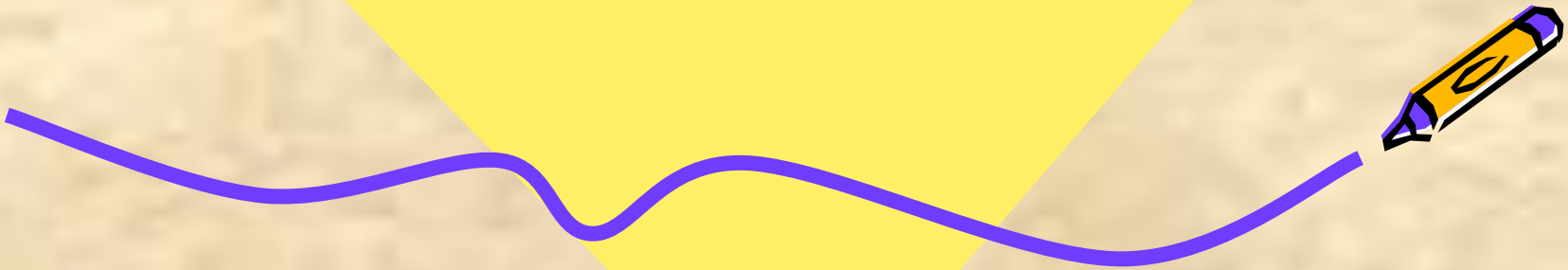
ОСНОВЫ ЯЗЫКА СИ

Литература

Макогон В.С. Язык программирования Си для начинающих.Одесса:НПФ "АСТРОПРИНТ",1993.-96 с.

НФ-2, ЧЗ-1, уф-35



Домашние задания по Си выполнить до 20 апреля

Все программы, разработанные для ТП (по 1-2 задания до л.р.9) выполнить на языке Си, сравнить результаты, оформить отчеты со ссылками на схемы алгоритмов в отчетах по ТП.

К зачету иметь конспекты лекций и ДЗ

ФОРМАТИРОВАННЫЙ ВЫВОД ДАННЫХ

printf("управляющая строка", параметры);

Символы преобразования:

d (или i)- используется для вывода целого десятичного числа (int),

u - десятичное целое без знака,

f - вещественное число в естественной форме (float),

e (E) - вещественное число в экспоненциальной форме,

g (G) - наиболее короткая запись числа из двух форм e или f,

c - для вывода отдельного символа,

s - для вывода строки символов,

o - восьмеричное число,

x - шестнадцатеричное число (буквы строчные),

X - шестнадцатиричное число (буквы прописные).

Перед символом преобразования м. стоять числовой коэффициент, явно указывающий количество позиций в выводимой строке, отведенных для элемента вывода.

printf (" %c = %5d \n",”z”,z); выводит z = 5

В общем случае шаблон преобразования записывается в виде

%[флажок] [длина] [.точность] [модификатор] символ преобразования

квадратные скобки означают, что данное поле может отсутствовать

ВВОД ДАННЫХ

Для ввода данных используется функция форматированного ввода scanf() вида

scanf ("управляющая строка",<адреса вводимых параметров>);

или

scanf ("список шаблонов", &a1, &a2, ...);

где **a1,a2 - имена переменных или аргументов**, **& - признак взятия адреса**.

В управляющей строке используются свои шаблоны со след. структурой:

[*] [длина] [модификатор] символ преобразования

*** - означает пропуск поля при вводе**, которое определено данным шаблоном (например, ***5d-** данная переменная читается, но не сохраняется).

Пример:

```
#include <stdio.h>
    void main()
      {
      int x,y,z;
      printf ("введите числа x,y \n");  /* элемент диалога */
      scanf ("%d %d", &x,&y);   /*функция ввода двух переменных*/
          z = x + y;
          printf (" \n z = %d \n", z);
      }
```

ТИПЫ ДАННЫХ

Объявление имеет формат вида

[<класс памяти>] <тип><идентификатор_1>
[[=<нач.зн.1;>][,<идентификатор_2>[=нач.зн.2]...];

Класс **auto** - автоматический, используется для описания локализованных в блоке переменных. Область действия ограничена той функцией или блоком, в которых она объявлена.

