним соединителем.

Ссылки к страницам могут быть приведены совместно с символом комментария для их соединителей. В приведенном алгоритме решается задача нахождения X по формуле $X = (15A - 2A^2) / B^2$.

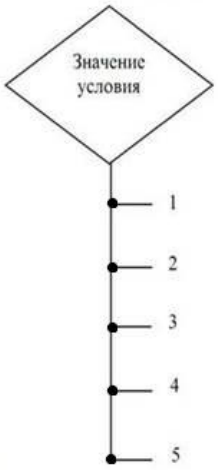


3. Несколько выходов

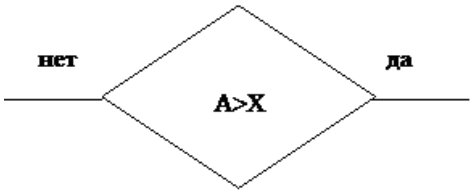
Несколько выходов из символа следует показывать:

a. несколькими линиями от данного символа к другим символам;
b. одной линией от данного символа, которая затем разветвляется в соответствующее число линий.

При многоальтернативном ветвлении каждый выход из символа должен сопровождаться соответствующим значением условия, чтобы показать логический путь, который он представляет, с тем, чтобы эти условия и соответствующие ссылки были индефицированы. Эта схема не может быть использована на структурных программах и должна заменяться цепочкой двусторонних ветвлений.



или

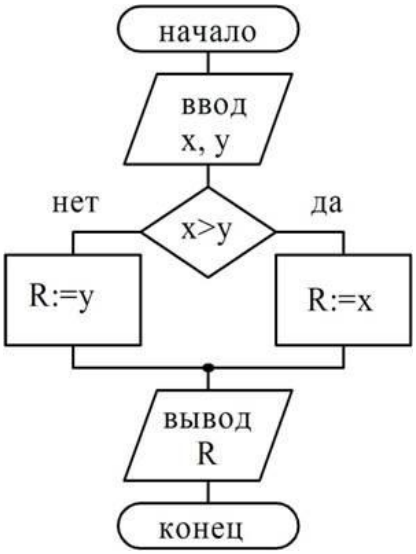


4. Повторяющееся представление

   Когда несколько символов представляют упорядоченное множество, это упорядочение должно располагаться от первого к последнему.

В общем виде алгоритм вычисления по формуле выглядит следующим образом:

Приведем пример алгоритма сравнения двух чисел и вывода на экран меньшего числа.



Контрольная работа по схемам алгоритмов

1. $X = (4A^2 – 3A + 7) / 8$

2. $X = 1 + A^2/2! + A^4/4!$

3. $X = (A^2+B^2) / C^3$

4. $X = A^3 / (2B^2 + B)$

5. $X = (8A^2 – 4A*C) / B$

6. $X = (2A^3 – 4A^2) / B^2$

7. $X = 16A^2 / (B^2 – 7A)$

8. $X = (A^2 + B^2) / (C^2 - 1)$

9. $X = 3C^3 / (A^2 + B^2)$

10. $X = (4A^2 – 2B^2 + A^2C) / C^2$

11. $X = (8A^3 + ABC) / C^2$

12. $X = A^3 / (B^2 - 1) * C$

13. $X = A^2 / 9(B^2 – 4A)$

Тема 3. Типы вычислительных процессов

Линейные процессы

Общая совокупность вычислительных процессов делится на три типа:

- линейные;
- ветвящиеся;
- циклические.

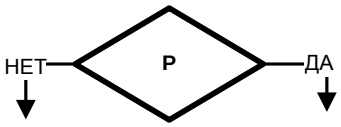
Перечисленные типы образуют так называемые *управляющие базовые структуры линейной композиции, выбора решения* и *повторения* соответственно. Доказано, что любая программа может быть написана с помощью этих трех типов канонических структур.

Линейным называется такой вычислительный процесс, в котором самостоятельные (отдельные) этапы вычислений выполняются в линейной последовательности их записи, т.е. в естественно порядке. На блок-схеме линейный вычислительный процесс представляется последовательностью блоков, выражающих автономные шаги вычислений. Блоки размещаются сверху вниз в порядке их выполнения.

Разветвляющиеся процессы

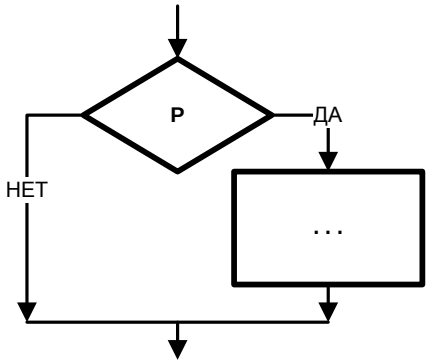
Разветвляющимся называется такой алгоритм, в котором выбирается один из нескольких возможных вариантов вычислительного процесса. Каждый подобный путь называется ветвью алгоритма.

1. *Двустороннее ветвление* – это основная базовая структура.



P – предикат (условие). Если условие выполняется, тогда выполняется оператор, стоящий в ветке «да», если нет, то идем по ветке «нет». Обычно обе ветки не подписывают, а называют только одну из них.

2. *Одностороннее ветвление*.



Одностороннее ветвление – частный случай двустороннего. В данном случае по ветке «да» выполняется предписание, а по ветке «нет» идем на выход (идем на следующий за ветвлением оператором).

Особенности обработки разветвляющихся процессов

1. Чтобы из двусторонней развилки сделать одностороннюю, надо предписание одной из веток (любой) выполнить до условного оператора – это необходимо предпринимать при реализации программ на языках низкого уровня.
2. Для того чтобы в условном операторе поменять ветки местами, условие надо поменять на противоположное.

Операции, которые используются в предикатах: *операции отношения* и логические операции.

- операции отношения (=, < >, < =, > = , <, >,)

Но чтобы осуществить пункт 2 (поменять условие на противоположное) нам потребуется применить отрицание операций отношения:

=, < >, < =, > = , <, >,

<>, = , >, < , >=, <=

- логические операции (смотрите следующую тему)

Циклические процессы

При решении многих задач вычислительный процесс имеет циклический характер. Это означает, что часть операторов многократно выполняется при различных значениях переменных. Применение циклов в программе позволяет эффективно использовать компьютер, приводит к уменьшению длины программы и сокращению времени на её составление и отладку.

В языках программирования имеется три разновидности операторов цикла:

1. Оператор с предварительным условием (предусловием);
2. Оператор цикла с последующим условием (постусловием);
3. Оператор цикла с параметром.

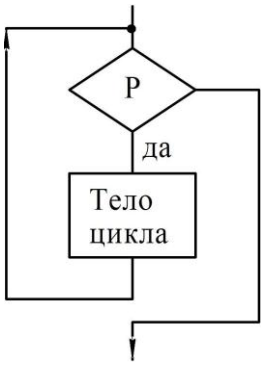
Классификация типов циклических процессов

- Вложенные циклы

Циклы могут быть вложены один в другой. Иногда их называют сложными циклами. При использовании вложенных циклов необходимо составлять программу таким образом, чтобы внутренний цикл полностью укладывался в циклическую часть внешнего цикла. Внутренний цикл может содержать в свою очередь другой внутренний цикл (циклы). На количество вложений цикла в цикл в итерационных циклах ограничений нет.

