Лицей при СПбГУТ

А.В. Красов

Программирование на языке Си

Часть первая

Основные конструкции языка

Издание 3, доработанное

Санкт–Петербург

1999 г.

А.В. Красов

Программирование на языке Си. Первая часть, Основные конструкции языка. СПб.: Лицей СПбГУТ 1998, -44 с.

Предлагаемое Вашему вниманию пособие посвящено обучению программированию на языке Си для IBM PC и совместимых с ними компьютеров.

Данный курс читается автором пособия с 1989 года для подшефных школьников и студентов СПбГУТ, а также СПбГЭТУ.

Характерным отличительным моментом методики рассмотрения материала является ориентация на наглядное графическое представления материала. Тем самым акцентируется внимание в первую очередь на алгоритм, суть задачи, и лишь затем реализация. Ориентация на выбранный стиль разработки программ сразу приучает к определенным подходам к программированию и оформлению текста своих программ, это принесет несомненные преимущества при разработке сложных приложений.

Большое количество примеров и наглядных задач делает курс интересным и занимательным. Приведенный план занятий позволяет оценивать темпы изучения материала и вносит элементы соревнования.

Пособие предназначено для учащихся лицея, базовых школ университета, студентов младших курсов СПбГУТ, и всех тех кто интересуется программированием.

Замечания, предложения о контактах можно направлять по адресу
KRASOV@mail.wplus.net

Пособие предназначено для учащихся лицея, базовых школ, студентов младших курсов СПбГУТ и всех тех, кто интересуется программированием.

Рассмотрены основные конструкции языка Си и их применение в разработке прикладных программ.

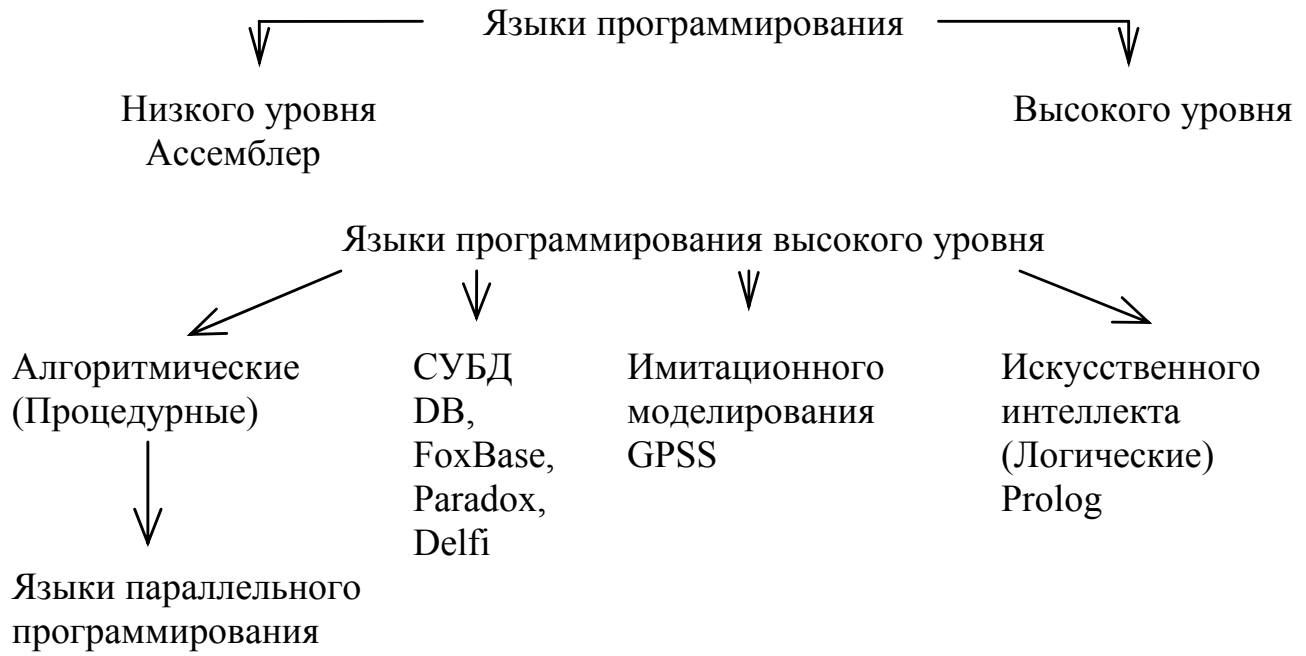
Изложение материалов иллюстрируется графическими блок-схемами.

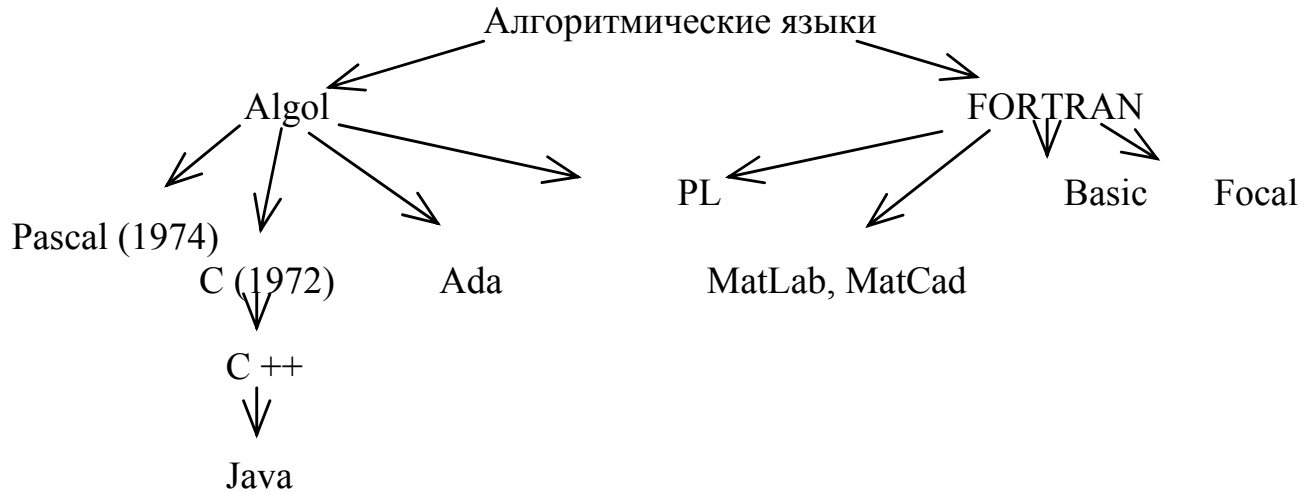
Рецензент:

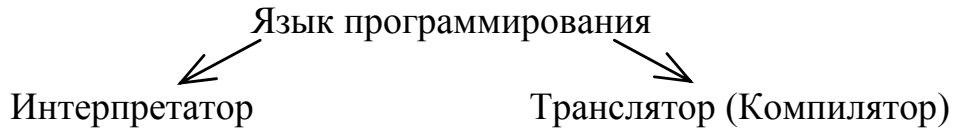
методист адмиралтейского района Н.И. Жиганова.

Часть 1
Классификация языков программирования

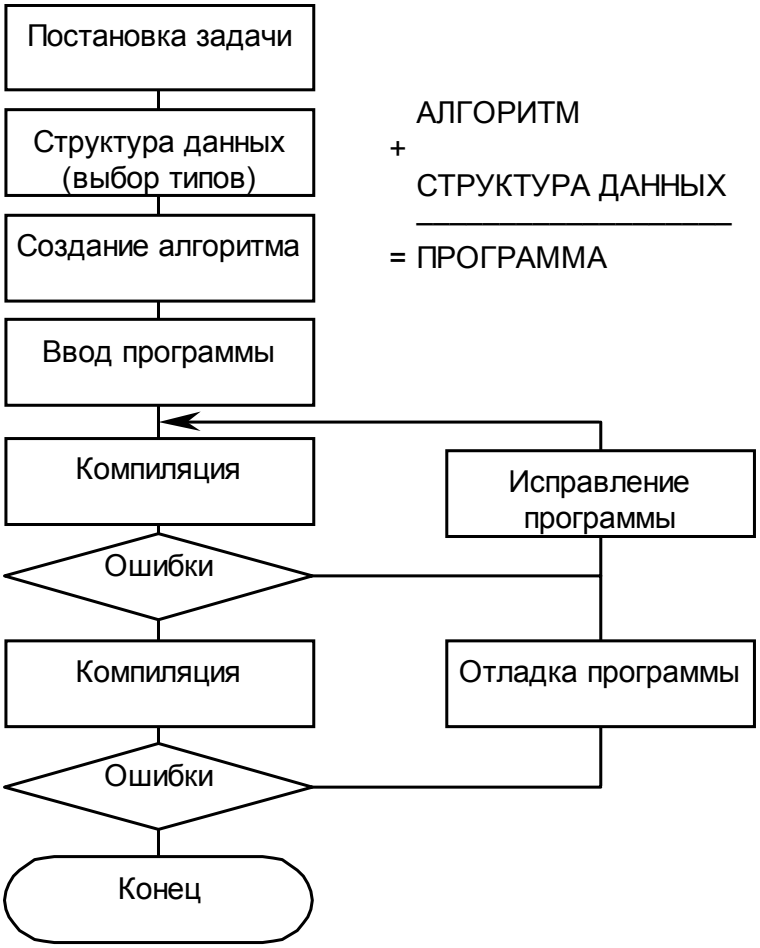
Языки программирования делятся по ресурсам, предоставляемым пользователю, классу решаемых задач, способу организации системы исполнения.