Класс **extern** - внешний, используется для явного описания глобальных переменных или для определения ссылок на внешние переменные.

Класс **register** - регистровый, описывает переменные, хранящиеся в сверхбыстродействующей памяти на регистрах; область действия аналогична переменным класса auto.

Класс **static** - статистический, подобно автоматическим переменным локализуется в блоке или функции, где они описаны, но при выходе из блока значения сохраняются.

Например. static char c, b; static int x=0;

№	Обозначение	Размер	Диапазон значений	Тип данных
1	char, signed char	1	-128...127	Символьный со знаком
2	unsigned char	1	0...255	Символьн.без знака
3	short, short int, signed short …	2	-32768...32767	Короткое целое со знаком
4	unsigned short, unsigned short int	3	0...65535	Короткое целое без знака
5	int, signed, signed int	1,2,4	зависит от реализации	целое
6	unsigned, unsigned int	1,2,4	зависит от реализации	целое без знака
7	long, signed long	4	-2147483648.. 2147483647	Длинное целое со знаком
8	unsigned long	4	0...4294967295	Длинное целое без знака
9	float	8	-3.4e-38...3.14e+38	С плав.точкой
10	double	8	-1.7e-308...1.7e+308	Удвоенной точности
11	long double	10	-3.4e-4932...3.4e4932	Длинное веществ. удвоенной точности

ОПЕРАЦИИ ПРИСВАИВАНИЯ

Операции присваивания могут быть простыми, многоступенчатыми и составными.

Простая операция присваивания имеют структуру: <идентификатор> = <выражение> **Тип правой части преобразуется к типу левой части.**

Многоступенчатое присваивание

(одно значение присваивается нескольким переменным)

Пр. i=j=k=6 эквивалент: i=(j=(k=6))

Операция выполняется справа налево.

Составные присваивания объединяют с операцией присваивания арифметические или побитовые операции (знак операции и равенство).

<идентификатор> <знак операции>= <выражение>;

выполняются как присваивание

<идентификатор> = <идентификатор> знак <выражение>

Пример: x + = 2; эквивалентно x = x + 2; z * = y + 1; z = z * (y + 1);

Таблица составных операций

Знак	Наименование операции	Примеры	Эквивалент
=	простое присваивание	x = 3	
+=	сложение с присваиванием	x += 1;	x=x + 1
-=	вычитание с присваиванием	x-=y;	x=x-y
*=	умножение с присваиванием	x*=y;	x=x*y
/=	деление с присваиванием	x/=10;	
%=	выделение остатка от деления с присваиванием	x%=10;	x=x%10
++	увеличение на 1(инкремент)	y++;	y+1
--	уменьшение на 1(декремент)	x--;	x-1
>>=	сдвиг двоичн. числа x вправо на l разрядов	x>>1	
<<=	сдвиг двоичн. числа влево		
&=	побитовая операция "и"с присваиванием для двоичных операндов	R&=m;	R=R&m
|=	поразрядная операция "или"с присваиванием	R|=m;	R=R|m
^=	исключающее "или" (сложение по модулю 2)	R^=m;	

Операции увеличения и уменьшения

Операция увеличения на единицу **(инкремента "++")** и операция уменьшения на единицу **(декремента "--")** относятся к унарным операциям присваивания.

Они соответственно увеличивают или уменьшают значение переменной на единицу. Переменная может быть целого или плавающего типа, либо указателем. Различают префиксную и постфиксную форму:

префиксная	**постфиксная**
++<идентификатор>	**<идентификатор>++**
--<идентификатор>	**<идентификатор>--**
просчитывается до использования идентификатора	**просчитывается после использования идентификатора**

В выражении **++N (--N) увеличение (уменьшение)** производится до использования . N, а в **N++ (N--) увеличение (уменьшение)** выполняется после использования N в выражении (после обработки остального выражения).