По способу реализации циклы делятся на:

Циклы с предусловием (тип «ПОКА»)



Циклы с постусловием (тип «ДО»)

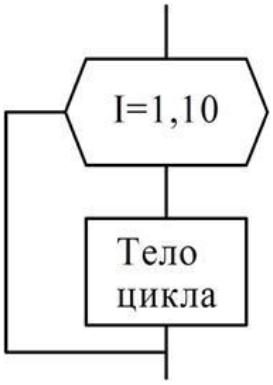
По виду выполнения циклы делятся на:

1. Арифметический (с параметром, с известным числом вхождений)
2. Итерационный (или с неизвестным числом вхождений)

Под вхождением понимается исполнение тела цикла.

На языках Си, Бейсик и Паскаль цикл с параметром реализуется по типу «ПОКА».

На языках Фортран и Ассемблер цикл с параметром реализуется по типу «ДО».



Особенности цикла с предусловием:

- Ни разу не выполняется, если условие ложно;

- Если ОПЕРАТОР составной, то он заключается в операторные скобки. Зачастую оператор должен быть составным, поскольку он включает оператор, который должен повторяться и оператор, который изменяет значение в логическом выражении, иначе будет зацикливание;

- ОПЕРАТОР может быть циклом (причём на количество вложений цикла в цикл ограничений нет).

Пример: Написать программу, которая вывод на экран таблицу значений функции $y=2x^2-5x-8$ в диапазоне -4 до 4. Шаг изменения аргумента 0,5.

```
void main (){
        float x, dx; // аргумент и его приращение
        float x1, x2; // диапазон изменения аргумента
        float y; // значение функции
        x1 = -4;
        x2 = 4;
        dx = 0.5;
        x = x1;
        printf (“----------\n”);
        printf (“   x |   y\n”);
        printf (“----------\n”);
        while (x < x2){
                y = 2*x*x – 5*x - 8;
                printf (“%3.2f  | %3.2f\n”, x, y);
                x += dx;
        }
        printf (“----------\n”);
        printf (“для завершения нажмите Enter\n”);
        getch();
}
```

Особенности цикла с постусловием:

- Тело цикла исполняется хотя бы один раз;
- ОПЕРАТОР не заключается в операторные скобки, даже если он составной;
- ОПЕРАТОР может быть циклом (причём на количество вложений цикла в цикл ограничений нет).

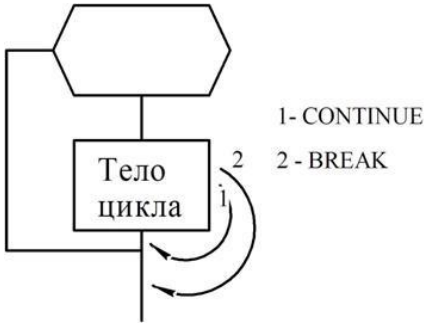
Пример: Написать программу, которая проверяет, является ли введенное пользователем число простым.

```
void main(){
    int n; // число
    int d; // делитель
    int r; // остаток от деления n на d
    printf (“введите целое число\n”);
    scanf(“%i”, &n);
```

```
    d = 2; // сначала будем делить на 2
    do{
        r = n % d;
        if (r != 0) d++;
    } while (r != 0); // пока n не разделится на d
    if (d == n)
        printf (“%i – простое число”, n);
    else
        printf (“%i – не простое число”, n);
    getch();
}
```

Особенности цикла с параметром:

- Если ОПЕРАТОР составной, то он заключается в операторные скобки;
- Количество вложений цикла в цикл *не более 15*;
- Параметр цикла после окончания цикла не определён;
- Параметр цикла меняется только в заголовке, не следует делать этого в теле цикла.
- Для досрочного выхода из цикла используются процедуры:
  - BREAK: передаёт управление на оператор, следующий за циклом;
  - CONTINUE: передаёт управление на начало тела цикла для следующей итерации.



Для цикла с параметром шаг всегда равен единице, но его можно сделать любым путём функционального преобразования его внутри тела цикла, но без изменения самого параметра.

Пример: Написать программу, которая выводит таблицу квадратов первых пяти целых положительных нечетных чисел.

```
void main(){
    int x = 1; // число
    int y; // квадрат числа
    int i; // счетчик циклов
    printf(“Таблица квадратов\n”);
    printf(“--------------------------\n”);
    printf(“Число\t Квадрат\n”);
    printf(“--------------------------\n”);
    for (i = 1; i <= 5; i++){
        y = x*x;
        printf(“%3i\t %4i\n”, x, y);
        x += 2;
```

```
    }
    printf("--------------------------\n");
    getch();
}
```

Логический тип данных. Логические операции not, and, or. Нахождение значений логических выражений

Переменные логического типа описываются как bool. Они могут принимать только два значения - False (ложь, 0) и True (истина, 1). Они описываются в разделе описания переменных:

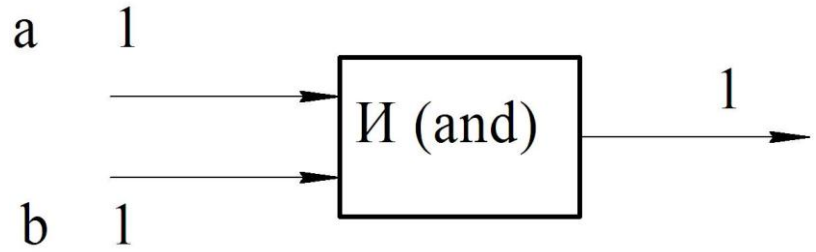
bool flag;

Переменные логического типа обычно получают значения в результате выполнения логических операций и операций сравнения. В программировании имеются логические операции, применяемые к переменным логического типа. Это операции not, and, or и xor.

Таблица 5 – Представление значений типа bool и применение к ним операций not, and, or и xor.

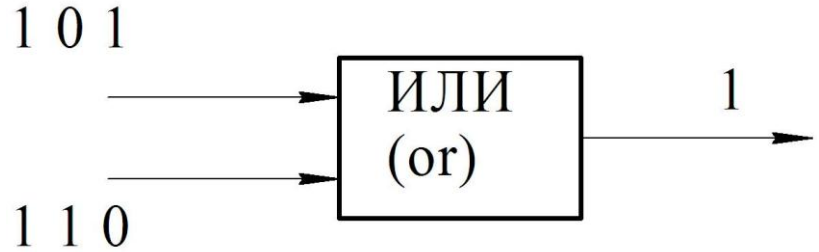
Значения операндов		Результат операции			
X	Y	not X	X and Y	X or Y	X xor Y
False (0)	False (0)	True (1)	False (0)	False (0)	False (0)
False (0)	True (1)	True (1)	False (0)	True (1)	True (1)
True (1)	False (0)	False (0)	False (0)	True (1)	True (1)
True (1)	True (1)	False (0)	True (1)	True (1)	False (0)

Схема логического умножения A and B



Результат операции and (и) есть истина, только если оба ее операнда истинны, и ложь во всех других случаях.

Схема операции or:



Результат операции or (или) есть истина, если какой-либо из ее операндов истинен, и ложен только тогда, когда оба операнда ложны.