Основные этапы создания программы



Постановка задачи - Содержательная постановка задачи, определение конечных целей решения;

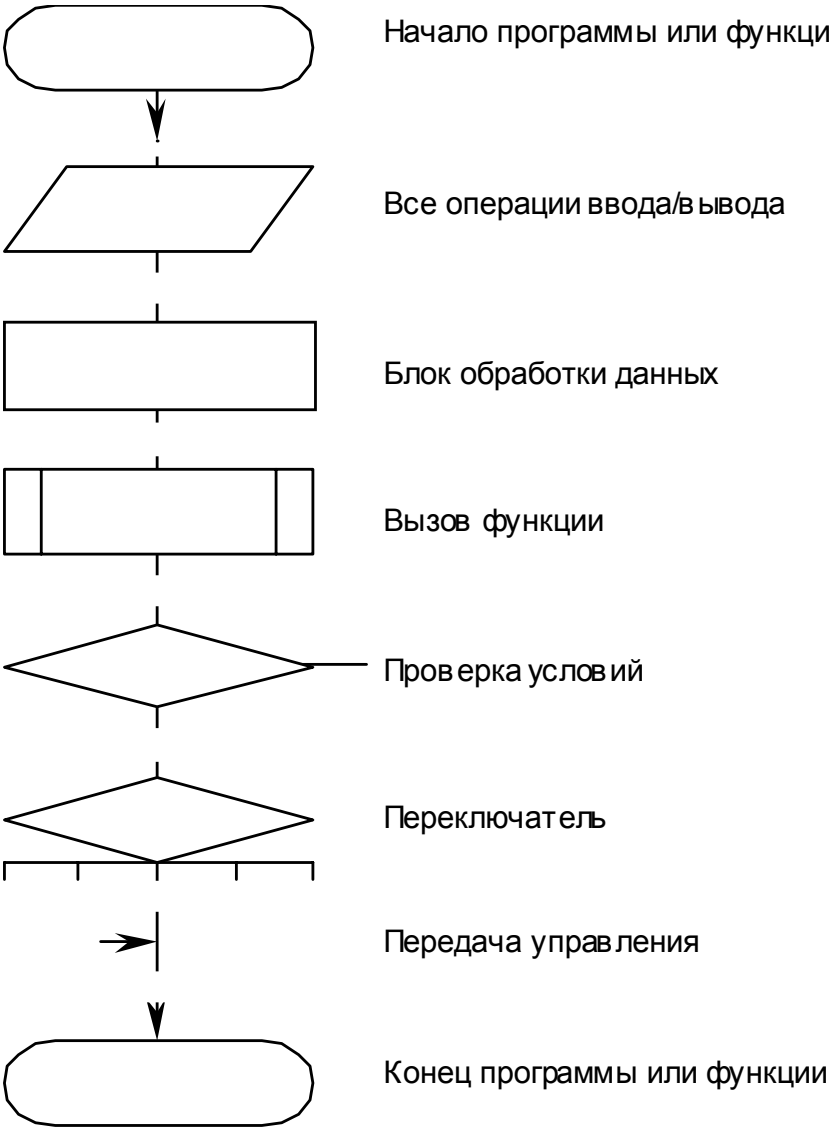
Алгоритм - Точно определенная последовательность действий, необходимых для решения задачи;

Блок-схема - Графическая форма записи алгоритма;

Программа - Последовательность команд ЭВМ, реализующих алгоритм и данные, необходимые для его выполнения;

- Форма представления алгоритма;
- Алгоритм + Структура данных = ПРОГРАММА.

Типовые блоки для записи алгоритма.



Язык программирования Си

Создан в 1972 г. Деннисом Ритчи. Его характеризует:

- Большая распространенность;
- Скорость работы написанных программ соизмерима с языком ассемблера, при этом программы более наглядны и просты;
- Переносимость программ;
- Совместимость с большинством других средств программирования;
- Легкий доступ к аппаратным средствам;
- Отсутствие ограничений на режимы работы с памятью;
- Включение языка ассемблера;
- Отсутствие жесткого контроля над действиями программиста.

На Си написаны: Операционные системы Windows, UNIX; языки программирования для UNIX Паскаль, Фортран, АПЛ, Лисп; лучшая игрушка 1993 г. Eye of Beholder; и многие другие программы.

Интегрированная среда

Интегрированная среда включает в себя:
- Редактор программ;
- Транслятор, Компоновщик;
- Отладчик;
- Справочник по системе программирования; примеры на основные функции.

Все действия по написанию и отладки программ можно выполнить, не покидая интегрированной среды.

Borland C, версия 3.1
Команды интегрированной среды

Все действия можно выполнить либо выбрав пункт меню, либо нажав установленные функциональные клавиши.
'+' - Одновременное нажатие нескольких клавиш.
',' - последовательное нажатие клавиш.

F1	- Помощь
Cntrl+F1	- Помощь по конкретному слову
F10	- Выход в меню
Alt+X	- Выход в DOS
F10,'F','N'	- Создание нового файла
F3	- Загрузка существующего файла
F2	- Запись редактируемого файла на диск
Ctrl+'K','B'	- Отметить начало блока
Ctrl+'K','E'	- Отметить конец блока
Ctrl+'K','C'	- Копировать отмеченный блок в указанное место
Ctrl+'Y'	- Удалить строку
F6	- Переход между окнами
Alt+<Номер>	- Переход к окну с номером
Alt+F3	- Закрытие окна
F9	- Трансляция программы (Проверить ошибки)
Cntrl+F9	- Транслировать и запустить на выполнение
F7	- Пошаговое выполнение программы
F4	- Выполнить до указанного места
Cntrl+F7	- Контроль значений переменных
Alt+F5	- Показать экран

Первая программа

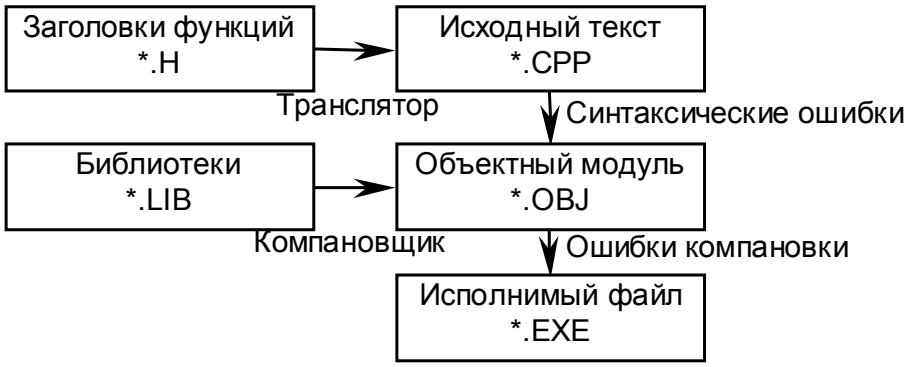
```
#include <stdio.h>
void main(void)
{
/* Комментарий, все что заключено между этими скобками не транслируется */
        printf("\n Привет !") ;
} /* Кон. Main() */
```

Комментарии к программе

#include <stdio.h> - Описание заголовков функций ввода вывода.

```
void main(void)                - Описание основной (обязательной) функции
  |     |     |
  |     |   Функция не имеет аргументов
  |   Имя основной функции
 Функция не возвращает ни какого значения
{                    - Начало описания функции
printf() ;           - Вызов функции вывода на экран
}                    - Завершение описания функции
```

Компиляция программы



Структура программы.

Заголовки стандартных функций
Описание своих типов *
Описание глобальных переменных *
Описание своих функций *
Описание функции main

* - Присутствует необязательно.

Описание функции

```
< Тип функции > < Имя функции >( < аргументы > )
{
        < Внутренние переменные >
        < Операторы >
 } /* Кон. Функции */
```

Пример:

```
int Abs(int x)
{
        int i ;
        if(x<0) return(-x) ;
        return(x) ;
 } /* Кон.Abs() */
 /* Все что заключено в этих "скобках" является комментарием */
```

Основные типы данных

Тип	Размер	Диапазон значений	Тип	Форматы
Void			Пустой.	
char	1 байт	-128..127	Символьный.	%c,%d,%s
unsigned char	1 байт	0..255	Беззнаковый симв.	%c,%d
int	2 байта	-32768..32767	Целый.	%c,%d
unsigned int	2 байта	0..65535	Беззнаковый целый	%c,%d
float	4 байта	Формат IEEE	Вещественный.	%f,%g
double	8 байт	Формат IEEE	Вещ. с двойн.точ.	%f,%g

Описание переменных

<Имя типа> <Имя переменной>, <Имя переменной>, ... <Имя переменной> ;

Допустимые имена переменных и функций

Имя переменных - последовательность из одной или более латинских букв, цифр и символов подчеркивания, которая начинается с буквы или с символа подчеркивания. Максимальное число символов в имени 32.
Строчные и заглавные буквы - РАЗНЫЕ СИМВОЛЫ.
Пример: char Ch,c ; int i,j,k ; float x,A,a ; double d ;

Операции присвоения

< Имя переменной > = < число или выражение> ;
^ ^

Сокращенные формы:

i=i+1 ;	i++ ;
i=i-1 ;	i-- ;
k=k*2 ;	k*=2 ;
k=k/3 ;	k/=3 ;

Арифметические операции

+ - сложение
- - вычитание
* - умножение
/ - деление
% - остаток от деления

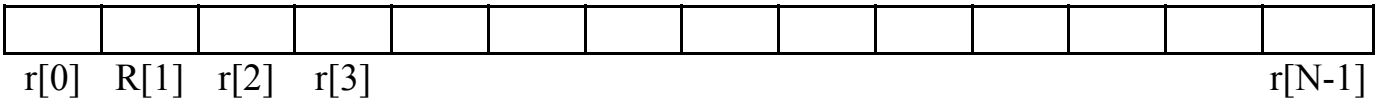
Преобразование результатов

	Операнд		
Операнд		Integer	float
	Integer	Integer	float
	float	Float	float

3 / 2 = 1

3.0/ 2 = 1.5

<Тип> <Имя>[<Кол-во элементов>] , ... ;
Пример:char St[80], s[23] ; float r[14] ;



Нумерация элементов в массиве (строке) начинается с 0.

Вывод на экран printf()

printf("<Строка>"[,<Значение>,...,<Значение>]) ;
Примеры:
printf("Привет !") ; - вывод строки на экран
printf("\nПривет!") ; - символ \ используется для задания управляющих символов:

\n - перевод строки
\r - возврат каретки
\\ - символ \
\' - символ '
\" - символ "
\0 - символ конца строки

printf("\n a=%f d=%d" ,a,d) ; - вывод значений переменных

%f,%d,%c,%s - форматы вывода переменных, вместо них при выполнении будет выведено значение указанное после символа ".

Точное задание форматов ввода вывода

%<кол-во поз.>.<поз. посл. тчк.>f	%7.2f
%<кол-во поз.>d	%3d, %03d
%<кол-во поз.>s	%20s

Ввод с клавиатуры scanf()

scanf("<формат>", & <Имя переменной>)

Символ & - означает операцию получить адрес переменной. Спецификация формата должна соответствовать типу читаемой переменной.

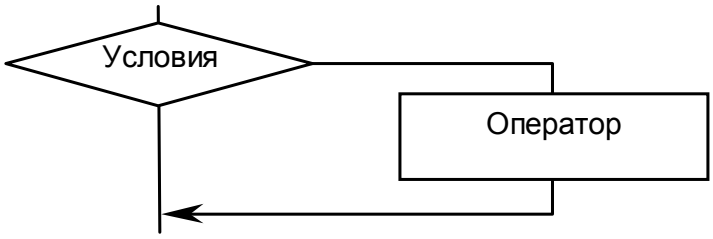
Функция scanf читает, по указанному формату, значение с клавиатуры и помещает его по заданному адресу.

