ТАРТУСКИЙ УНИВЕРСИТЕТ

ОСНОВЫ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

II

ТАРТУ 1990

ТАРТУСКИЙ УНИВЕРСИТЕТ
Кафедра программирования

ОСНОВЫ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

II

О. Кихо, Е. Маширина, Я. Маширин

Тарту 1990

Утверждено на заседании совета математического факультета ТУ 23 ноября 1990 года

С О Д Е Р Ж А Н И Е

В В Е Д Е Н И Е

Используемые при общении с ЭВМ знаковые системы записи программ и структур данных разделяются на два принципиально разных класса. Наиболее распространенной является чисто текстовая форма записи, в которой объект представляется в виде последовательности символов. При этом, для указания структуры объекта используются как круглые, так и словесные скобки (if..fi , do..od , begin..end), а также возможность записи текста лесенкой.

Этот способ с середины 50-х годов бесконкурентно господствовал до самого последнего времени. Лишь сравнительно недавно разработаны новые методы и соответствующие системы графического представления структур, в том числе рассматриваемый здесь метод схемного программирования [7, 8, 9, 10]. В этих системах более элементарные компоненты объекта, например, формулы, простые операторы и их последовательности, представляются в традиционной текстовой форме. Для указания же их композиции в более сложные структуры применяются графические средства. Преимущество графических знаковых систем перед чисто текстовыми способами заключается в более продуктивном использовании мощного зрительного аппарата и ассоциативного мышления человека.

Первое сообщение о методе схематического программирования опубликовано в 1983 году. С самого начала этот метод хорошо себя зарекомендовал как при обучении студентов программированию, составлению сложных алгоритмов, так и в качестве основы разработки специальных систем программирования. В настоящее время схематическое программирование как разновидность графических методов программирования приобретает еще большую актуальность: расширяющиеся графические возможности новых ЭВМ повышают, в частности, плодотворность идеи схематического представления структур программ. Разработана соответствующая система программирования [12], реализованная на базе языка программирования Си. Данное пособие предназначено для ознакомления с программированием в этой системе. Оно дает обзор средств программирования в системе SKC, демонстрируя некоторые из них конкретными примерами решений

различных математических задач. Поскольку система представляет собой метод схематического программирования, реализованный на базе языка Си, то в работе содержится описание некоторых элементов данного языка программирования.

Приведенные примеры решений задач могут рассматриваться как учебное пособие при практической работе, которую разумнее начинать с некоторой модификации и дополнения уже имеющихся программ. Только затем возможно создание новых программ, причем, начинать надо с написания аналогичных.

Система SKC работает на IBM PC(AT). Некоторые программы предполагают использование средств графики. Для работы необходимо образовывать комплект файлов:

SKM.EXE - схемный редактор и процессор;

TCC.EXE, TLINK.EXE - компилятор и сборщик TurboC;

*.H - стандартные головные файлы;

*.LIB, *.OBJ - библиотечные файлы;

*.SKM - тексты создаваемых схематических программ.

Для лучшего восприятия текст программ ограничен объемом одного листа. Программы со сложным алгоритмом решения в работе рассматриваться не будут.

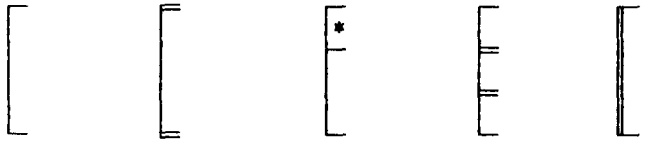
Работа включает четыре главы, список использованной литературы и приложение. Первая и вторая главы содержат обзор средств программирования системы SKC и схемного редактора системы. В третьей главе даются некоторые советы по написанию программ. В четвертой главе помещены тексты конкретных программ,составленных в качестве примеров, краткие описания их алгоритмов, а также примеры решений. Программы подразделяются по темам. В приложении приводится в алфавитном порядке перечень основных терминов с указанием страниц, на которых они впервые встречаются в тексте работы или в конкретной программе.

ГЛАВА 1. СРЕДСТВА ПРОГРАММИРОВАНИЯ В СИСТЕМЕ SKC

1. Структура алгоритма

Алгоритм - это строгая и четкая конечная система правил, которая определяет последовательность действий над некоторыми объектами и после конечного числа шагов приводит к достижению поставленной цели. Основные понятия, используемые при составлении алгоритмов: элементарная инструкция - описание объекта или отдельный шаг преобразования данных; условие - проверка ситуации (с целью определения разветвления). Условие управляет выполнением последовательности элементарных действий. Сложные алгоритмы образуются объединением элементарных инструкций и условий. В дальнейшем элементарные инструкции и условия будем называть общим термином "элементарные компоненты". Порядок их выполнения строго регулирован.