Логические операции, операции отношения и арифметические операции часто встречаются в одном выражении. Логические операции имеют более высокий приоритет, поэтому отношения, стоящие слева и справа от знака логической операции, должны быть заключены в скобки. Вообще принят следующий приоритет операций:

- not
- and, *, /, mod
- or, xor, +, -
- Операции отношения

Логическую операцию and еще называют логическим умножением, логическую операцию or - логическим сложением, а xor логическим сложением по mod 2.

Кроме того, порядок выполнения операций может изменяться скобками. Например, в логическом выражении расставим порядок действий: A or B and not (A or B). Сначала выполняется заключенная в скобки операция or, а затем операции not, and, or. Если подставить вместо переменных A и B значения True и False, то, используя уже рассмотренный порядок действий, получим значение всего выражения равное True.

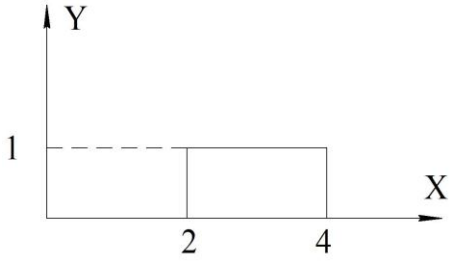
Рассмотрим отрицание сложного предиката:

NOT ((X>2) AND (X<4)) —ему соответствует→ (X<=2) OR (X>=4)

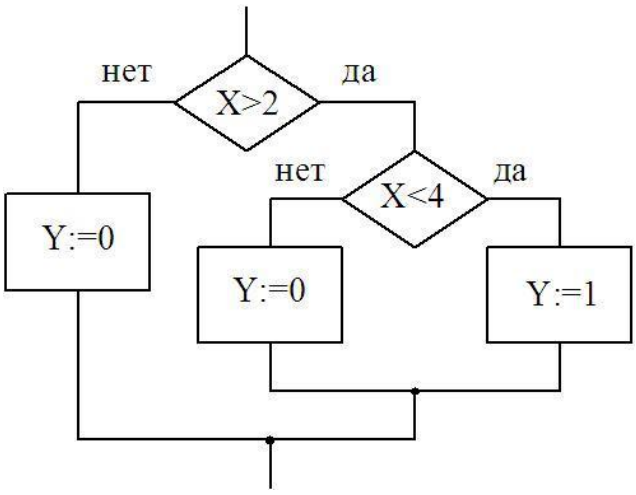
Т.е. при отрицании сложных предикатов операции отношения меняются на противоположные, а логические OR на AND.

При анализе составных предикатов оказывается, что если один условный оператор стоит в ветке «да» второго оператора, то мы можем поместить все в один предикат, в котором условия объединяться логической операций AND.

Рассмотрим задачу: для функции единичного скачка вывести y для x, лежащего в следующей области: от 2 до 4, не включая концы отрезков.



После выполнения выше сказанного предикат будет выглядеть следующим образом: (x>2) and (x<4).

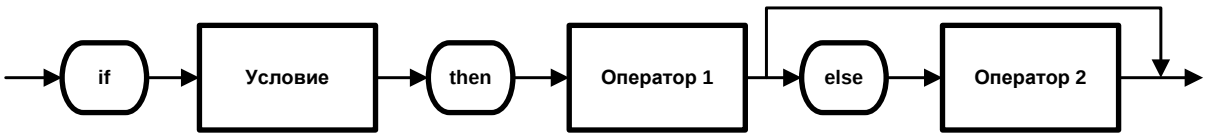


Если один условный оператор стоит в ветке «нет» второго оператора, то они могут быть объединены с помощью операции OR: (x <= 2) or (x >= 4).

Для получения n веток потребуется n-1 условный оператор (допустим мы хотим получить 6 веток, тогда нам потребуется 5 условных операторов).

Оператор условия If

Оператор условия If является одним из самых популярных средств, изменяющих естественный порядок выполнения операторов программы. Синтаксическая диаграмма этого оператора выглядит следующим образом:



Как видно из диаграммы, он может принимать одну из следующих форм:

Полный оператор	Неполный оператор
if «условие» then «оператор 1» else «оператор 2»	if «условие» then «оператор 1»

Условный оператор работает по следующему алгоритму:

Сначала вычисляется выражение, записанное в условии. В результате его вычисления получается значение булевского типа. В первом случае, если значение есть True, выполняется « оператор1 », указанный после слова Then. Если результат вычисления выражения есть False, то выполняется « оператор2 ». Во втором – если результат выражения True, то выполняется «оператор», если False – оператор, следующий сразу за условным.

Особенности условного оператора:

- если «оператор 1» и «оператор 2» являются составными, то они заключаются в операторные скобки;

- «оператор 1» и «оператор 2» в свою очередь тоже могут быть условными операторами: (на количество вложений нет ограничений)

  - В неполный оператор можно вложить неполный (а) и полный (б);
  - В полный оператор также можно вложить неполный (в) и полный операторы.

а)

if «условие 1» then
    if «условие 2» then «оператор 1»

Предыдущее выражение можно записать и следующим образом:

if «условие 1» and «условие 2» then «оператор 1» (подробнее это объяснено в предыдущей теме)

б)

if «условие 1» then
    if «условие 2» then «оператор 1»
        else «оператор 2»

в)

if «условие 1» then
    if «условие 2» then «оператор 1»
else «оператор 2»

Пункты б, в - это коллизия. Чтобы пункт в работал так как он написан необходимо поставить вложенный If в операторные скобки:

```
if «условие 1» then
                    begin
                        If «условие 2» then «оператор 1»;
                    end
                   else «оператор 2»
```

Так к какому же if относится else? На этот счет предусмотрено следующее правило: else ВСЕГДА ОТНОСИТСЯ К ПОСЛЕДНЕМУ if.

Пример: Написать программу вычисления площади кольца. Программа должна проверять корректность исходных данных.

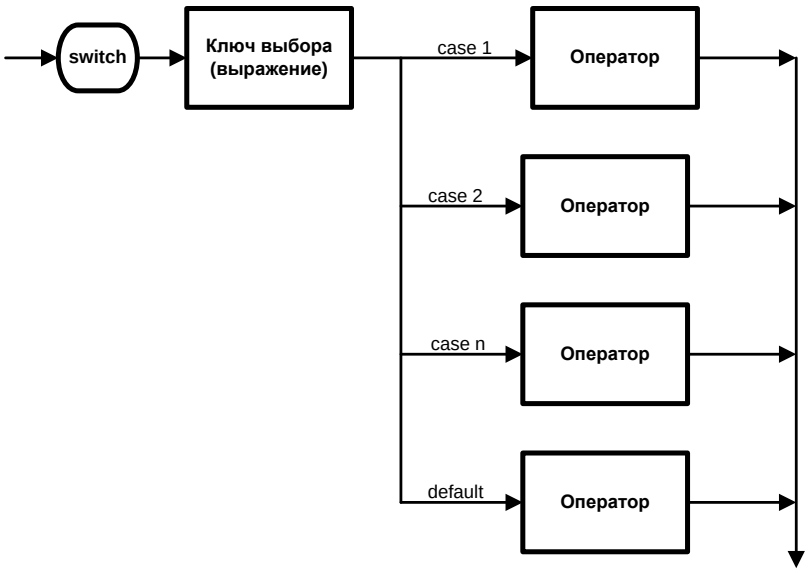
```
void main(){
    float r1, r2; // радиус кольца и отверстия
    float s; // площадь кольца
    printf("\nВведите исходные данные:\n");
    printf("радиус кольца (см) -> ");
    scanf("%f", &r1);
    printf("радиус отверстия -> );
    scanf("%f", &r2);
    if (r1 > r2){
        s = 2* 3.14 * (r1-r2);
        printf("\nПлощадь кольца %6.2f кв.см\n", s);
    }
    else{
        printf("'\nОшибка! Радиус отверстия не может быть больше радиуса
                                                                кольца. \n");
    }
    getch();
}
```

Оператор выбора Case

Ранее Вы познакомились с условным оператором If, который позволяет программе выполнять переходы на ту или иную ветвь по значению булевого условия. Используя несколько операторов If, можно производить ветвление по последовательности условий.