Пример:

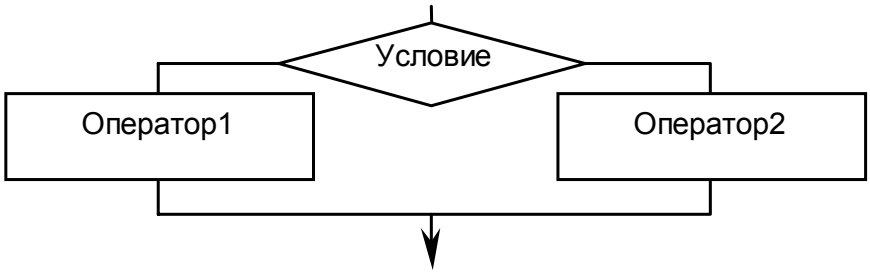
```
scanf("%d",&i) ;
scanf("%f",&r) ;
scanf("%s",&st[0]) ; /* Правильно, но можно лучше */
scanf("%20s",st) ; /* Имя массива (строки) - адрес первого элемента, (номер в массиве
                      - смещение от начала) */
```

Условный оператор if, if...else

if(<Условие>) <Оператор>



if(<Условие>) <Оператор 1>
else <Оператор 2>



Составной оператор

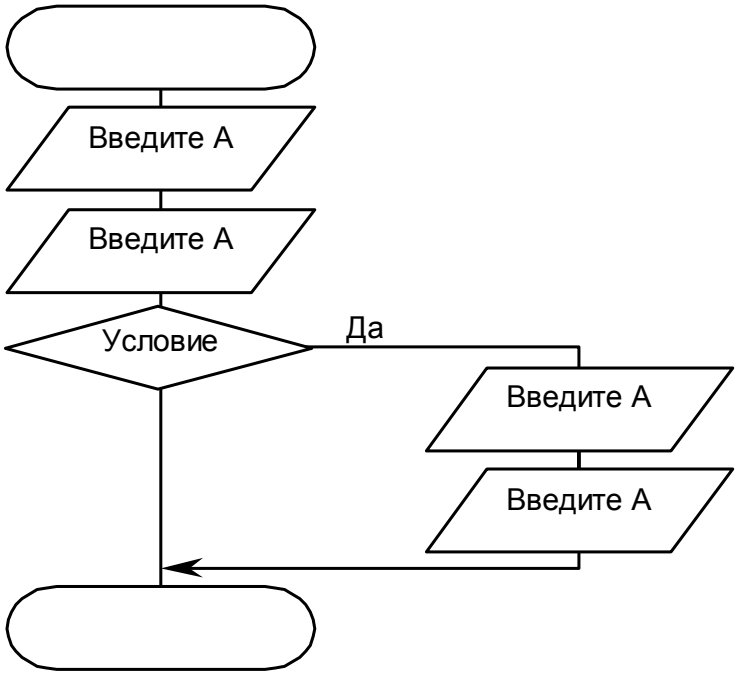
Составной оператор применяется когда требуется несколько операторов, а по синтаксису можно поставить только один. Например if, for, while, итп. После составного оператора; не ставятся.

```
{
        < Оператор 1 > ;
         < Оператор N> ;
}
```

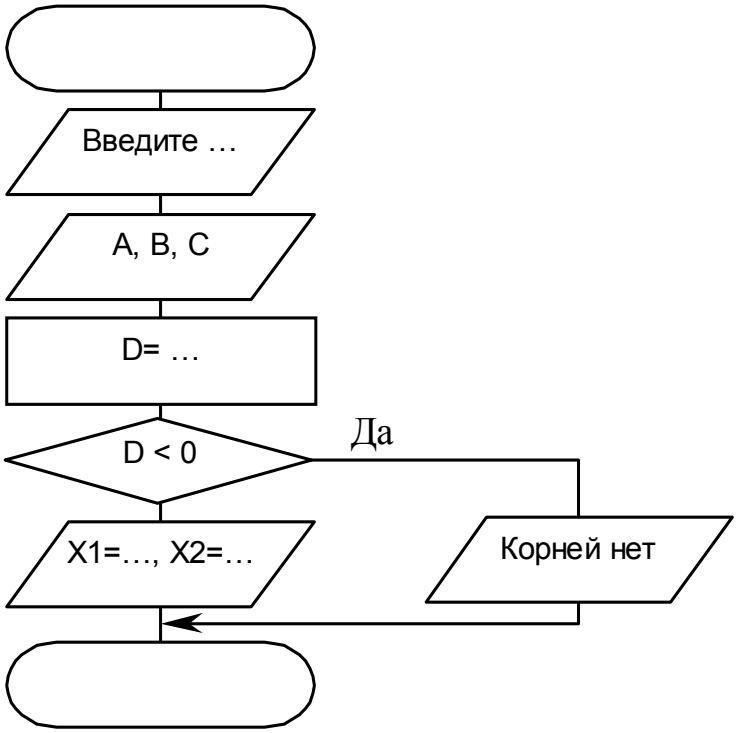
Допускается любой уровень вложенности составного оператора.

Вычисление модуля

```
#include <stdio.h>
void main(void)
{
        float A ;
        printf("\n Введите A:") ;

        scanf("%f", &A ) ;


        if( A < 0 )
        {
                printf("\n A < 0 ") ;

                A = - A ;

        }/* кон. if() */
}/* кон. main() */
```



*Вычисление корней уравнения a*x^2+b*x+c=0*



Математические функции math.h

Для использования этих функций необходимо подключить модуль math.h
#include <math.h> в начале программы.

k=abs(j)	Модуль целого числа	y=tan(x) -	Тангенс
y=sqrt(x)	Квадратный корень	y=pow(x,n) -	x в степени n,
y=fabs(x)	Модуль вещ. числа	n может быть дробным	
y=sin(x)	Синус	y=log(x)	Натуральный логарифм
y=cos(x)	Косинус	y=exp(x)	Экспанента
y=log10(x)	Десятичный логарифм		

Задания условия

Истина (Да) 1			Ложь (Нет) 0
Равно	==		a==b
Не равно	!=		ch !=27
		>, <, >=, <=	a>10

Сложные условия

Инверсия	!	(Противопол. знач.)	!(a>b) /* a<=b */
Логическое И	&&	(И то и другое)	(5<a)&&(a<8)
Логическое ИЛИ	||	(Либо то либо другое)	(ch==27)||(a=10)

Логическое ИЛИ A || B

		A	
		Ист.	Ложь
B	Ист.	Ист.	Ист.
	Ложь	Ист.	Ложь

Логическое И A && B

		A	
		Ист.	Ложь
B	Ист.	Ист.	Ложь
	Ложь	Ложь	Ложь

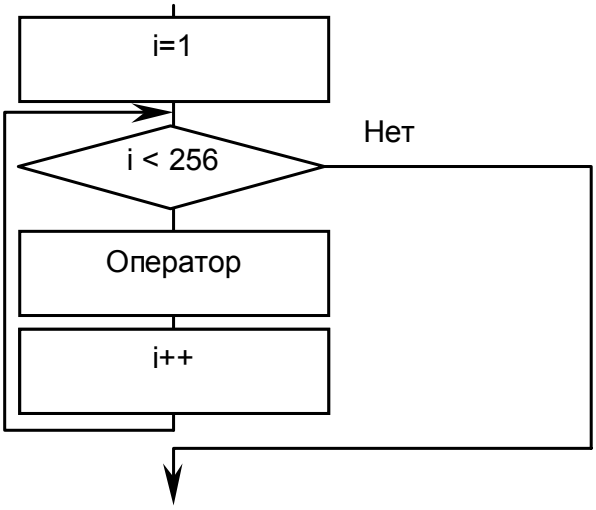
Внимание: a=b - ИСТИНА если b !=0 иначе ЛОЖЬ

Цикл for

Цикл for, как правило, используется для организации известного числа повторений какого либо действия.

for(<Нач.> ; <Конец> ; <Продв. по циклу>) <Ореператор> ;
Пример:
for(i=1;i<256;i++) printf("\n %c - символ с номером %d",i,i) ;

<Нач.> - Задание начального значения переменной цикла
<Конец> - Цикл выполняется пока выполняется условие
<Продв. по циклу> - Любое изменение переменной цикла
<Оператор> - Оператор или составной оператор, повторяемый указанное количество раз



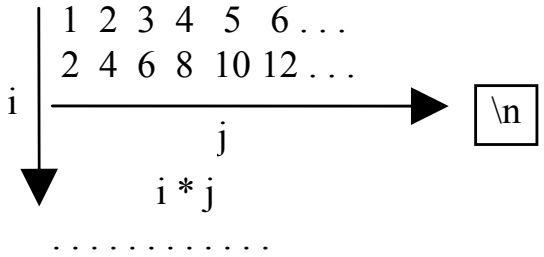
Пример:
for(i=10;i>0;i--) printf("\n i=%d",i) ;
for(i=1;i<20;i=i*2) printf("\n i=%d",i) ;

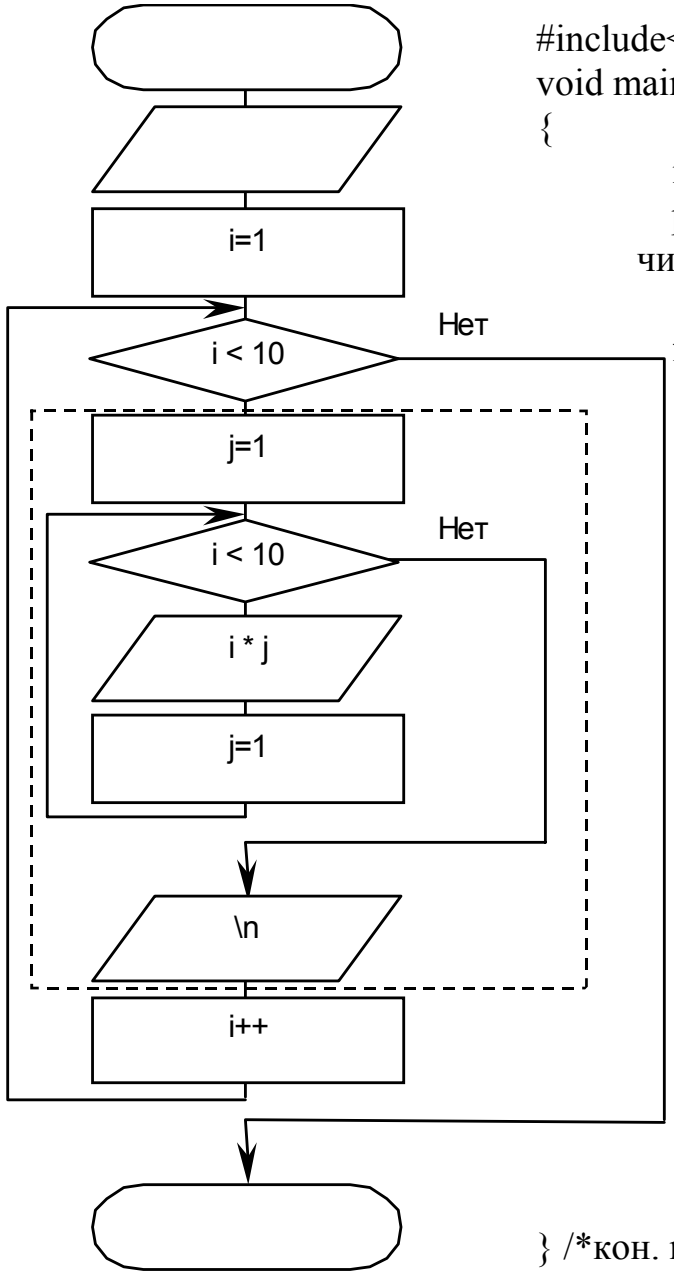
Генератор случайных чисел
Модуль stdlib.h и time.h

randomize() - Инициализировать генератор случайных чисел
i=random(int N) - Случайное число в диапазоне [0..N)
Если N=4; i может принимать значения: 0,1,2,3 Пример:
randomize() ;
for(i=0;i<10;i++) printf("\n Случайное число: %d",random(5)) ;

Инициализация генератора случайных чисел randomize() выполняется один раз в начале программы и привязывает генератор к значениям в рабочих регистрах системного таймера компьютера.