Пример:

1) x = 3;

Оператор У = ++x; //эквивалент У = x+1 присвоит x=4 и У=4

K = x--; K = 3; x = 2;

2) Для выполнения x = x+1; можно использовать x++; или ++x; (не имеет значения)

x = x-1; //эквивалент x--; --x;

3) Нельзя писать (x+к)++. Операндом может быть только переменная.

Оператор W= (X+K)++ не допустим.

Приоритет декремента и инкремента выше приоритета арифметических операций, если знаки - до переменной.

*Пример:

x * --к //эквивалент x * (--к)

A = 2; B = 4; C = (A+ B++)*3 //C = (2+4) * 3 = 18; B = 5;

c=(A+ ++B)*3 (2+5)*3 = 21 B=5

Во избежание возникновения ошибок при применении этих операций рекомендуется:

1) не применять данные операции к переменной, присутствующей в более чем одном аргументе функции.

2) не применять к переменной, которая входит в выражение более одного раза.

Пр. для int a, n=5; выражение a=n/2 + 2*(1+ n++) м. вычисляться

1 вариант a = n/2 + 2*(1+5) = n/2 + 12(n=6) = 3 + 12 = 15

2 вариант a = 5/2 + 2*(1+5) = 2 + 12 = 14

Все операции в языке СИ подразделяются на унарные (с одним операндом), бинарные (с двумя операндами) и тернарную (с тремя операндами: условная операция "?:").

ПРИОРИТЕТЫ ОПЕРАЦИЙ

Вес	Знак	Наименование операции	Тип операции	Порядок
1	( )	вызов функции	Выражение	слева направо ->
	[ ]	выделение элемента массива		
	.	выделение элемента структуры или объединения		
	->	выделение элемента структуры (объединения) адресуемой (го) указателем		
2	!	логическое отрицание	унарная	<- справа налево
	~	побитовое отрицание		
	-	изменение знака (унарный минус)		
	++	увеличение на единицу		
	--	уменьшение на единицу		
	&	определение адреса		
	*	обращение по адресу		
	(тип)	преобразование типа		
	sizeof	определение размера байтов		
3	*	умножение	бинарная арифметика	->
	/	деление		
	%	деление по модулю (определение остатка от деления)		
4	+	сложение	бинарная арифметика	->
	-	вычитание		

5	<<	сдвиг влево	сдвига	
	>>	сдвиг вправо		
6	<	меньше чем	отношения	->
	<=	меньше или равно		
	>	больше чем		
	>=	больше или равно		
7	==	равно	отношения	->
	!=	не равно		
8	&	побитовая операция "и"	поразрядная	->
9	^	побитовая искл. "или"	поразрядная	->
10	|	побитовая "или"	поразрядная	->
11	&&	логическая операция "и"	логическая	->
12	||	логическая операция "или"	логическая	->
13	?:	условная операция	тернарная	справа
14	=	присваивание*=, /=, +=, -=, %=, <<= >>=, |=, ^=, &=		справа <-
15	,	операция "запятая" (соединения)	бинарная	->

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

К ним относятся

2	- (унарный минус)	Для целых и вещ.
3	* / % (определение остатка от деления)	типов
4	+ (сложение) - (вычитание)	

Операции выполняются с учетом приоритета слева направо над целыми операндами и операндами плавающего типа. Исключение составляет % - только для целых операндов.

В качестве операнда может использоваться константа, идентификатор, вызов функции, индексное выражение, выражение выбора элемента или более сложное выражение, сформированное из (простых) операндов и знаков операций.

При делении целых положительных чисел типа unsigned (целое без знака) результат усекается до ближайшего меньшего числа. Если один из операндов отрицательный, то направление усечения результата (к нулю или от нуля) определяется реализацией (обычно к нулю).