Схема изображается как охваченный слева особой скобкой набор элементарных компонент и (под)схем. Форма скобки определяет род схемы:



простая схема	неогражденный цикл	огражденный цикл	схема выбора	ниша

Охватывающая вертикальная скобка называется линией уровня. Язык, на котором пишутся элементарные компоненты, называется базовым языком. В общем случае к базовому языку предъявляются минимальные требования. В схематическом программировании программисту необходимо знать только подмножество некоторого языка программирования, поскольку базовый язык не включает структур управления. Последние универсальны и определены независимо от базового языка [7].

Пример простой схемы, в которой содержатся два элементарных компонента:

```
┌ x > 0 ?                  --- условие
  увеличить n;             --- элементарная инструкция
└
```

Тремя минусами начинается строка (или часть строки), называемая комментариями. При выполнении программы она опускается (до конца строки). Согласно семантике условия, о которой подробнее будет рассказано ниже, выполнение данной схемы происходит следующим образом: проверяется условие "x > 0 ?" и, в случае его выполнения, увеличивается n.

На рисунке 1 показана вложенность двух простых схем. В этом примере, если выполняется, что x > 0, увеличивается значение n; перед этим, если и y > 0, то увеличивается также m.

```
┌ x > 0 ?
  ┌ y > 0 ?
    увеличить m;
  └
  увеличить n;
└
```

Рисунок 1. Вложенность схем.

Наряду с элементарными компонентами и подсхемами элементом схемы может быть выходная стрелка (т.н. терминатор). Выполнение выходной стрелки означает завершение выполнения той схемы, охватывающей скобки которой достигает стрелка. Вместе с тем стрелка означает конец области действия предшествующего ей условия, если такое имеется. Таким образом, семантика условия определяется так:

если условие удовлетворяется, то выполнение идет в естественном порядке;

если условие не удовлетворяется, то пропускается выполнение следующих за ним элементарных компонент до ближайшей стрелки или, в случае отсутствия терминатора, до конца схемы.

Семантика условия одинакова во всех схемах. Выполнение схемы, показанной на рисунке 2а, происходит прежде всего на основании оценки выражения |x - y|:

если |x - y| > 1, то выводится на печать x и выполняется подсхема;

если |x - y| ⩽ 1, то выводятся на печать x и y .

В подсхеме на том же рисунке в зависимости от того, x больше y или нет, уменьшается значение x или y.

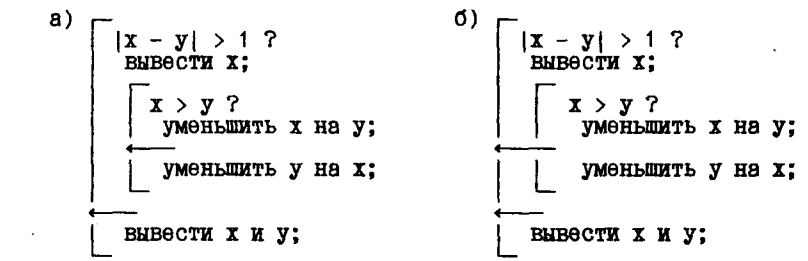


Рисунок 2. Равноценные схемы.

Выходная стрелка может быть продлена, то есть может обозначать прерывание одной или нескольких охватывающих схем. Выполнение продленной стрелки означает прерывание выполнения всех соответствующих схем и выход из схемы, скобки которой достигает стрелка. Продленная стрелка считается компонентой той схемы, в которой она начинается. Поэтому схемы а и б на рисунке 2 равноценны.

Таким образом простая схема образуется охваченной слева скобкой особого рода ([ ) вертикальной последовательностью, членами которой могут быть элементарные компоненты (условия и элементарные инструкции), подсхемы и стрелки. Выполнение простой схемы происходит последовательно, в направлении сверху вниз, причем в случае, если условие не удовлетворяется, идущая за условием часть схемы до ближайшей стрелки или

до конца схемы не выполняется.

Сложная схема на рисунке 3а дополнена пунктирными стрелками, которые поясняют ход выполнения в случае, если условие не удовлетворено. В простой схеме на рисунке 3б используются внутренние линии уровня для изображения "охраняемых" условиями команд. Только в последней подсхеме скобку можно опустить, но в целях симметрии это делать все-таки нежелательно.