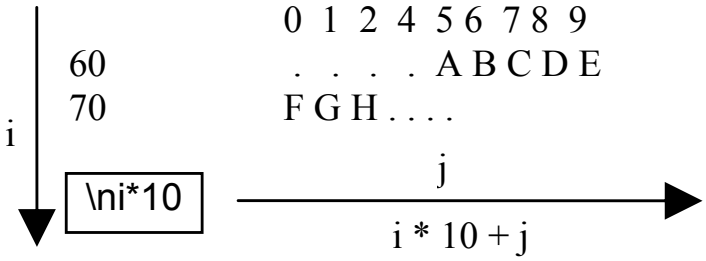
Таблица умножения

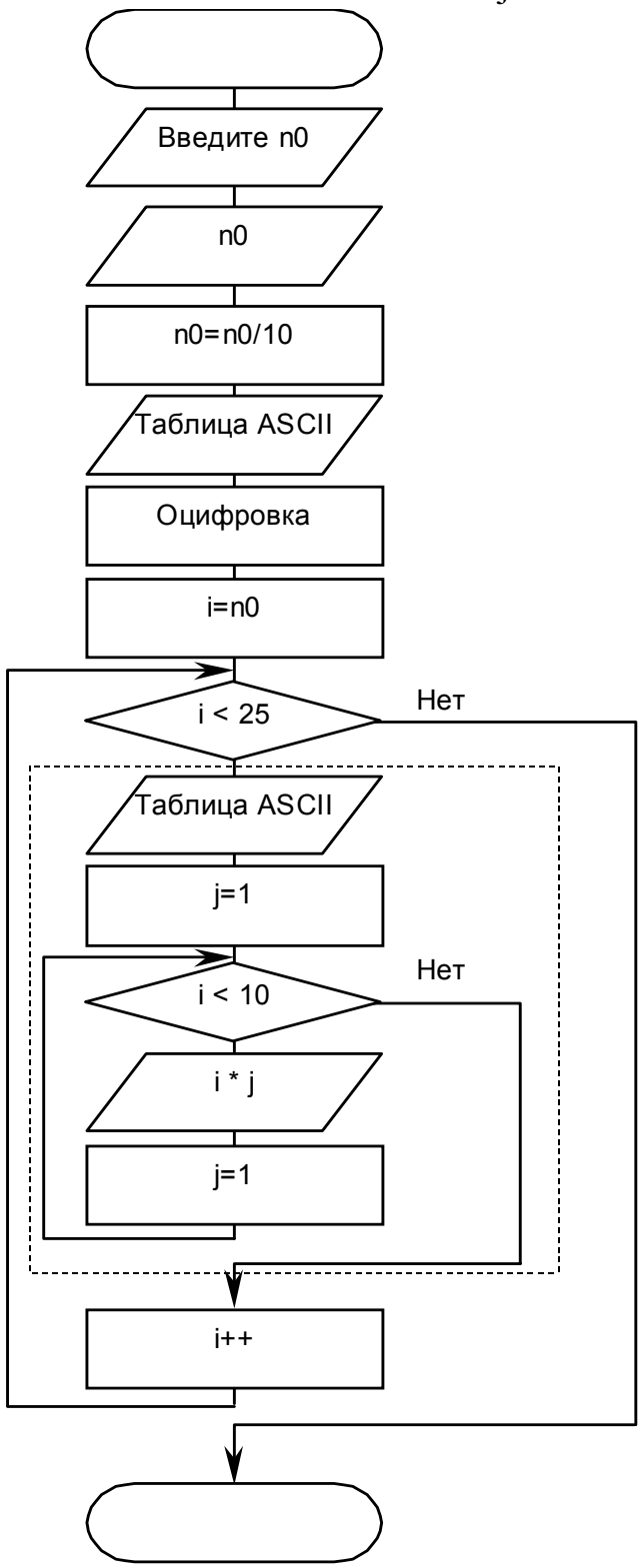





```
#include<stdio.h>
void main(void)
{
        int i,j;
        printf("\nТаблица умножения целых
    чисел\n");

        for(i=1;i<10;i++)
        {

            for(j=1;j<10;j++)

                printf("%2d",i*j);

            printf("\n");

        } /*кон. for(i=…)*/
} /*кон. main()*/
```

Таблица всех символов ПК (ASCII)





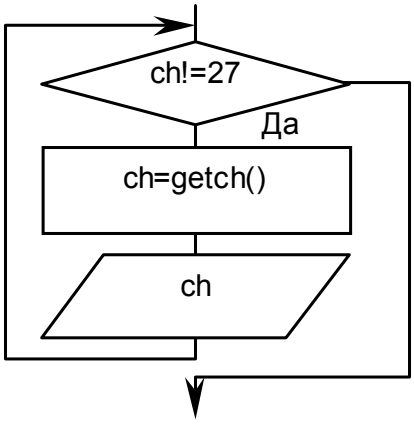
Первые символы ASCII таблицы содержат управляющие символы, при выводе таблицы их лучше не выводить.

Для удобства вывода удобнее выводить не с символа с номером n0, а с его десятка.

Написать заголовок таблицы.
В цикле расставить номера. Перед циклом рекомендуется сделать отступ из пробелов на ширину колонки номеров десятков.

Цикл while

while(<Условие>) <Оператор> ;
Цикл выполняется пока выполняется условие.
Пример:

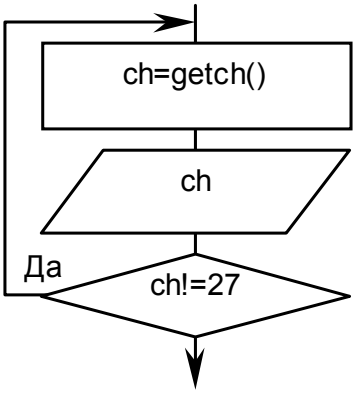



```
while (ch!=27)
{

        ch=getch() ;

        printf("\n Нажат символ %c с кодом %d",ch,ch);

} /* кон. while */
```

Цикл do ... while

do <Оператор> while(<Условие>) ;

Предыдущий пример может ни разу ни выполниться, если ch=27, до цикла. В цикле do ... while проверка происходит в конце цикла и этого не произойдет.




```
ch=27 ;
do
{

        ch=getch() ;

        printf("\n Нажат символ \'%1c\' с кодом %d",ch,ch);

} while(ch!=27) ;
```

Заключение.

Если в цикле необходимо использовать более одного оператора то используется составной оператор. Ограничений на вложенность нет.

При необходимости прекратить выполнение цикла, или пропустить часть текущего прохода можно использовать команды break и continue.

Работа с экраном conio.h

clrscr()	-	Очистка экрана (цветом фона)
textcolor(<>)	-	Цвет вывода символов
textbackgroubd(<>)	-	Цвет фона под выводимыми символами
cprintf(...)	-	Вывод на экран (цветной)
cscanf(...)	-	Чтение с клавиатуры (цветное)
gotoxy(<X>,<Y>)	-	Позиционирование по экрану
getch()	-	Читает символ без отображения

Размер экрана

(1, 1) (80, 1)

(1,25)
(80,25)

Цвета

0	-	BLACK	(черный)	8	-	DARKGRAY	(тем.-серый)
1	-	BLUE	(синий)	9	-	LIGHTBLUE	(св.-голубой)
2	-	GREEN	(зеленый)	10	-	LIGHTGREEN	(св.-зеленый)
3	-	CYAN	(бирюзовый)	11	-	LIGHTCYAN	(св.-бирюзовый)
4	-	RED	(красный)	12	-	LIGHTRED	(св.-красный)
5	-	MAGENTA	(малиновый)	13	-	LIGHTMAGENTA	(св.-малиновый)
6	-	BROWN	(коричневый)	14	-	YELLOW	(желтый)
7	-	LIGHTGRAY	(св.-серый)	15	-	WHITE	(белый)

Пример:

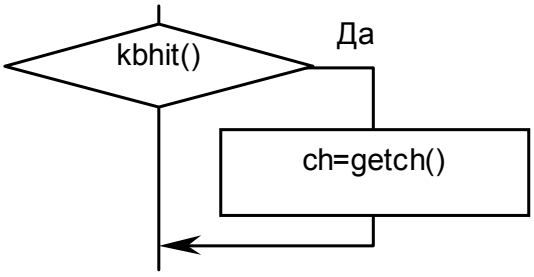
```
#include<conio.h>
void main(void)
{
        int i ;
        clrscr() ;
        for(i=0;i<16;i++)
        {
                gotoxy(5,5+i) ;
                cprintf("Цвет N~%2d",i) ;
```

```
        } /* Кон. for */
        for(i=0;i<16;i++)
        {
                textbackground(i) ;
                gotoxy(20,5+i) ;
                cprintf(" ") ;
        }/* кон. for */
} /* кон. main() */
```

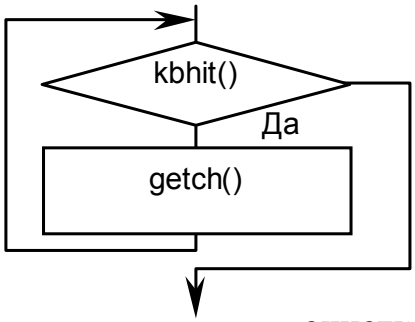
Проверка нажата ли клавиша
Модуль stdlib.h

i=kbhit() - Возвращает 1 (ИСТИНА) если в буфере клавиатуры есть нажатые клавиши, но не читает их; иначе 0. Необходимость в такой функции вызвана тем что, функция getch() и
некоторые другие функции ввода останавливают выполнение программы.
Функция kbhit() позволяет избежать лишних остановов программы. Например, в игрушке, если не нажата клавиша продолжать движение в старом направлении.

Буфер клавиатуры может запомнить много нажатых символов, в некоторых случаях его целесообразно очистить. Это тоже можно сделать с помощью функций kbhit() и getch().