Пример: 47/10 = 4; 47/(-10)=-4; -47/10 = -4 (к нулю);
9/4=2 9./4=2.25 -47/10 = -5 (от нуля)

Результатом операции % (деления по модулю) является остаток от деления первого операнда на второй. Знак результата зависит от реализации (обычно совпадает со знаком первого операнда)

47%10 = 7; 47%(-10) = 7; -47%10 = -7; -47%(-10) = -7

При выполнении арифметических операций производится **автоматическое преобразование типов** (если операнды разных типов) обычно к типу того операнда, который имеет наибольший размер:

в соответствии со следующей иерархией (к более высокому типу)
short<int<unsigned<long<unsigned long< float<doule<long double char

Условная операция " ? :"

Ее формат **<выражение1> ? <выражение2> : <выражение3>**

Все части обязательны! Ни одно <выражение> не может отсутствовать.

Алгоритм выполнения: Вычисляется <выражение1> и сравнивается с нулем; если его значение не=0 (истинно), то вычисляется <выраж2>; если =0(ложно), то вычисляется <выраж3>. Вычисляется лишь одно <выражение>, его значение и станет результатом выполнения операции (, которое должно быть присвоено к.-либо переменной).

(Функция ее аналогично условному оператору **if - else**), но результатом условной операции является значение.

Пример: вычислить максимум из двух значений

```
max = (a>в)?a:в;
// мах присваивается а или в в зависимости от выполнения условия а>в?
```

Если операнды различаются по типу, то тип результата задается правилами арифметического преобразования.

Применение условных выражений вместо условных операторов приводит в ряде случаев к получению более короткой программы.

УСЛОВНЫЙ ОПЕРАТОР IF

if (<выражение>) <оператор1> [;else <оператор2>];

Вычисляется <выражение>. Если оно истинно (не нуль), то выполняется <оператор1>
если ложно (=0), то <оператор2>.

Пример:

```
if (i>j) i++;     | if (k!=0)  эквивалентно  if(k)
else              |
        {
    j = j - 1;
    i++;
        }
```

Допускается использование вложенных операторов if в любой части (if или else).
Рекомендуется группировать операторы во вложенных операторах if, используя фигурные скобки. Если фигурные скобки опущены, то компилятор связывает часть 1el
с ближайшей сверху части 1if .

Примеры:

1)
```
main()
{  int a=4, b=9, c=5;
  if (a>b)
        { if (b<c) c=b; }
  else c=a;
  printf ("c=%d\n", c);
}  // Результат c=4
```

2)
```
main()
{  int a=4, b=9, c=5;
if (a>b)
        if (b<c) c=b;
  else c=a;
  printf ("c=%d\n", c);
}  // Результат c=5
```

2)
```
...char sign; int x,y; …
  if (sign=='+') x+=y;
  else  if (sign=='-') x-=y;
    else  if (sign=='*') x*=y;
          else printf("операция не верна");
```

При использовании многих вложенных if с одним параметром рекомендуется использовать оператор-переключатель (см.ниже switch).

Замечание: при объявлении переменных можно задавать _начальные значения (см. пример) в виде

<идентификатор> = <значение>

Если явно не задано значение переменной, то она инициализируется нулем.

Операции отношения

Они сравнивают 1-й операнд со 2-м. Результатом операции является1, если проверяемой отношение истинно, и 0 (нуль), если ложно.

Операнды могут быть целого (в том числе и символьные), плавающего типа или указателем.

Тип результата int.

Приоритет	**Знаки операций отношения**	
6	**<<=** **>>=**	Пр.1 a>=c>b эквив. (a>=c)>d Операции отношения одинакового приоритета выполняются слева направо.
7	**== (равно)** **!= (не равно)**	

Пр.2. y!=w==z выполняется как (y!=w)==z

Приоритет арифметических операций выше приоритета логических операций.

Пример: x*y > x+z //эквивалент (x*y) > (x+z) a=c>d; //a=1, если c>d истинно.

Операции "==" "!=" "<=" ">=" не рекомендуется использовать при работе с плавающими типами, ввиду их неточного представления в ЭВМ (ПК).

Обычно операции отношений применяются при формировании условных выражений в операторах while или if.

Пример: if (c!='x') i++;

ОПЕРАЦИЯ СДВИГА

<операнд1> << <операнд2> или <операнд1> >> <операнд2>

Производится сдвиг <операнда1> на число битов, указанных в <операнде2>.