Вы уже, наверное, представили, насколько этот подход однообразный и утомительный, а также возможны ошибки связанные с коллизией. Существует другая управляющая структура, оператор выбора switch, которая позволяет построить ветвление по ряду условий в форме, более удобной для чтения программ.

Оператор выбора позволяет выбрать одно из нескольких возможных выполнений программы. Параметром, по которому осуществляется выбор, служит так называемый ключ выбора (или селектор) - выражение любого простого типа.



Общая форма записи следующая:

```
switch (выражение){
        case значение 1: оператор (группа операторов) break;
        case значение 2: оператор (группа операторов) break;
        . . .
        default: оператор (группа операторов) break;
}
```

Выбор оператора для выполнения осуществляется в зависимости от равенства значения переменной-селектора константе, указанной после слова case. То есть переменная-селектор и константы должны быть одного типа. Если значение переменной-селектора не равно ни одной из констант, записанных после слова case, то выполняются инструкции, расположенные после слова default.

Пример: Написать программу, которая запрашивает у пользователя номер дня недели, затем выводит название дня недели или сообщение об ошибке.

```
void main(){
        int nd; // номер дня недели
        puts (“\n Введите номер дня недели (1..7)”);
        printf(“->”);
        scanf(“%i”, &nd);
        switch (nd){
                case 1: puts(“Понедельник”); break;
                case 2: puts(“Вторник”); break;
                case 3: puts(“Среда”); break;
                case 4: puts(“Четверг”); break;
                case 5: puts(“Пятница”); break;
                case 6: puts(“Суббота”); break;
                case 7: puts(“Воскресенье”); break;
                default: puts(“Введенное число не верно”); break;
        }
        getch();
}
```

Контрольная работа по типам вычислительных процессов

Вариант 1.

1. Задание на циклы с предусловием и постусловием: В последовательности целых чисел, завершающейся нулем, подсчитать число элементов.
2. Задание на итерационные циклы: Вычислить приближенно значение суммы. Нужное приближение считается полученным, если вычислена сумма нескольких первых слагаемых, и очередное слагаемое оказалось по модулю меньше данного положительного числа ε (ε ввести в режиме диалога).

   $\frac{1}{2!}+\frac{3}{3!}+\frac{5}{4!}+\ldots \quad \sum_{k=1}^{\infty}\frac{2k-1}{(k+1)!}$
3. Записать в ячейку P единицу, если введенные целые числа A, B и C удовлетворяют условию A< B< C, и нуль в противном случае.

Вариант 2.

1. Задание на циклы с предусловием и постусловием: В последовательности целых чисел, завершающейся нулем, вывести те члены последовательности, которые при делении на 7 дают остаток 1, 2 или 5.
2. Задание на итерационные циклы: Вычислить приближенно значение суммы. Нужное приближение считается полученным, если вычислена сумма нескольких первых слагаемых, и очередное слагаемое оказалось по модулю меньше данного положительного числа ε (ε ввести в режиме диалога).

   $1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\ldots \quad \sum_{k=1}^{\infty}\frac{(-1)^{k+1}}{k} \qquad \ln 2$
3. Записать в ячейку P единицу, если существует треугольник с длинами сторон A, B и C, введенными в виде целых положительных чисел и нуль в противном случае.

Вариант 3.

1. Задание на циклы с предусловием и постусловием: В последовательности различных целых чисел, завершающейся нулем, найти разность между максимальным элементом и минимальным элементом.
2. Задание на итерационные циклы: Вычислить приближенно значение суммы. Нужное приближение считается полученным, если вычислена сумма нескольких первых слагаемых, и очередное слагаемое оказалось по модулю меньше данного положительного числа ε (ε ввести в режиме диалога).

   $1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\ldots \quad \sum_{k=1}^{\infty}\frac{(-1)^{k-1}}{2k-1} \qquad \pi/4$
3. Пять целых чисел расположить в порядке возрастания.

Вариант 4.

1. Задание на циклы с предусловием и постусловием: В последовательности символов, завершающихся «;», проверить, существует ли баланс открывающих и закрывающих скобок.

2. Задание на итерационные циклы: Вычислить приближенно значение суммы. Нужное приближение считается полученным, если вычислена сумма нескольких первых слагаемых, и очередное слагаемое оказалось по модулю меньше данного положительного числа ε (ε ввести в режиме диалога).

$\frac{1}{1*2}+\frac{1}{2*3}+\frac{1}{3*4}+\ldots$ $\quad \sum_{k=1}^{\infty}\frac{1}{k(k+1)}$ $\quad 1$

3. Перемножить три введенных числа. Если произведение четно, то разделить его на два, если нечетно, то прибавить единицу и разделить на два.

Вариант 5.

1. Задание на циклы с предусловием и постусловием: В последовательности символов, завершающихся точкой, подсчитать количество слов, предусмотрев различные разделители.
2. Задание на итерационные циклы: Вычислить приближенно значение суммы. Нужное приближение считается полученным, если вычислена сумма нескольких первых слагаемых, и очередное слагаемое оказалось по модулю меньше данного положительного числа ε (ε ввести в режиме диалога).

$\frac{1}{1*2*3}+\frac{1}{2*3*4}+\frac{1}{3*4*5}+\ldots$ $\quad \sum_{k=1}^{\infty}\frac{1}{k(k+1)(k+2)}$ $\quad 1/4$

3. Заменить нулевую цифру трехзначного числа суммой двух других, если это невозможно, то их положительной разностью.

Вариант 6.

1. Задание на циклы с предусловием и постусловием: Является ли введенное число X перевертышем? (например:15251 –является, а 123 - нет)
2. Задание на итерационные циклы: Вычислить приближенно значение суммы. Нужное приближение считается полученным, если вычислена сумма нескольких первых слагаемых, и очередное слагаемое оказалось по модулю меньше данного положительного числа ε (ε ввести в режиме диалога).

$1-\frac{1}{3^3}+\frac{1}{5^3}-\frac{1}{7^3}+\ldots$ $\quad \sum_{k=1}^{\infty}\frac{(-1)^{k-1}}{(2k-1)^3}$ $\quad \pi^3/32$

3. Напечатать «да», если введенное число делится на 3 и 2 одновременно, «или», если или на 3 или 2, «нет», если не делится ни на 3, ни на 2.

Вариант 7.

1. Задание на циклы с предусловием и постусловием: Ввести натуральное число X (меньше32768) и K (не более 5). Получить сумму K последних цифр числа X.
2. Задание на итерационные циклы: Вычислить приближенно значение суммы. Нужное приближение считается полученным, если вычислена сумма нескольких первых слагаемых, и очередное слагаемое оказалось по модулю меньше данного положительного числа ε (ε ввести в режиме диалога).