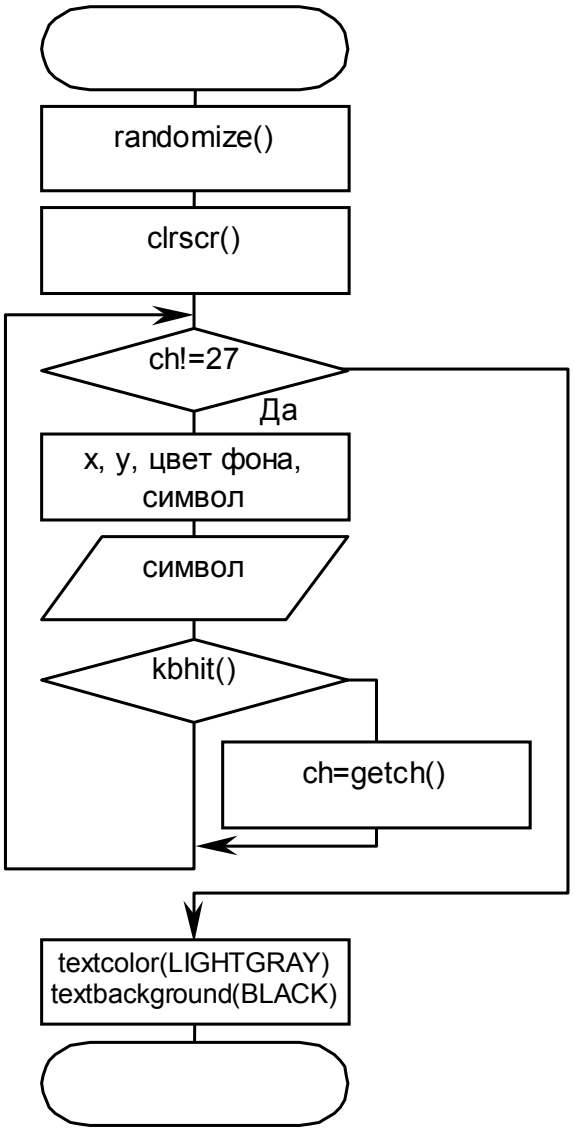

```
/* если в буфере клавиатуры есть нажатый
символ, то прочитать его */
if( kbhit() )

        ch = getch() ;
```




```
/* Очистка буфера клавиатуры */
while( kbhit() )

        getch();
```

/* Какие конкретно символы читаются при очистке буфера клавиатуры нам не важно */

Программа заполнения экрана цветными символами

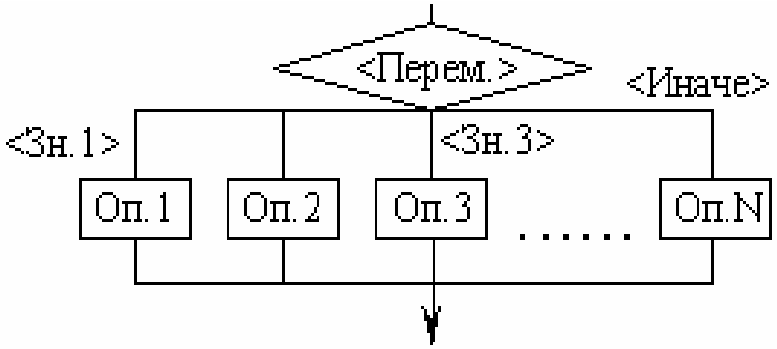


Инициализировать генератор случайных чисел

Установить случайные координаты, цвет фона и цвет символа
Вывести случайный символ код в диапазоне [50 .. 250]

Восстановить первоначальный цвет и фон отображения

Переключатель switch ()




```
switch ( <Перем.> )
{
   case <Зн1>:<Операторы> ;
                    break ;
   case <Зн2>:<Операторы> ;
                    break ;
   . . . . .

   efault : <Операторы> ;]
}
```

<Перем.> - Переменная или выражение перечисляемого типа (integer, char, и т.п.)
<Зн.> - Допустимые значения
break - Прервать выполнение switch. Если его нет, выполняются следующие строки программы
Default - Если ни одно из перечисленных.[ ]- Необязательный параметр.
<Опер-ры> - Группа операторов, составной оператор в этом случае использовать необязательно

Пример:

```
ch=getch();
switch (ch)
{
        case 'A':
        case 'B': ;
        case 'C':printf("\n Нажаты заглавные символы") ;
                printf("\n Нажат символ %c ",ch) ;
                break ;
        case 13 :printf("\n Нажата клавиша \'Enter\'") ;
                break ;
        case 27 :printf("\n Нажата клавиша \'Esc\'") ;
                break ;
 }/* кон. switch */
```

Движение символа по экрану

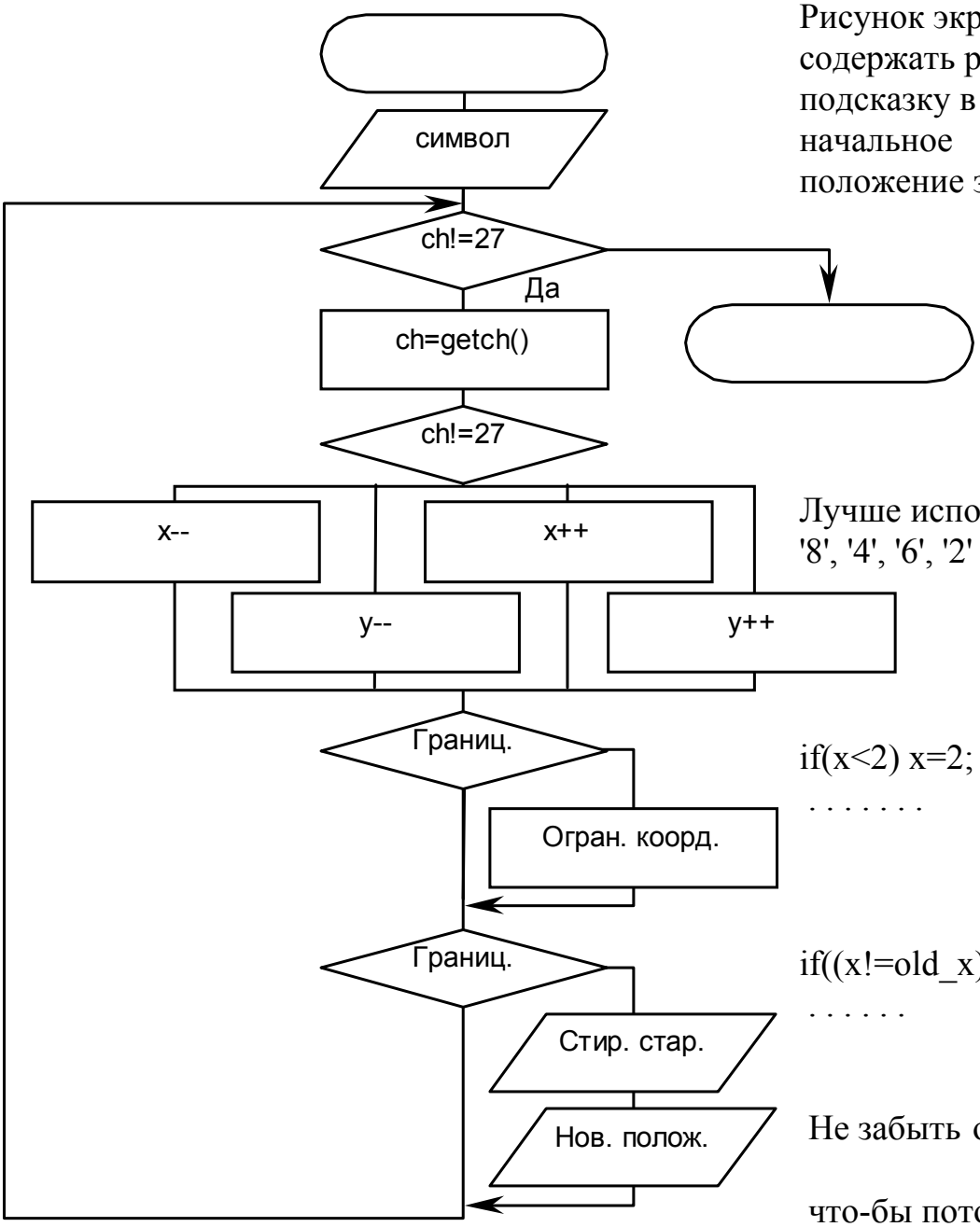


Рисунок экрана должен содержать рамку, подсказку в нижней строке и начальное положение звездочки.

Лучше использовать знач. '8', '4', '6', '2'

if(x<2) x=2;
. . . . . .

if((x!=old_x)||(y!=old_y))
. . . . . .

Не забыть old_x=x ;
old_y=y ;
что-бы потом стирать уже в новом месте

Рекомендуется использовать переменные:

int x,y; - Текущее положение символа на экране

int old_x,old_y; - Старое положение символа на экране

Рисование символа:
```
gotoxy(x,y);
cprintf("*"); /* функция printf() не допускает позиционирования
                 по экрану */
```

Стирание символа:
```
gotoxy(old_x,old_y);
cprintf(" ");
```

Функции

Довольно часто программа содержит повторяющиеся в нескольких местах группы одинаковых команд, такие фрагменты целесообразно оформить в виде функций.

Каждая функция описывается следующим шаблоном:

```
<Тип функции> <Имя функции>(<Аргументы>)
{
        <Внутренние переменные> ;
        <Операторы> ;
} /* Кон. Функции */
```

Пример для предыдущей программы:

```
void PutC(int x,int y,char ch)
{
        gotoxy(x,y) ;
        cprintf("%1c",ch) ;
} /* Кон. PutC() */
```

Пример вызова:

```
if((old_x!=x)||(old_y!=y))
{
        PutC(x,y,'*') ;
        PutC(old_x,old_y,' ') ;
        old_x=x ;
        old_y=y ;
}
```

Исходный текст функции должен быть описан до места (текста функции), где он используется.

Собственные функции можно объединять в свои библиотеки, и присоединять их к программе на этапе компановки. Это увеличит скорость работы транслятора, часть текста не нужно будет компилировать. Библиотеки создаются из объектных (*.obj) файлов, полученных после трансляции отдельно текстов функций. Кроме этого создается файл с заголовками функций, аналогично стандартным, но подключается он в основную программу не в <> а в "". Пример: #include "window.h"

Передача параметров

В Си параметры в функцию передаются значением. Это значит, что их в функции нельзя изменить. Для описанного выше примера:

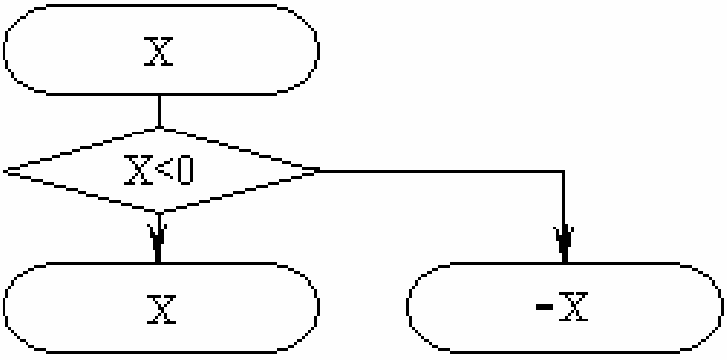
```
void PutC(int x,int y,chat ch)
{
x=20 ; /* - ОШИБКА, здесь x не переменная, а конкретное значение заданное при обращении к функции */
} /* Кон. PutC() */
```

При обращении к функции тоже может стоять не переменная PutC(10,10,'*') и пытаться число 10 на другое невозможно.