Оба операнда д.б. **целыми** величинами. Тип результата - это тип левого операнда после обычных арифметических преобразований.

x << 2 /*сдвиг выполняется для двоичного представления числа, сдвиг X на 2 двоичных разряда эквивалентен умножению на 4 */

При сдвиге влево правые освобождающиеся разряды заполняются нулями. При сдвиге вправо метод заполнения освобождающихся левых битов зависит от типа, полученного после преобразования <операнда1>. Если тип unsigned, то освобождающиеся слева разряды также заполняются нулями. В противном случае (тип со знаком) они заполняются копией знакового бита.

Пр.int a=-2, res1, res2; 2=0010 -2 = 1111...1110

```
res1=a<<2;   //res1  1111...1000   -8
res2=a>>2;   //res2  1111...1111 -1
```

Преобразования, выполняемые операциями сдвига, не обеспечивают обработку ситуации переполнения и потери значимости. Информация теряется, если результат операции сдвига не может быть представлена типом первого операнда после преобразования.

ОПЕРАТОРЫ СИ

Все операторы языка Си могут быть условно разделены на категории:

1) условные операторы (if ... else, оператор выбора switch);
2) операторы цикла (for, while, do-while);
3) операторы переходов (goto, break, continue, return);
4) другие операторы (оператор "выражение", пустой, составной).

Один оператор может занимать одну или более строк. На одной строке можно записать несколько операторов, но по правилам структурировния рекомендуется каждый оператор начинать с новой строки.

Операторы в программе могут объединяться в составные (блоки) с помощью фигурных скобок. Любой оператор в программе м.б. помечен меткой, состоящей из имени и следующего за ним двоеточия

<идентификатор>:

Все операторы, кроме составных, заканчиваются ';'.

Рекомендуется записывать { - скобки } друг под другом.

Тело блока записывается с отступом (например на 2 позиции) от { }.

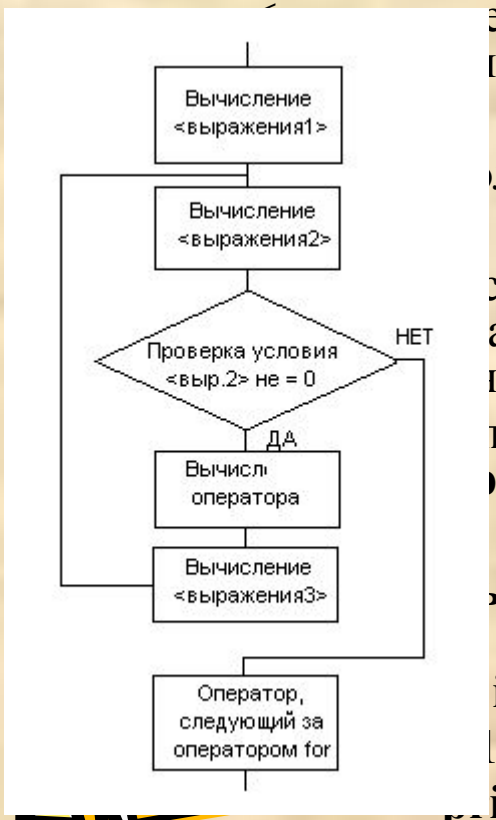
В операторе if, части if - else выравниваются по первой букве друг под другом. Тело оператора for выравнивается под именем параметра цикла.

ОПЕРАТОР ЦИКЛА for

for (<выражение1>;<выражение2>;<выражение3>) оператор;
инициализации условное итерации

<выражение1> - описывает инициализацию цикла и используется для установки начального(ых) значения(й) переменной(ых), управляющий(х) циклом.

<выражение2> - определяет условие, при котором оператор цикла будет выполнятся.



е3> - вычисляется после каждой
ременную цикла.

олнения оператора **for**.

словия всегда выполняется в начале цикла.
азу не выполниться, если условное
но.

иси оператора for
ратор>

ычислить значения квадратов чисел от 1до9

i;
;i<10,i++)
intf ("i=%d, квадрат=%d\n," i,i*i);}

Допускается использовать несколько переменных, управляющих циклом.