$x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots$ $\quad \frac{(-1)^k * x^{(7k+1)}}{(2k+1)!}$ $\quad \sin x$

3. Напечатать «да», если точка с целочисленными координатами X и У попадает внутрь круга радиусом =25, «нет», если лежит вне его, «на окружности», если попадает на окружность.

Вариант 8.

1. Задание на циклы с предусловием и постусловием: Ввести натуральное число X (меньше32768). Выбросить из записи этого числа все цифры 0. Например: 3007 – 37, 100 – 1, 209 – 29.
2. Задание на итерационные циклы: Вычислить приближенно значение суммы. Нужное приближение считается полученным, если вычислена сумма нескольких первых слагаемых, и очередное слагаемое оказалось по модулю меньше данного положительного числа ε (ε ввести в режиме диалога).

$$1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots \qquad \sum_{k=0}^{\infty}\frac{x^k}{k!} \qquad e^x$$

3. Ввести в виде целых чисел X и У координаты точки А. Вывести, в какую четверть попадает эта точка. Если она лежит на одной из осей, вывести на экран «ось».

Вариант 9.

1. Задание на циклы с предусловием и постусловием: Является ли введенное число X числом Фибоначчи?
2. Задание на итерационные циклы: Вычислить приближенно значение суммы. Нужное приближение считается полученным, если вычислена сумма нескольких первых слагаемых, и очередное слагаемое оказалось по модулю меньше данного положительного числа ε (ε ввести в режиме диалога).

$$-\frac{1}{2!}+\frac{3}{4!}-\frac{5}{6!}+\ldots \qquad \sum_{k=1}^{\infty}\frac{(2k-1)(-1)^k}{(2k)!}$$

3. По четырем сторонам и одному углу распознать вид четырехугольников: квадрат, прямоугольник, ромб и другой.

Вариант 10.

1. Задание на циклы с предусловием и постусловием: Ввести число К. Найти первые К натуральных чисел, которые состоят из одинаковых цифр.
2. Задание на итерационные циклы: Вычислить приближенно значение суммы. Нужное приближение считается полученным, если вычислена сумма нескольких первых слагаемых, и очередное слагаемое оказалось по модулю меньше данного положительного числа ε (ε ввести в режиме диалога).

$$\frac{4}{0!}+\frac{9}{1!}+\frac{16}{2!}+\ldots \qquad \sum_{k=1}^{\infty}\frac{k^2+2k+1}{(k-1)!}$$

3. Ввести семь целых чисел. Напечатать только те из них, которые лежат в интервале от –3 до 3.

Вариант 11.

1. Задание на циклы с предусловием и постусловием: В последовательности целых чисел, завершающейся нулем, найти количество и сумму тех членов, которые делятся нацело на 5 и не делятся на 7.
2. Задание на итерационные циклы: Вычислить приближенно значение суммы. Нужное приближение считается полученным, если вычислена сумма нескольких первых слагаемых, и очередное слагаемое оказалось по модулю меньше данного положительного числа ε (ε ввести в режиме диалога).

   $9+\frac{1}{2!}+\frac{9}{3!}+\frac{25}{4!}+\ldots \qquad \sum_{k=0}^{\infty}\frac{(2k-3)^2}{k!}$
3. Ввести целое X. Вывести Y = X + 10, если X <= -5, Y =X, если -5< X<5, Y=2X, если X>=5.

Вариант 12.

1. Задание на циклы с предусловием и постусловием: Вывести «да», если в последовательности символов, завершающихся «\», содержится символ, отличный от латинских букв, лежащих в интервале от «a» до «z».
2. Задание на итерационные циклы: Вычислить приближенно значение суммы. Нужное приближение считается полученным, если вычислена сумма нескольких первых слагаемых, и очередное слагаемое оказалось по модулю меньше данного положительного числа ε (ε ввести в режиме диалога).

   $-1-\frac{1}{2^2}-\frac{1}{3^2}-\frac{1}{4^2}-\ldots \qquad \sum_{k=0}^{\infty}\frac{(-1)^{2k+1}}{(k+1)^2}$
3. Записать в ячейку P единицу, если все целые A, B и C – положительны, минус единицу, если все отрицательны, и нуль, если они различны по знаку.

Вариант 13.

1. Задание на циклы с предусловием и постусловием: Найти НОД и НОК двух введенных чисел.
2. Задание на итерационные циклы: Вычислить приближенно значение суммы. Нужное приближение считается полученным, если вычислена сумма нескольких первых слагаемых, и очередное слагаемое оказалось по модулю меньше данного положительного числа ε (ε ввести в режиме диалога).

   $x-\frac{x^3}{3!}+\frac{x^5}{5!}+\frac{x^7}{7!}+\ldots \qquad x<\infty \qquad \ln(1+x)$
3. Если сумма целых чисел A, B и C меньше единицы, то наименьшее из этих трех чисел заменить суммой двух других.

Тема 4. Массивы

Массивы - это совокупность однородных элементов, хранящихся в смежных ячейках памяти, обращение к каждому элементу производится по имени всей структуры с указанием индекса.

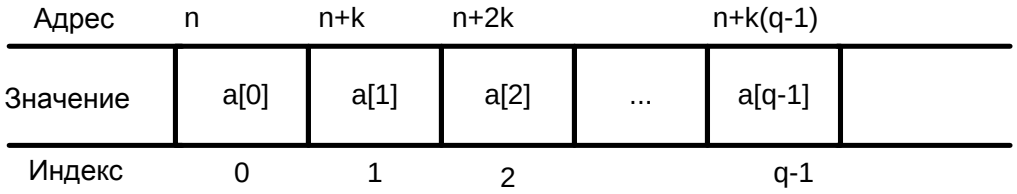
Массив определяется следующими характеристиками:

- Имя;
- Тип компонент, которые в нем хранятся;
- Размер массива.

Например, int mass[5];

Одномерные массивы

Графически расположение массива в памяти компьютера можно представить следующим образом.



Данный массив содержит q элементов с индексами от 0 до q-1. Каждый элемент занимает k байт, а их расположение последовательно. Адрес i-того элемента n+k*i.

По выделению памяти массивы разделяются на статические и динамические.

Доступ к элементу массива осуществляется через индекс (номер) элемента. Для доступа, вывода и ввода массива удобно использовать инструкции циклов (for, while).

Типичной ошибкой при использовании массивов является обращение к несуществующему элементу, иными словами, выход индекса за допустимое значение.

Например: int mass[5]; mass[6] = 1;

Для выделения динамической памяти под массив необходимо описать указатель, представляющий собой начальный адрес хранения элементов массива:

int *p;

Для использования функций динамического выделения памяти необходимо подключение библиотеки malloc.h. Функции динамического выделения памяти:

- void* malloc (размер массива в байтах);
- void* calloc (число элементов, размер элемента в байтах);

Поскольку обе представленные функции в качестве возвращаемого значения имеют указатель на пустой тип void, требуется явное приведение типа возвращаемого значения.

Для определения размера массива в байтах, используемого в качестве аргумента функции malloc() требуется количество элементов умножить на размер одного элемента. Поскольку элементами массива могут быть как данные простых типов, так и составных типов (например, структуры), для точного определения размера элемента в общем случае рекомендуется использование функции int sizeof(тип); Эта функция определяет количество байт, занимаемое элементом указанного типа.