Возвращение значений

Функция возвращает значение соответствующее ее типу, указанному в описании:
r=sin(0.12) ; r - вещественное число.

Функция может иметь несколько точек выхода (возврата значений). Завершение функции выполняет команда return(<Значение или выражение>) ;



Например функция модуля:

```
float Absffloat x);
{
        if(x<0)
                return (-x);
        return (x);
} /* Кон. Abs() */
```

Факториал

Факториал - математическая функция определенная для целых переменных. Обозначается знаком '!'.

>0 - не существует.

0! = 1
n! = 1*2*3*....*(n-1)*n ;
n! = n*(n-1)! ;

Пример: 5!=1*2*3*4*5=120

Программа вычисления факториала

```
#include <stdio.h>
int Fac(int X)
{
        . . . .
} /* кон. Fac() */
int main(void)
{
        int x1,x2 ;
        printf("\n Введите два целых числа ") ;
        scanf("%d",&x1) ;
        scanf("%d",&x2) ;
        if((x1<0)||(x2<0))
        {
                printf("\n Ошибка ! факториал для x<0 не существует") ;
                return(0) ;
        } /* кон. if() */
        printf("\n %d!=%d \n %d!=%d",x1,Fac(x1),x2,Fac(x2)) ;
        return(1) ;
} /* кон. main() */
```

В данном примере функция main() имеет две точки выхода, конкретное значение не актуально.

Функцию Fac() можно реализовать по разному. Самое простое через цикл, однако более красиво вариант с рекурсией.

```
int Fac(int X)
{
if(X==0) return(1) ; /* В принципе можно записать if(!X) */
return(Fac(X-1)*X) ;
} /* кон. Fac() */
```

Избежать проблем с отрицательными значениями можно если использовать тип unsigned int в место int. Проблема переполнения решается описанием типа функции как float Fac().

Возвращение нескольких значений

В этом случае в функцию передается не значение переменной, а ее адрес по которому заносятся возвращаемые данные, аналогично функции scanf(), рассмотренной ранее.

Описание	Переменная	Указатель	Описание
Описание переменной Присвоение значения Ввод с клавиатуры Адрес переменной	float A ; A = 5 ; scanf("%f",&A); $A	flaot *B ; *B = 5 ; scanf("%f",B); B B=&A ;	Описание указателя

Указатель B указывает на переменную A, это одна переменная.
*(B+1) - Следующая за *B вещественная переменная (элемент массива).

Педыдущий пример:

```
void ABS(float *X, int *i)
{
        if(*X<0) *X=-(*X) ;
        *i=10 ;
} /* Кон. ABS() */
```

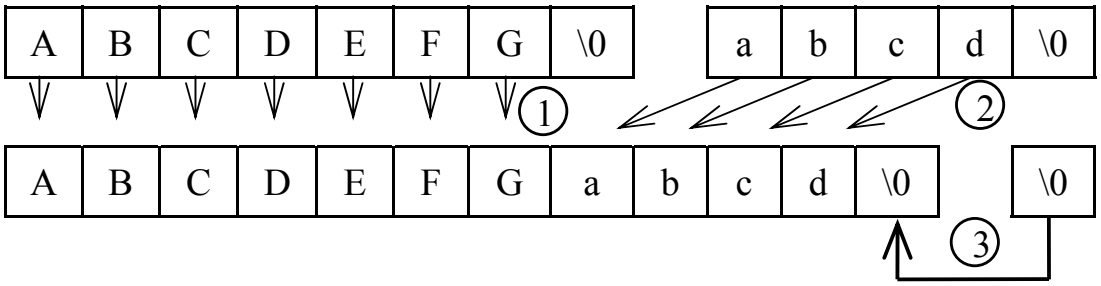
Но при вызове уже нельзя написать ABS(-5) - это не адрес, можно только так:

```
float t int i ; t=-5 ;
ABS(&t,&i) ;
```

Слияние двух строк

В этом примере в функцию CopyStr() указателями передаются два массива. Си не осуществляет контроля за размером массивов, это необходимо учитывать в программе. Признак конца строки символ '\0'.

```
#include <stdio.h>
void CopyStr(char *st1,char *st2,char *Rez)
{
        Rez[i]=st1[i];
        . . .  /* Символ '\0' обозначает конец строки.*/
} /* Кон. CopyStr() */
void main(void)
{
        chat s1[20],s2[20],rezultat[40] ;
        printf("\n Введите две строки") ;
        scatf("%20s",s1) ;
        scanf("%20s",s2) ;
        CopyStr(s1,s2,resultat) ;
        printf("\n Результат : %s",resultat) ;
} /* кон. main() */
```



Работа со строками

Си имеет большой набор стандартных функций для работы со строками. Для работы с ними необходимо подключить файл string.h.

strcpy(char *st1,char *st2) - строка st2 копируется в st1
strcat(char *st1,char *st2) - к строке st1 добавляется st2
n=strlen(char *st) - возвращается длинна строки
strlwr() - преобразует все символы строки в заглавные
strcmp() - сравнение строк

Структуры

Кроме базовых типов данных в Си существует возможность конструировать свои типы данных, ориентируя их на свои описания объектов. Тип данных структура является простейшей реализацией классов, без скрытия данных. Структуры, а не классы, лежат в основе стиля программирования по Ms Windows.

Структура позволяет объеденить в одном объекте совокупность полей различного типа и назначения.

```
struct < Имя >
{
        < Тип > < Имя поля > ;
                . . .
        < Тип > < Имя поля > ;
} [ < Имя переменной > ] ;
```

Пример:

```
struct BOOK              /* Описание своего типа struct BOOK */
{
        char Name[20];   /* Поле имени */
        char Adres[40];  /* Поле адреса */
        int Year;        /* Возраст */
};                       /* Кон. описания структуры BOOK */
struct BOOK Book[50]; /* Описание переменной массива структур */
```

В приведенном примере в одну переменную объединены поля различных типов int и char. Такое представление позволяет выполнять некоторые операции сразу над всей структурой, а не над каждом полем в отдельности, такие операции как присвоение значения, передача в качестве параметров в функцию, запись/чтение из файла, сравнение и т.п..

Доступ к полям структуры

Работа с полями структуры осуществляется как с обычной переменной, но указывается не только имя структуры, но и имя поля. Разделитель между ними символ '.'.

Для предыдущего примера:

```
Book[4].Year=15 ; cprintf("%s",Book[4].Name) ;
```

Если структура описана, или передана, как указатель то доступ проще всего организовать так:

```
struct BOOK *B ;
(*B).Year=16 ; /* Обратите внимание, что не *B.Year !!! */
```

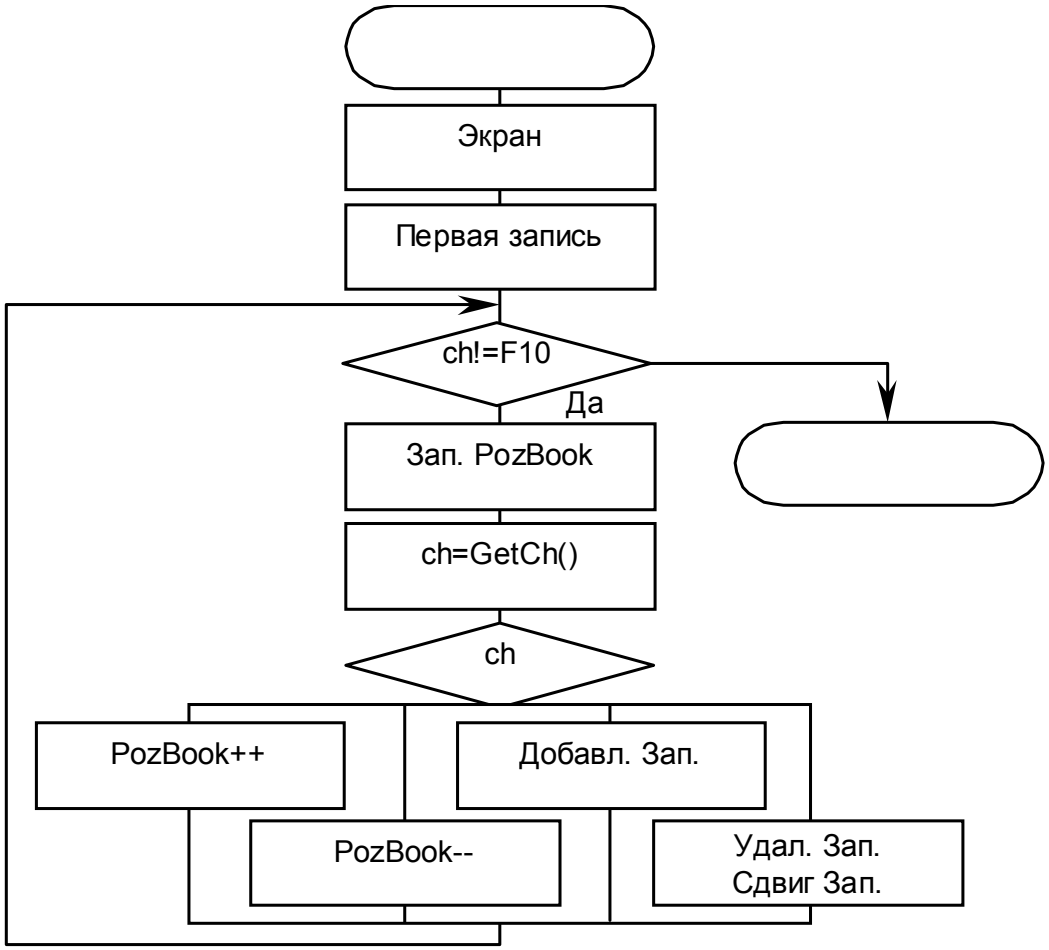
Телефонный справочник

Программа должна вводить, удалять, просматривать телефонный справочник. Рекомендуется использовать:

Для хранения информации - массив структур struct BOOK B[100];
Для записи кол-во введенной информации int N_Book ;
Для номера текущей записи int PozBook ;

Для определения кодов клавиш рекомендуется использовать свою функцию int GetCh(void), которая бы учитывала служебные клавиши (если первый символ 0, то брала бы следующий и добавляла к нему 512).

```
int GetCh(void)
{
        int ch ;
        ch=getch() ;
        if(ch==0) ch=512+getch() ;
        return(ch) ;
} /* кон. GetCh() */
```



Для сдвига записей: N_Book-- ;

for(i=PozBook;i<N_Book;i++) B[i]=B[i+1] ; При всех операциях необходимо проверять выход за размерность описанного массива структур. Избежать этого можно если создавать элементы динамически, т.е. не массив, а список.