Пример: Программа для записи чисел в прямом и обратном порядке

```
#include<stdio.h>
void main()
 {int i,j;
   for (i=0,j=4; i<5; i++,j--)
   printf ("i=%d, j=%d \n", i,j); }
```

результаты:

i=0, j=4

i=1, j=3

i=2, j=2

i=3, j=1

i=4, j=0

Замечание: В операнде цикла выражение1 или 2 или 3 могут отсутствовать одновременно или может отсутствовать одно из них.

Пример: for (;i<n;i=i+2) <оператор> **//нач. значение - до цикла**

for(;;) оператор; **// бесконечный цикл**

for(<выражение1>;;<выражение3>) оператор;

Пропущенное условие2 по умолчанию считается истинным, оператор в этом случае организует бесконечный цикл. Выход из такого цикла возможен оператором break, который прерывает выполнение.

Замечание: может отсутствовать сам оператор, выполняемый в цикле.

for (i=0;i<100000;i++) ; м. использоваться для организации задержки выполнения.

ОПЕРАТОР ЦИКЛА с предусловием

while (<выражение>) <оператор>;

<оператор> м.б. либо простым, либо пустым, либо составным.

Если <выражение истинно>, то <оператор> выполняется до тех пор, пока <выражение> не станет ложным.

Если <выражение> ложно, управление передается оператору, следующему за циклом.

Оператор цикла

for(<выражение1>;<выражение2>,<выражение3>)<оператор>;

м.б. представлен оператором

while (<выражение2>){<оператор>;<выражение3>}

Пример: найти сумму цифр целого числа N

```
#include <stdio.h>
void main()
{ int N, S=0, ost;
printf("Введите N\n");
scanf ("%d",&N);
while(N)                    // (N!=0)
        { ost = N % 10;     // остаток
        N = N / 10;         // целая часть
        S += ost;           // сумма:   S=S+ost
        }           //составной оператор
printf("Сумма цифр = %d \n", S);
}
```

Составной оператор или **блок**

синтаксически эквивалентен одному оператору. Он имеет следующий формат:

{ [объявления;] // описания переменных и задание начальных значений оператор ;[оператор;] ...

}

В конце блока после правой } точка с запятой не ставится.

Все объявления, включенные в блок, должны быть в начале. Действие составного оператора заключается в последовательности выполнения составляющих его операторов. Основное назначение - группировать операторы в исполняемый модуль.

ОПЕРАТОР передачи управления goto

Формат оператора **goto <метка>; ...**

<метка> : <оператор>

Оператор goto выполняет безусловную передачу управления оператору с указанной меткой. Помеченный оператор должен находиться в той же функции, что и оператор goto. Метка должна быть уникальным идентификатором, за которым следует двоеточие.

Редко используется, если необходимо выйти из вложенных управляющих структур, например из двух или более циклов

Используя оператор goto можно передавать управление внутрь составного оператора (блока). Однако это делать следует осторожно, так как в этом случае обходится (пропускается) инициализация переменных, которая размещается в начале блока.

Пустой оператор

состоит только из ';' и никаких действий не выполняет. Обычно используется в операторах if, for, while, do-while, когда тело оператора отсутствует, хотя по синтаксису оператор необходим, а также, если требуется пометить фигурную скобку меткой. Синтаксис языка СИ требует, чтобы после метки обязательно следовал оператор, фигурная скобка же оператором не является.

Пример 1. С помощью оператора for и пустого оператора

```
for (i=0;i<100000;i++) ;
```

может быть организована задержка выполнения программы.

Пример 2. Структура программы, в которой имеется метка end:

```
main()                  // и три вложенных составных оператора
{...   for(...)
            {for (...)
                  { if (...) goto end; // оператор goto
                  else  ...
                  }
            }
       end ;}            // метка end и пустой оператор ;
```

ОПЕРАТОР ЦИКЛА с постусловием do-while

Применяется в тех случаях, когда тело цикла должно выполниться хотя бы один раз.

рмат оператора

ераторы> } while(<выражение>);

яются <операторы>, затем проверяется истинность
, если оно истинно (!=0), то <операторы> вновь
и вычисляется значение <выражения>. Если
ложно, то управление передается следующему за dowhile.