Память, динамически выделенная с использованием функций calloc(), malloc(), может быть освобождена с использованием функции free(указатель);

Инициализация массива представляет собой набор начальных значений элементов массива, указанных в фигурных скобках через запятую. Например,

int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

Если количество инициализирующих значений, указанных в фигурных скобках, меньше, чем количество элементов массива, указанное в квадратных скобках, то все оставшиеся элементы в массиве (для которых не хватило инициализирующих значений) будут равны нулю.

int a[10] = {0}.

Также возможно заполнение массива случайными числами. Для этого существует два оператора rand и srand. Для их использования требуется подключить библиотеку stdlib.h.

- int rand() – возвращает случайное целое число в диапазоне от 0 до RAND_MAX. Перед первым обращением к функции rand необходимо инициализировать генератор случайных чисел. Для этого надо вызвать функцию srand.
- void srand(unsigned int) – инициализирует генератор случайных чисел. Обычно в качестве параметра функции используют переменную, значение которой предсказать заранее нельзя, например, это может быть текущее время.

Приведем пример создания статического массива и заполнением его случайными числами.

```
void main(){
        int arr[5]; // массив из 5 элементов
        int i;
        srand(time(NULL));
        for (i = 0; i < 5; i++){
                arr[i] = rand();
                printf (“Случайное значение %i элемента массива -> %i\n”, i, arr[i]);
        }
        getch();
}
```

Приведем пример создания массива с помощью динамического выделения памяти. Требуется написать программу, которая выводит минимальный элемент введенного с клавиатуры массива целых чисел.

```
void main(){
        int *a; // массив
        int size; // размер массива
        int min; // номер минимального элементы
        int i;
        printf (“\nПоиск минимального элемента массива\n”);
        printf(“Введите размер массива -> ”);
        scanf(“%i”, &size);
        a = (int*) malloc (size * sizeof(int));
        printf(“Введите в одной строке элементы массива, \n”);
        printf(“%i целых чисел и нажмите Enter\n”, size);
        for (i = 0; i < size; i++){
                scanf(“%i”, &a[i]);
```

```
        }
        min = 0; // предположим, что первый элемент минимальный
        for (i = 1; i < size; i++){
                if (a[i] < a[min])
                        min = i;
        }
        printf("Минимальный элемент массива: ");
        printf("a[%i] = %i", min+1, a[min]);
        getch();
}
```

Многомерные массивы

Отличие многомерного массива от одномерного в том, что в одномерном массиве положение элемента определяется по одному индексу, а в многомерном – несколькими. Самым простым примером является матрица. Можно сказать, что двумерный массив – это одномерный массив одномерных массивов.

Пример двумерного массива A[4][5]:

A[0][0]	A[0][1]	A[0][2]	A[0][3]	A[0][4]
A[1][0]	A[1][1]	A[1][2]	A[1][3]	A[1][4]
A[2][0]	A[2][1]	A[2][2]	A[2][3]	A[2][4]
A[3][0]	A[2][1]	A[3][2]	A[3][3]	A[3][4]

В первых квадратных скобках указывается количество строк двумерного массива, а во вторых – количество столбцов двумерного массива.

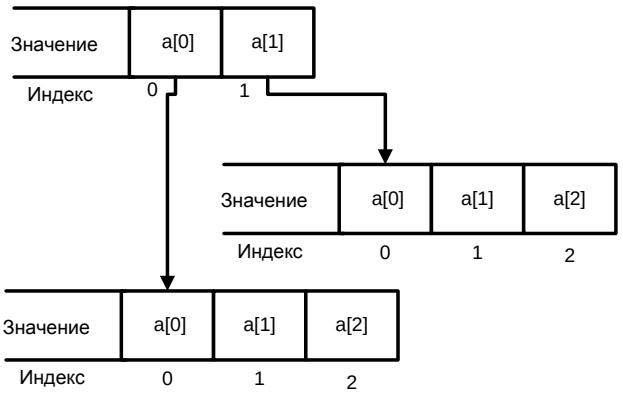
Инициализация двумерного массива происходит по той же схеме, что и инициализация одномерного массива:

int a [2][3] = {{3, 5, 1}, {7, 9, 4}}

Число внутренних фигурных скобок совпадает с числом строк в массиве. В каждой фигурной скобке через запятую 3 описаны элемента, так как количество столбцов 3.

Кроме статического выделения памяти при работе с многомерными массивами возможно также и динамическое выделение памяти. Для этого необходимо сначала создать массив указателей, после чего каждому из элементов этого массива присваивается адрес нового массива. Для удаления такого массива необходимо повторить операцию в обратном порядке.

Схематично выделение памяти для двумерного массива будет иметь следующий вид:



Приведем пример выделения динамической памяти под двумерный массив размером 5x6.

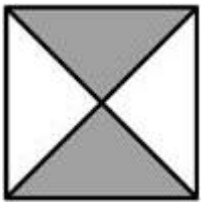
```
void main(){
        int **p; // указатель для выделения памяти под двумерный массив
        int i;
        p = (int**) malloc (5 * sizeof(int*)); //Выделили память под массив указателей,
                                                                                под строки
        for (i = 0; i < 5; i++){
                p[i] = (int*) malloc (6 * sizeof(int)); // выделяем память в каждой строке новый
                                                                        массив, под столбцы
        }
        int j;
        srand(time(NULL));
        for (i = 0; i < 5; i++){
                for (j = 0; j < 6; j++){
                        p[i[j] = rand();
                        printf ("Случайное значение [%i][%i] элемента массива -> %i\n", i, j,
                                                                                p[i][j]);
                }
        }
        getch();
}
```

Многомерный массив может быть и трехмерным, если представлять это графически, то он будет иметь форму куба. Задание такого массива будет уже соответственно: int arr[2][3][4] или int ***p. Выделение динамической памяти будет выполняться по аналогии с двумерным массивом.

Контрольная работа по массивам

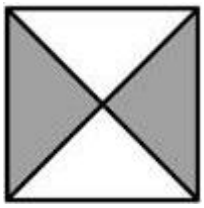
Вариант 1.

1. Переписать из массива A в массив B те элементы массива A, индексы которых являются степенями двойки (1,2,4,8,16,…). Размеры массивов A и B вводятся с клавиатуры, а сами массивы заполняются случайными числами.
2. Найти максимальный элемент в заштрихованной области двумерного массива. Размер и данные двумерного массива задаются с клавиатуры.



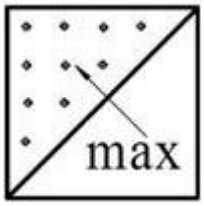
Вариант 2.

1. Переписать из массива A в массив B те элементы массива A, индексы которых являются числами Фибоначчи: 1,2,3,5,8,13,21,34… Размеры массивов A и B вводятся с клавиатуры, а сами массивы заполняются случайными числами.
2. Найти сумму элементов массива в заштрихованной области двумерного массива. Размер и данные двумерного массива задаются с клавиатуры.



Вариант 3.

1. Переписать из массива A в массив B те элементы массива A, индексы которых являются полными квадратами (1,4,9,16,25…). Размеры массивов A и B вводятся с клавиатуры, а сами массивы заполняются случайными числами.
2. Найти максимальный элемент в области ниже побочной диагонали двумерного массива и заполнить им область выше побочной диагонали. Размер массива вводится с клавиатуры и заполняется случайными числами.