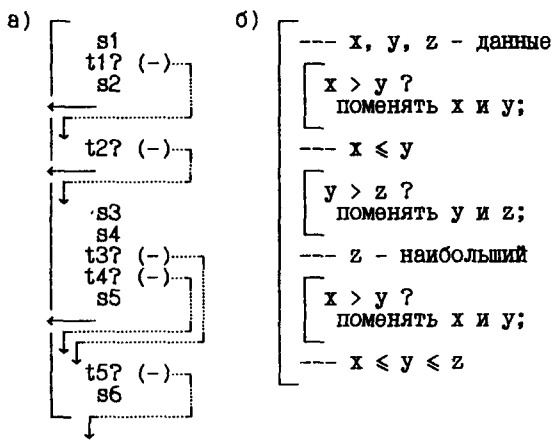
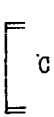


Рисунок 3. Роль условий в схеме (а) и алгоритм сортировки трех чисел по неубыванию (б).

Наряду с простыми схемами используется другой важный род схем - неограниченный цикл или цикл без заголовка. Особенность обозначения неограниченного цикла по сравнению с простой схемой заключается в ином виде охватывающей скобки, а именно:



Принцип выполнения условий и стрелок, входящих в неограниченный цикл, остается без изменений. Отличие от выполнения простой схемы состоит в том, что выполнение цикла начинается

с начала, как только достигается конец, то есть выполнен последний элемент схемы цикла. Цикл без заголовка не имеет границы числа повторений содержания С. Его завершение регулируется только условиями и терминаторами внутри содержания. Семантика условия остается прежней, то есть, если условие не выполняется, то пропускаются следующие за ним элементарные компоненты до ближайшей стрелки или конца схемы, если стрелка отсутствует. В последнем случае выполнение цикла заканчивается. Выполнение стрелки означает выход из схемы. В связи с циклом вводится понятие слабой стрелки. Слабая стрелка обозначается пунктирной линией и означает прерывание шага цикла. На рисунке 4 демонстрируются три возможности написания алгоритма Евклида (нахождения наибольшего общего делителя двух положительных целых чисел), используя неогражденный цикл.

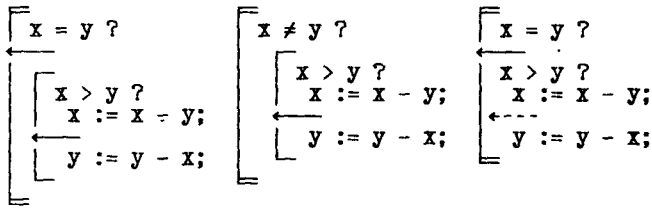
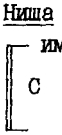


Рисунок 4. Неогражденные циклы.



означает макрокоманду, которая во время трансляции схематической программы заменяется содержимым файла, имя которого указывается в заголовке. Если такого файла нет или же имя вообще не указано, то ниша заменяется на группу элементарных инструкций С. Таким образом ниша служит средством пошаговой разработки программы [9]. В схематических алгоритмах нишей изображается подалгоритм, который определяется либо отдель-

но, либо уточняется позже. Внутри ниши обычно пишутся комментарии, которые объясняют ее назначение. На рисунке 5 приведен алгоритм, включающий в себя два подалгоритма A и B, которые выделены в отдельные ниши и состоят из элементарных инструкций.

Огражденный цикл имеет вид

где p - заголовок цикла, C - содержание цикла. Огражденные циклы отличаются друг от друга разновидностью заголовков, содержание же цикла всегда представляет собой вертикальную последовательность, которая может включать элементарные компоненты, подсхемы и выходные стрелки. Значения условий, стрелок те же самые, что и в случае неогражденного цикла.

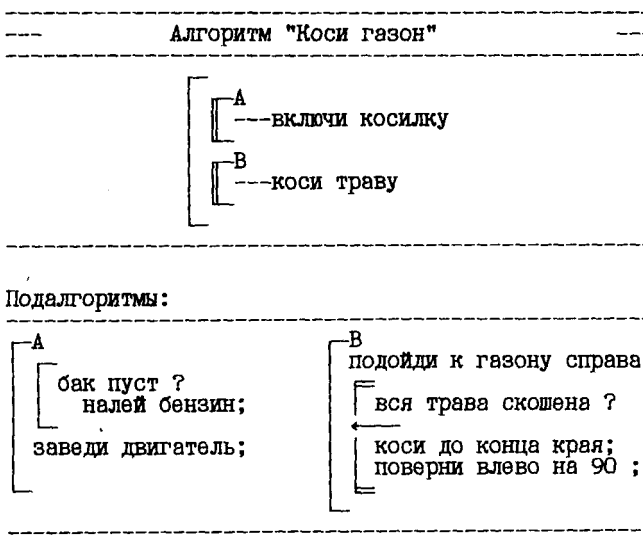


Рисунок 5. Алгоритм с использованием ниш.