Пример: Протабулировать функцию y=sin x + cos x при 0 7, 0x<2 7p 0 с шагом p/8.

```
#include <stdio.h>
#include <math.h>
#define Pi 3.14
void main()
{ double  x=0, y;
do   y=sin(x) + cos(x);
printf("x=%lf\t y=%lf\n",x,y);
while(x>=2*Pi)
}
```

Чтобы прервать цикл до того, как условие станет ложным, можно использовать оператор break.

ОПЕРАТОР выхода BREAK ; (разрыв)

Обеспечивает прекращение выполнения самого внутреннего из объемлющих его операторов цикла for, do-while, while или оператора switch. После выполнения оператора break, управление передается оператору, следующему за прерванным.

Одно из назначений оператора - закончить выполнение цикла при присваивании некоторой переменной определенного значения.

Пример организации бесконечного цикла, пока не будут введены корректные значения длин сторон A, B, C для построения треугольника.

```
for (;;)     { scanf("%d %d %d", &A,&B, &C);
       if (A<B+C && B<C+A && C<A+B) break;
       else printf("A,B,C\n") }
```

Пример: программа подсчета числа различных элементов в массиве

```
#include <stdio.h>          // Каждый очередной элемент a[i]
void main ()                // сравнивается с последующими
{ static int a[] = {7,3,7,4,3,6}; // элементами массива. Если он
int i, j, m=6, k=0;         // не совпадает ни с одним из
for (i = 0; i<m-1; i++)     // этих элементов, в счетчик k
{                           // добавляется 1. В противном
for (j = i+1; j<m; j++)     // случае внутренний цикл преры-
if (a[i]==a[j])  break;     // вается оператором break и
if (j==m) k++;              // начинается новая итерация
}                           // для внешнего цикла
printf("%d различных элементов\n",k);
}                           // Ответ: 4 различных элемента
```

ОПЕРАТОР передачи управления continue

Прерывает выполнение тела цикла и передает управление на следующую итерацию. Формат оператора:

continue;

В операторах цикла **while, do** заново выполняется проверяющая часть, а в операторе **for** управление передается на вычисление выражения итерации (изменение параметра цикла).

Пример: программа подсчета положительных элементов в массиве

```
#include <stdio.h>
void main ()
{ static int a[] = {7,3,7,4,-3,-6};
int i, k, n = 6;
for (i = 0, k = 0; i<n; i++)
{ if (a[i] < 0) continue; //пропуск отрицательного числа
k++;
}
printf(" %d\n",k);
}
```

Оператор **continue** применим лишь к циклам, но не применим к переключателям. Как и оператор break, он относится только к самому внутреннему циклу (прерывает самый внутренний из объемлющих его циклов).

ОПЕРАТОР ВОЗВРАТА return

Завершает выполнение текущей функции и возвращает управление в вызывающую функцию в точку, непосредственно следующую за вызовом.

return[<выражение>] ;

Выражение передает свое значение в вызывающую функцию. Выражение может отсутствовать, в этом случае возвращаемое функцией значение не определено. Выражение может заключаться в круглые скобки.

Если в вызываемой функции, оператор **return** отсутствует, то управление автоматически передается в вызывающую функцию после выполнения последнего оператора функции. Возвращаемое функцией значение в этом случае не определено. Если функция не возвращает значения, ее следует объявить типом **void** (пустой).

Таким образом оператор return используется

1) для немедленного выхода из функции

Например:

```
void print(char x)
{ if (x==0) {printf("аргумент=0\n"); return;}
printf("Введен аргумент %c\n",x);
}      // return используется для выхода из функции, если аргумент равен нулю
```

2) если функция должна возвращать значение

Например:

```
func sum(int a, int b);    // a, b – формальные аргументы
{  return (a + b);
}
```

Работа с файлами, структуры,
Возврат из функции нескольких значений
См. в файле
лекCPPфайлСтрукт.doc
или учебном пособии Шафеевой О.П.
«Технологии программирования», 2007