Классы

Понятие Классы позволяет объединить данные с методами для их обработки. Это понятие почти аналогично структуре. Используя базовые классы и виртуальные функции можно создавать списки неоднородных элементов, с собственными функциями для обработки каждого типа данных. При этом, вне класса, обращение к ним будет одинаково.

Операция ++ означает увеличение, по этому С++ - дополнения к языку С. Подобные вещи можно реализовать и на стандартном Си. Использование классов, так называемый стиль программирования ООП (Объектно ориентированное программирование), позволяет на определенном уровне абстрагироваться от внутреннего описания объекта (подпрограмм реализующих определенные функции связанные с физическим смыслом программируемой задачи). Стиль ООП, в настоящее время, широко распространен благодаря большому количеству библиотек написанных в этом стиле, TurboVisual, визуальное программирование.

Однако далеко не каждая задача ставит проблемы, которые проще решить используя стиль ООП. Вопрос об использовании этого стиля каждый решает для себя сам.

Модернизация телефонного справочника под стиль ООП

Объектом выступает информация о человеке (имя, адрес, возраст) и методы (вывод на экран, ввод с клавиатуры).

```
class BOOK
{
        char Name[20] ;
        char Adres[40] ;
        public:
                int Year ;
                void PrintBook(void) ;
                void GetBook(void) ;
}
```

Слово public – означает часть доступную из вне класса, все что описано выше не доступно за пределами класса. Функции описываются как прототипы.

Описание функций класса

При описании функций класса перед именем ставится указание на класс к которому она относится. Переменные описанные внутри класса (Name) доступны, для методов, непосредственно.

Пример:

```
void BOOK::PrintBook(void)
{
        gotoxy(10,10) ;
        cprintf("%s",Name) ;
        . . .
} /* Кон. PrintBook() */
```

Вызов из основной программы

Доступ к открытым элементам класса осуществляется аналогично доступу к элементам структуры.

Пример:

```
. . .
class BOOK B[100] ;
. . .
B[PozBook].PrintBook() ;
. . .
B[N_Book++].GetBook() ;
. . .
B[PozBook].Year=12 ;
. . .
B[PozBook].Name[0]='\0' ; - НЕДОПУСТИМО !
```

поле Name описано в закрытой части класса, и доступно только его методам.

Для работы с классами необходимо пользоваться фалом с расширением *.CPP или указать это в опциях компилятора.

Некоторые замечания о стилях программирования

Распределение ресурсов

При написании программы учитывайте правило стоимостного анализа. Если функция вызывается 100 раз, а другая только один раз за цикл работы программы, при дефиците ресурсов первая должно получить значительно больше внимания чем вторая. Если программа предназначена для вывода данных, место для них на экране важнее, чем для рамок и прочего оформления.

Функции

Если фрагмент программы, реализующий законченное действие, используется более одного раза, его целесообразно вынести в функцию. Затраты времени на сам вызов функции не значительны. Старайтесь чтобы каждая функция решала только ОДНУ задачу.

Имена переменных и функций

Старайтесь давать осмысленные имена. Чтобы по ним сразу можно было сказать для чего используется эта переменная, что выполняет эта функция. Если вы пользуетесь специфичными типами поймете это в имени переменных.
Например:

```
int i,j,k;          /* Рабочие переменные для счетчиков */
char ch,St[80];     /* Символы */
long lh;            /* Помечено что переменная long (длинная) */
float x,y;          /* Явно координаты,(аргумент и значение) */
```

Не решайте проблем которых не существует

Не старайтесь писать функции с учетом дальнейших все возможных улучшений и добавлений. Лучще оставляйте возможности для расширения, а не перегружайте функции.

Комментарии

Обязательно комментируйте Ваши программы, но избегайте бессмысленных комментариев. Обязательно помечайте закрывающие }. Распологайте открывающие и закрывающие { } на одном уровне, со сдвигом от предыдущего оператора. Если есть вложенные циклы, указывайте, по какой переменной они организованы. Старайтесь распологать комментарий на той строчке, к которой он относится.
Пример:

```
for(i=0;i<10;i++)
{
        . . . . .
} /* Кон. for() */
```

НО:

```
for(i=0;i<10;i++)
{
        . . . .
        for(j=0;j<10;j++)
        {
                . . . .
        } /* Кон. for(j=0;j<10;j++) */
        . . . .
} /* Кон. for(i=0;i<10;i++) */
```

Набор текста

При наборе текста не жалейте места, это не увеличивает размер программы и не уменьшает скорость ее работы. Обязательно отступайте при вводе составного оператора, программа в таком оформлении (см. пред. пример) сразу становится более читабельной, в ней проще разбираться. Старайтесь не помешать в строке больше одного оператора. Разделяйте законченные фрагменты пустыми строками.

Используйте разделители для визуального разделения подпрограмм. Например:

```
/*--------------------------------------------------*/
```

Или помещайте их в отдельные файлы, с аналогичными названию функции именами.

Оператор if

При реализации оператора if старайтесь избегать случая:

```
if(<Условие>)
{
.. Много строк ..
} /* Кон. if() */
else
return(<Значение>) ;
```

Лучше использовать обратное условие, это упростит программу: if(!<Условие>)

```
return(<Значение>) ;
. . .
```

Оператор switch

При использовании оператора switch не забывайте про default, даже если он Вам и не нужен, используйте его для индикации ошибочных ситуаций.

Пример:

```
switch(Type)
{
        case 1: . . . .
        case 2: . . . .
        default: printf("\n Невозможное значение Type%d",Type) ;
} /* Кон. switc() */
```

Оператор goto.

Язык Си содержит оператор goto, но использование его есть очень плохой стиль. При использовании стиля ООП (C++) его применение не допустимо. В общем случае goto следует избегать, так как конструкция языка имеет более гибкие решения всех Ваших проблем.

Коды возврата ошибок

Всегда проверяйте коды возврата ошибок при операциях, где они могут возникнуть: инициализация графики, открытие файлов, запись в файлы, выделение памяти.

Интерфейс вашей программы

Как работать с Вашей программой должно быть ясно из ее внешнего вида. Не забывайте о строках подсказки, иначе пользователь не поймет, что хочет от него программа. Интерфейс требующий меньшего нажатия клавиш лучше того, где этих операций много.

Алгоритм

В начале работы над программой обязательно сделайте себе четкую постановку задачи. Помните что если Вы не можете выразить то что собираетесь сделать на русском языке, Вы не реализуете это ни на каком языке програмироавния. После этого напишите какие вам необходимы данные (структуру данных) и блок-схему алгоритма. Продумайте форму рабочих экранов и переключение между ними. Разбивайте сложные проблемы на составные части (функции). Старайтесь оформить Вашу задачу в виде набора функций, а не сваливать все в одну кучу. Все это очень облегчит написание программы.

Игровая программа Питон

Программа создается на базе программы движения символа по экрану.

Общие требования к программе

Программа должно содержать строку подсказки в нижней строке. Игровое поле должно быть ограниченно рамкой. В верхней строке должно располагаться текущее время, уровень, количество набранных очков. При старте программы должна выводится информация об авторах программы. В конце работы выводиться результат игры и таблица рекордов (желательно отсортированная). При записи таблицы рекордов на диск можно ее зашифровать.

* - Реализуется во вторую очередь.

** - Не забыть удалить координаты съеденного зайца из массива и уменьшить количество зайцев.

Рекомендуется использовать:

Для хранения координат:

```
struct CORD
{
        int x ;
        int y ;
};
```

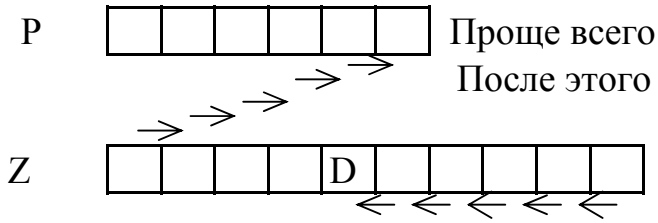
Для запоминания питона:	struct CORD P[100] ;
Для длинны питона:	int NP ; /* NP<100 */
Для запоминания координат зайцев:	struct COORD Z[100] ;
Для количества зайцев:	int NZ ; /* NZ<100 */
Координаты головы:	int x,y ;
Старые координаты головы:	int old_x,old_y ;
Для рисования символа:	void PutC(int x,int y,chat ch) ;
Признак выхода из программы:	char Quit ; /* Если 1 то выход */

Проверка, что удав врезался сам в себя: Необходимо просмотреть все звенья удава P[i] и проверить их на совпадение с координатами головы (x,y). Если совпадет то, устанавливается флаг выхода Quit=1 ;.

Флаг стирания хвоста удава: char flag ; flag=1 - необходимо стирать хвост. Если удав съел зайца (длинна увеличилась), хвост стирать не надо - flag=0 ;.

Проверка что удав съел зайца: Необходимо сравнить координаты головы удава (x,y) с координатами всех зайцев Z[i]. Если совпадает flag=0 , длинну удава увеличиваем на 1, но не больше 100 (так описаны массивы).

Сдвиг удава:



Проще всего это организовать через обратный цикл.
После этого P[0]= <Новые координаты головы>