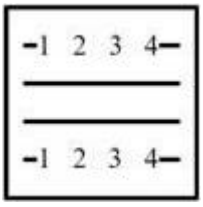


Вариант 4.

1. В массиве заменить каждый из элементов суммой соседей, кроме крайних. Размер массива вводится с клавиатуры и заполняется случайными числами.
2. Найти минимальную сумму элементов в строках двумерного массива. Размер массива вводится с клавиатуры и заполняется случайными числами.

Вариант 5.

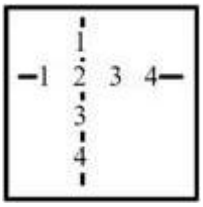
1. В массиве поменять местами попарно нечетные элементы с четными. Размер массива вводится с клавиатуры и заполняется случайными числами.
2. Определить номера строк двумерного массива, которые равны между собой. Размер и данные двумерного массива задаются с клавиатуры.



Вариант 6.

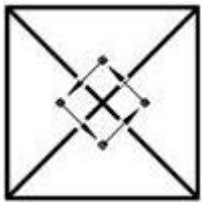
1. Массив A содержит K элементов. Вычислить сумму: P=A(1)+A(1)*A(2)+A(1)*A(2)*A(3)+…+A(1)*A(2)*…*A(K). Размер массива вводится с клавиатуры и заполняется случайными числами.

2. Определить номера строк и столбцов двумерного массива, которые равны между собой. Размер и данные двумерного массива задаются с клавиатуры.



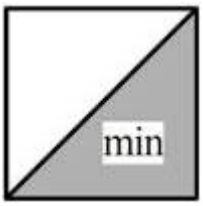
Вариант 7.

1. Переписать элементы массива в обратном порядке, не пользуясь другим массивом. Размер массива вводится с клавиатуры и заполняется случайными числами.
2. В двумерном массиве поменять местами элементы как показано на рисунке. Размер массива вводится с клавиатуры и заполняется случайными числами.



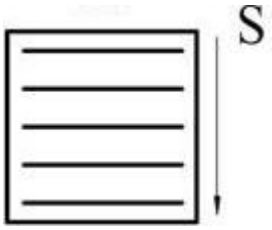
Вариант 8.

1. Дан массив A из K элементов. Найти максимальное число, встречающееся в массиве более одного раза: A=1,9,8,4,8,5. Размер массива вводится с клавиатуры и заполняется случайными числами.
2. Найти минимальный элемент в заштрихованной области двумерного массива. Размер массива вводится с клавиатуры и заполняется случайными числами.



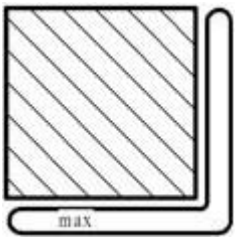
Вариант 9.

1. Дан массив A из K элементов и массив B из M элементов. Сформировать массив C, являющийся пересечением двух заданных, т.е. включить в него элементы, принадлежащие одновременно двум массивам A и B. Размеры массивов A и B вводятся с клавиатуры, а сами массивы заполняются случайными числами.
2. Упорядочить строки двумерного массива по возрастанию значения суммы элементов в этих строках. Размер массива вводится с клавиатуры и заполняется случайными числами.



Вариант 10.

1. Найти наименьшее среди элементов массива A, которое не входит в массив B. Размеры массивов A и B вводятся с клавиатуры, а сами массивы заполняются случайными числами.
2. Найти в двумерном массиве максимальный элемент в каждой диагонали, параллельной главной. Размер массива вводится с клавиатуры и заполняется случайными числами.

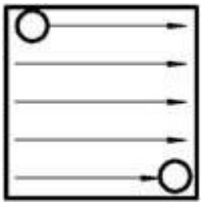


Вариант 11.

1. Обнулить все компоненты массива, находящиеся между минимальным и максимальным элементами. Размер массива вводится с клавиатуры и заполняется случайными числами.
2. Найти в двумерном массиве минимальный элемент в каждой диагонали, параллельной побочной. Размер массива вводится с клавиатуры и заполняется случайными числами.

Вариант 12.

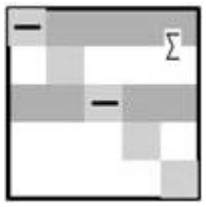
1. Сделать исходный массив чередующимся по четности, прибавляя единицу к числу в том случае, если такой зависимости нет. Размер массива вводится с клавиатуры и заполняется случайными числами.
2. Упорядочить элементы в строках двумерного массива по возрастанию. Вывести первый элемент в первой строке полученного массива и последний элемент в последней строке. Размер массива вводится с клавиатуры и заполняется случайными числами.



Вариант 13.

1. Упорядочить массив по убыванию методом пузырька. Размер массива вводится с клавиатуры и заполняется случайными числами.

2. Вычислить сумму элементов в строках двумерного массива, где элементы, стоящие на главной диагонали, отрицательны. Размер массива вводится с клавиатуры и заполняется случайными числами.



Тема 5. Функции в языке программирования С

Функции, которые создаются программистом требуют объявления прототипа. Прототип дает основную информацию по использованию функции: какое значение функция возвращает, как вызывается, какие аргументы принимает. Обязательно в конце прототипа ставить точку с запятой, иначе компилятор решит, что вы описываете функцию.Функция может не принимать параметров, а может принимать несколько. Важно, что тип каждого фактического параметра в инструкции вызова функции должен совпадает с типом соответствующего формального параметра, указанного в объявлении функции.

Приведем пример прототипов двух функций:

- float f1 (int x, int y); Данная функция имеет тип возвращаемого значения float, имя f1 и два аргумента x и y типа int.
- void f2(); Данная функция не возвращает значение, имеет имя f2 и не принимает никаких аргументов.

Когда программист приступает к описанию функции, то начинает с написания прототипа, но без точки с запятой в конце. Сразу после прототипа должен идти блок, заключенный в фигурные скобки { }.

Приведем пример написания и использования функции деления двух чисел.

```
float div (int num1, int num2); // прототип функции
void main(){
        int n1;
        int n2;
        printf (“Введите два числа для умножения : ”);
        scanf (“%d”, &n1);
        scanf (“%d”, &n2);
        printf (“Результат деления %d\n”,  div(n1, n2));
        getch();
}
float div (int num1, int num2){
        return num1/num2;
}
```

Описанная программа начинается с прототипа функции деления. Функция div принимает два целочисленных значения и возвращает результат деления.

Определение прототипов функций необходимы только если фактическое определение самой функции будет располагаться после main функции. Если же функцию определить до главной функции, то прототип не нужен.

Ключевое слово return нужно для того, чтобы заставить функцию возвращать значение. Если функция возвращает значение типа void, это значит, то функция не возвращает значения.

Написание отдельных функций необходимо, когда в программе есть блок кода, который необходимо выполнять в разных местах программы, возможно, и по нескольку раз. Таким образом, один раз объявив функцию, и вызывая ее в нужных местах кода, мы сделаем свою программу более читабельной, и это сэкономит нам много места. Более

того, наличие функции позволит нам легче и быстрее модифицировать код, если это станет необходимо.

Еще одной причиной является разделение всего кода на отдельные логические части, что позволяет сложные задачи разбить на более простые задачи.