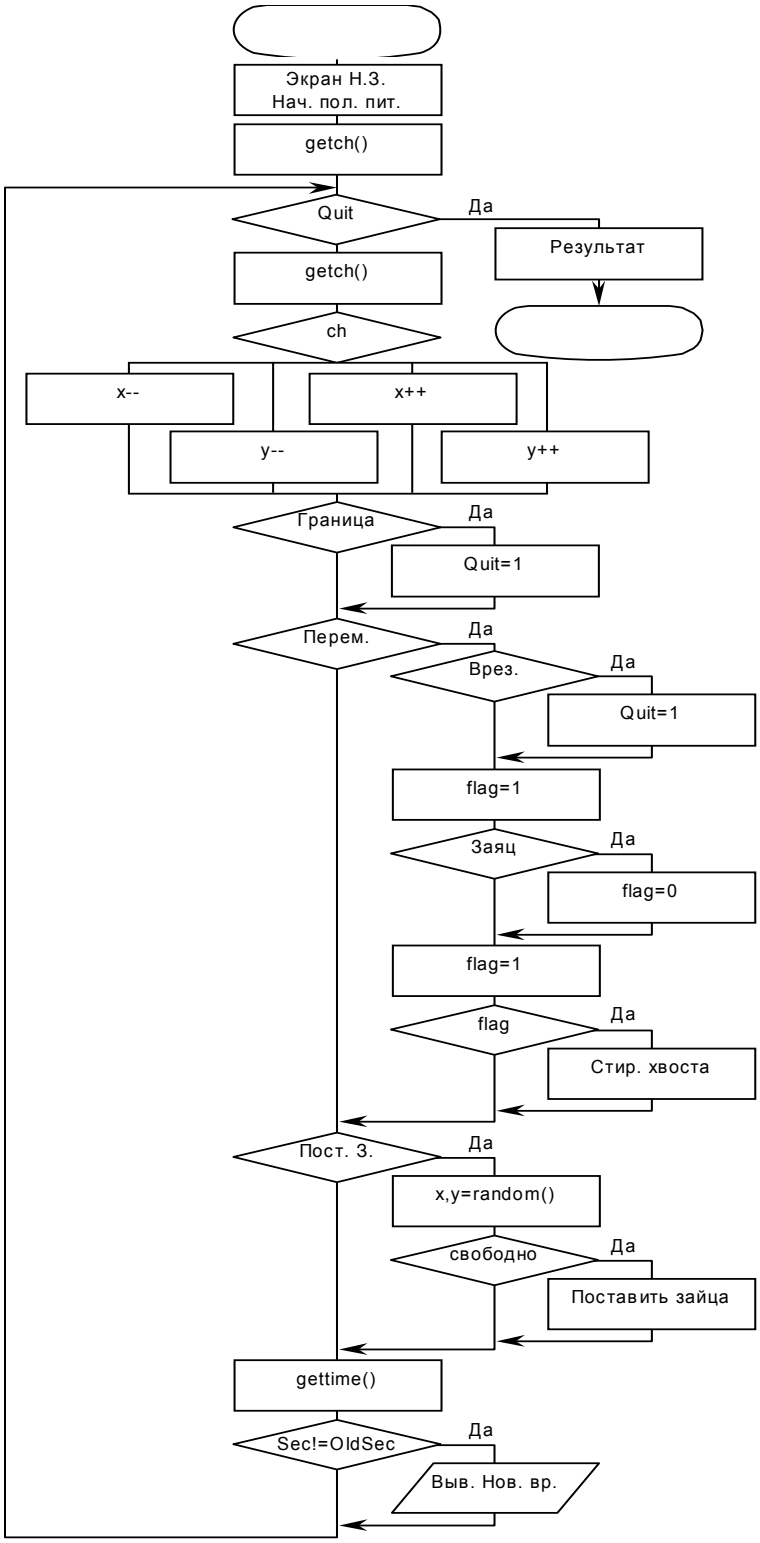
Если удав съел зайца: необходимо удалить координаты съеденного зайца из массива Z, сдвинуть часть массива, что бы не было пропуска, и уменьшить счетчик зайцев (NZ).

Поставить зайца: эту операцию надо проводить в случайные интервалы времени. Проще всего реализовать это с помощью генератора случайных чисел. Берется случайное число в диапазоне от 0 до 100 и проверяется, если оно больше 95, тогда ставится новый заяц. Вероятность этого 5%, так что зайцы будут ставиться не часто. Порогом (95) можно менять скорость их появления. После этого берутся случайные координаты, проверяется, что место свободно (удобнее использовать проверку, что место занято, тогда continue), если свободно то ставится заяц, его координаты добавляются в массив Z, увеличивается NZ (но не больше 100).

При чтении клавиши необходимо проверять, нажата ли она. Если нет, то сделать задержку, рекомендуется на RANGE*10 милисекунд, и продолжать движение в старом направлении.

Переменная RANGE обозначает уровень, чем меньше, тем быстрее бегает питон, Ее можно уменьшать автоматически по мере роста питона, или по времени игры.

Блоксхема программы «Питон»



Работа с временем
Модуль dos.h

Для получения системного времени и даты Си содержит большой набор функций, отличающийся формой получения результата. Проще всего пользоваться функцией gettime(struct time *t). Функция записывает текущее время в поля структуры, описанной в модуле dos.h.

```
struct time
{
        unsigned char ti_hour ; /* часы*/
        unsigned char ti_min ; /* минуты*/
        unsigned char ti_sec ; /* секунды*/
        unsigned char ti_hund ; /* сотые доли секунды */
}
```

Пример работы со временем:

```
#include <dos.h>
#include <conio.h>
void main(void)
{
        struct time T ; /* Сама структура уже описана в dos.h */
        char OldSec ;
        while(!kbhit())
        {
                gettime(&T) ;
                if(OldSec!=T.ti_sec)
                {
                        OldSec=T.ti_sec ;
                        gotoxy(70,1) ;
                        cprintf("%2d:%02d:%02d",T.ti_hour,T.ti_min,T.ti_sec) ;
                } /* Кон. if */
        } /* Кон. while */
} /* Кон. main() */
```

В программе перерисовка времени осуществляется только если изменились секунды, иначе оно будет часто моргать. При выводе времени использовался формат %02d, иначе бы время писалось в виде 9: 7:36.

Работа с файлами модуль stdio.h

Язык Си содержит два варианта работы с файлами. Первый через указатель на файл, второй как с устройством.

```
FILE * fopen(<Имя файла>,<Режим доступа>) ;
fprintf(<Указатель на файл>,"<Строка>",[<Переменные>]) ;
fscanf(<Указатель на файл>,"<Формат>",<Адрес переменных>) ;
```

fclose(<Указатель на файл>) ;
feof(<Указатель на файл>) ;

Функция fopen возвращает указатель на открытый файл. Если операция закончилась не удачно (файл не удалось открыть), возвращается значение NULL, это обязательно надо проверить, иначе могут быть непредсказуемые сбои. В качестве аргументов функции передаются имя открываемого файла и режим доступа. Режим доступа может быть одним из:

"r" - Открыть для чтения (существующий)
"w" - Открыть для записи, если файл был то он уничтожается
"a" - Добавлять в конец файла

Функции fscanf(...) и fprintf(...) аналогичны функциям scanf(...) и printf(...), но ввод/вывод происходят из/в файла а не с клавиатуры/экрана. Первым аргументом в них передается указатель на файл с которым необходимо работать.

Функция fclose(...) закрывает открытый файл. Это обязательно необходимо сделать, иначе содержимое файла будет утраченным, а на диске могут образоваться потеренные сектора.

Функция feof(...) возвращает Да (1), если достигнут конец файла, чаще, пользуются обратным условием - пока не достигнут конец файла.

```
#include <dos.h>
#include <stdlib.h>
int main(void)
{
	char Name[20],ch ;
	FILE *Input , *Output ;
	printf("\n Копирование файлов.\n Введите имя исх. файла:") ;
	scanf("%s",Name) ;
	printf("\n Копирование %s в RESERV.DAT") ;
	Input=fopen(Name,"r+") ;
	if(Input==NULL)
	{
		printf("\n Ошибка открытия файла \"%s\" ",Name) ; return(0) ;
	} /* Кон. if */
	Output=fopen("RESERV.DAT","w+") ;
	if(Output==NULL)
	{
		printf("\n Ошибка открытия файла \"RESERV.DAT\" ") ; return(0) ;
	}/* Кон. if */
	while(!feof(Input))
	{
		fcanf(Input,"%c",&ch) ;
		fprintf(Output,"%c",ch) ;
	} /* Кон. while */
```

```
        fclose(Input) ;
        fclose(Output) ;
        return(1) ;
} /* Кон. main() */
```

Приведенный пример для такой операции не очень эффективен, лучше было бы читать информацию сразу блоком, а не по одному символу.

Работа с файлами
Модуль io.h, sys\stat.h, fcntl.h

Работа с BINARY файлами удобнее для записи не текстовой информации. Дополнительным удобством этого способа работы с файлами является то, что данные при операциях ввода вывода не преобразуются, а также то, что можно записывать сразу блоки памяти - например весь массив.

int Handle;	Целая переменная являющаяся идентификатором файла
Handle=open(<Имя>,<Режим>,<Атр.>)	Открыть файл в указ. режиме. Если не открылся Handle==-1
read(Handle,<Куда>,<Сколько>)	Прочитать из файла <Сколько> байт то адресу <Куда>.
write(Handle,<Откуда>,<Сколько>)	Записать в файл с адреса <Откуда> указанное число байт
lseek(Handle,<Смещение>,SEEK_SET)	Переместить указатель в файле eof(Handle) Истина если достигнут конец файла
close(Handle)	Закрыть файл

Режимы открытия файла

O_BINARY	-двоичный файл
O_CREAT	-создать
O_TRUNC	-открыть с усечением (обнулить размер)
O_APPEND	-открыть для дополнения
O_RDWR	-открыть для чтения и записи

Атрибуты создаваемого файла

S_IWRITE - разрешение записи, если он не указан файл создается с атрибутом только для чтения.

Константы для позиционирования lseek()

SEEK_SET - от начала файла.

Пример:

```
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>
int main(void)
{
	float A[10],B[10],C[10] ;
	int Handle ;
	Handle=open("test.dat",O_BINARY) ;
	if(handle==-1)
	{
		printf("\n Ошибка открытия файла test.dat") ; return(0) ;
	} /* Кон. if() */
	read(Handle,A,sizeof(A)) ;
	lseek(Handle,0L,SEEK_SET) ;
	read(Handle,B,sizeof(B)) ;
	close(Handle) ;
	Handle=open("res.dat",O_BINARY|O_CREAT|O_TRUNC|O_RDWR,
		S_IWRITE);
	if(handle==-1)
	{
		printf("\n Ошибка создания файла res.dat") ;
		return(0) ;
	} /* Кон. if() */
	write(Handle,C,sizeof(C)) ;
	close(Handle) ;
	return(1) ;
} /* Кон. main() */
```

Функция sizeof()- возвращает количесво байт в указанной переменной.
В функции lseek() используется 0L, L - это длинное целое.

Оглавление

Движение символа по экрану	22	9	16
Функции	23	10	17
Передача параметров	24	10	17
Возвращение значений	24	-	17
Факториал	24	10	17
Программа вычисления факториала	25	10	17
Возвращение нескольких значений	25	-	18
Слияние двух строк	27	-	18
Работа со строками	27	-	18
Структуры	28	11	19
Доступ к полям структуры	28	11	19
Телефонный справочник	29	-	20
Классы	30	18	21
Модернизация телефонного справочника под ООП	30	-	22
Описание функций класса	31	-	21
Вызов из основной программы	31	-	21
Некоторые замечания о стилях программирования	31	17	23
Игровая программа Питон	35	12	25-27
Работа со временем	38	14	28
Работа с файлами, модуль stdio.h	38	15	29
Работа с файлами, модуль io.h, sys\stat.h, fcntl.h	40	-	29

Некоторые рекомендации к плану занятий

Данный курс читается в 3 режимах:

1. 1 год, 2 часа в неделю (72 часа);
2. 1.5 года, 2 часа в неделю + летняя практика (132 часа)
3. 2 года, 2 часа в неделю + летняя практика (168 часов)

Вариант 1 года - слишком сокращен и сжат, многие задания даются в сокращенном варианте и пониженных требованиях к программам. Необходимость в таком варианте вызвана недостатком часов в программе некоторых групп.

Вариант 1.5 года - чуть более интенсивный вариант 2 годичного цикла.

Вариант 2 года - основной и рекомендуемый к использованию. Немного растянутая водная часть создает возможность сделать запас времени для сильных групп, или подтянуть более слабые групп, обеспечивая плавное вхождение в курс.