Рекурсивные функции

Функция, которая вызывает сама себя, называется рекурсивной функцией.

Для завершения процесса рекурсии в алгоритме рекурсивной функции должна быть веточка, обеспечивающая непосредственное завершение рекурсии.

Приведем пример: необходимо написать рекурсивную функцию вычисления факториала.

```
int factor (int k){
        if (k == 1)
                return 1;
        else{
                return(k*factor(k-1));
        }
}
void main(){
        int n; // число, факториал которого надо вычислить
        int f; // факториал числа n
        puts(“Введите число, факториал которого надо вычислить”);
        printf (“->”);
        scanf(“%u”, &n);
        f = factor(n);
        printf (“\nФакториал числа %u равен %u”, n, f);
        getch();
}
```

Функции работы с файлами

Чтобы работать с файлом, необходимо проделать ряд операций:

1. Открыть файл. Открывать файл можно для чтения, записи, чтения и записи, переписывания или записи в конец. При открытии файла могут возникнуть ошибки, например, файла может не существовать, у вас может не быть прав доступа к файлу и пр.
2. Работа с файлом. Это по большей части чтение или запись.
3. Закрыть файл. Файл является внешним ресурсом для программы, если него не закрыть, то он продолжит «висеть» в памяти.

Для работы с файлом необходим объект FILE. Этот объект хранит идентификатор файлового потока и информацию, которая нужна, чтобы им управлять, включая указатель на его буфер, индикатор позиции в файле и индикаторы состояния. Создание и выделение памяти под объект типа FILE осуществляется с помощью функции fopen. Функция fopen открывает файл. Она получает два аргумента – строку с адресом файла и строку с режимом доступа к файлу. Имя файла может быть как абсолютным, так и относительным.

fopen возвращает указатель на объект FILE, с помощью которого далее можно осуществлять доступ к файлу.

FILE* fopen(const char* имя_файла, const char* режим);

В таблице приведены режимы, с которыми возможно открытие файла

Таблица 6 – Режимы и доступные с ними действия

Режим	Действие
r	Файл открывается только для чтения.
w	Файл открывается только для записи. Если файл с указанным именем в качестве первого параметра fopen уже существует и содержит данные, то новы данные записываются поверх старых, т.е. старый файл фактически уничтожается.
a	Добавление в файл. Файл открывается для записи данных в конец существующего файла. Если файл с указанным именем в качестве первого параметра функции fopen не существует, то он будет создан.

В случае успешного открытия файла функция fopen возвращает указатель на поток, из которого можно читать или в который можно записывать. Если по какой-либо причине операция открытия файла не была выполнена, fopen возвращает NULL. В этом случае, чтобы получить информацию л причине ошибки, следует обратиться к функции ferror.

Перечислим функции для работы с файлом:

- Функция ferror возвращает ненулевое значение, если последняя операция с указанным потоком завершилась ошибкой.
  int ferror (FILE *поток);
- Для выполнения операции вывода в файл необходимо использовать функцию fprintf, которая имеет следующий синтаксис:
  int fprintf(FILE *Поток, Формат, Список переменных);
  Файл должен быть уже открыт к моменту записи в файл.
- Для выполнения записи строки символов необходимо использовать команду fputs.
  char* fputs (char *строка, FIE *поток);
  Команда записывает указанную строку в указанный файл, которйы должен быть заранее открыт. Символ конца строки в поток не записывается.
- Для выполнения операции чтения из файла необходимо использовать функцию fscanf.
  int fscanf(FILE *Поток, const char* Формат, Список Адр);
  Эта команды выполняет чтение значений переменных из файла, связанного с потоком, указанным в качестве первого параметра. Файл должен быть уже открыт.
- Еще одна команда для чтения из файла – это fgets.
  char* fgets(char* строка, int количество символов, FILE * поток);
  Эта команда читает из указанного потока символы и записывает их в строку, указанную при вызове функции. Чтение заканчивается, если прочитан

символ с номером «Количество символов - 1» или если очередной символ является символом новой строки.

- Функция проверки конца файла – feof.
  int feof (FILE *поток);
  функция возвращает ненулевое значение, если в результате выполнения последней операции чтения из потока достигнут конец файла.
- Функция закрытия файла – fclose.
  int fclose (FILE *поток);
  Функция закрывает указанный поток.

Пример: необходимо написать программу, которая вычисляет среднее арифметическое чисел, находящихся в файле numbers.txt

```
#define FNAME “numbers.txt”
void main(){
    char fname[2] = FNAME;
    FILE *in; // поток для текстового файла
    int a; // число
    int n = 0; //количество чисел
    int sum = 0; // сумма чисел
    float sr; //среднее арифметическое
    puts (“\nВычисление среднего арифметического\n”);
    //открываем файл в режиме чтения (r) текста (t)
    if ((in == fopen(fname, “rt”)) == NULL){
        printf (“Ошибка открытия файла для чтения”);
        getch();
        return;
    }
    while (!feof(in)){
        fscanf(in, “%i”, &a);
        sum += a;
        n++;
    }
    fclose (in); // закрываем файл
    sr = (float) sum / n;
    printf (“Введено чисел %i\n”, n);
    printf (“Сумма чисел %i\n”, sum);
    printf (“Среднее арифметическое %3.2f\n”, sr);
    getch();
}
```

Контрольная работа по функциям

1. Напишите функцию, которая вычисляет объем цилиндра. Параметрами функции должны быть радиус и высота цилиндра. Радиус и высота хранятся в файле.
2. Напишите функцию, которая возвращает максимальное число из пяти чисел, полученных в качестве аргумента. Числа хранятся в файле.

3. Напишите функцию вычисления процента, которая возвращает процент от полученного в качестве аргумента числа. Число, процент и результат расчета необходимо сохранить в файл.

4. Напишите функцию вычисления дохода по вкладу. Исходными данными для функции являются: величина вклада, процентная ставка (годовых) и срок вклада (количество дней). Результат вычислений необходимо сохранить в файл.

5. Написать функцию, которая выводит на экран и в файл строку из звездочек. Длина строки (количество звездочек) является параметром функции.

6. Напишите функцию, которая вычисляет объем и площадь поверхности параллелепипеда. Данные для вычисления объема и площади хранятся в файле.

7. Напишите функцию, которая считывает из файла имя и отчество, затем здоровается с пользователем. Вывод необходимо организовать на консоль.

8. Напишите функцию, которая считывает из файла строку и подсчитывает ее длину посимвольно.

9. Напишите функцию, которая высчитывает сумму и средне арифметическое последовательности положительных чисел, которые вводятся с клавиатуры. Числа и результаты операций сохраняются в файл.

10. Напишите функцию, которая преобразует введённое с клавиатуры десятичное число в двоичную форму. Число и его двоично представление необходимо сохранить в файл.

11. Напишите функцию, которая генерирует три последовательности из нескольких чисел случайным образом. Количество чисел в каждой последовательности вводится пользователем с клавиатуры. Последовательности необходимо сохранить в файл.

12. Напишите функцию, которая выводит в файл таблицу значений функции $y = -2.4x^2+5x-3$ в диапазоне от -2 до 2, с шагом 0,5.

13. Напишите функцию, которая сохраняет в файл степени числа, которое вводится с клавиатуры, степени от 0 до 10.