Рассмотрим подробнее разновидности огражденного цикла. В случае цикла вида

```
* k
  C
```

число повторений тела цикла равно значению выражения m=max([k],0), то есть выполнение содержания C происходит не более, чем m раз. На рисунке 6 представлен алгоритм движения по сторонам квадрата (длина стороны 5 шагов).

```
* 4
  * 5
    шаг вперед;
  поворот направо;
```

Рисунок 6. Ходьба по квадрату.

В заголовке огражденного цикла второго вида используется параметр i - имя некоторой целой или вещественной переменной с любой точностью, a, b, c - целые или вещественные выражения:

```
* i = a..b;c
  C
```

```
* i = a..b --- c=1 по умолчанию
  C
```

Значения a, b и c представляют, соответственно, начальное и предельное значения и приращение параметра. Запись a..b;c означает кортеж (a, a+c, a+2c,..., a+(n-1)c), где n=[(b-a+c)/c]. Выражение a..b означает кортеж (a, a+1,..., b). Число повторений равно нулю, если b<a и c>0 или b>a и c<0. В случае, когда c=1, заголовок пишут без указания шага (;1). Пример использования цикла с параметром в заголовке показан на рисунке 7.

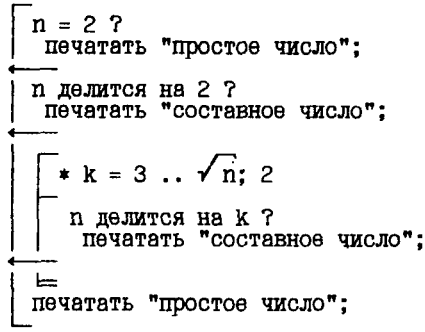


Рисунок 7. Алгоритм нахождения простых чисел.

Заголовки вида *k либо *i = a..b;c называются элементарными. Составной заголовок представляет собой кортеж написанных обычно друг под другом элементарных заголовков. Схема цикла с составным заголовком - есть сокращенная запись группы вложенных друг в друга огражденных циклов. Более точно смысл такой схемы показан на рисунке 8.

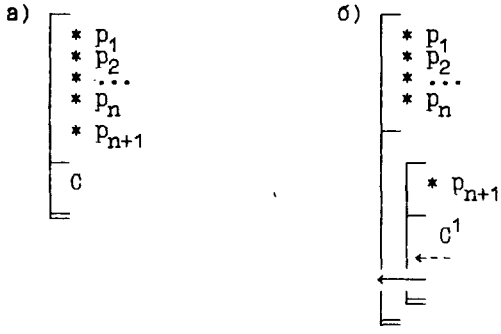


Рисунок 8. Цикл с составным заголовком (а) и его семантика (б).

В схеме справа через C^1 обозначена последовательность, полученная из C увеличением протяженности всех терминаторов на единицу, кроме слабых терминаторов, вложенность которых в

исходной схеме равняется их протяженности. Протяженность таких слабых терминаторов не изменяется. Понятия протяженности и вложенности здесь не уточняем [8]. Практически достаточно учитывать, что стрелка в C (рис.8а) прекращает весь цикл, а слабая стрелка в C прекращает шаг самого внутреннего цикла.

На рисунке 9 показан еще один род схем - схема выбора.

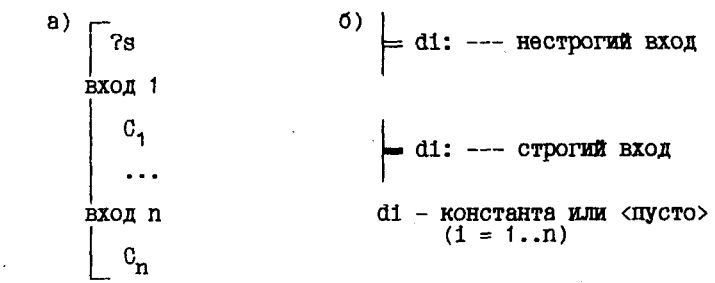


Рисунок 9. Общий вид схемы выбора (а) и виды выходов (б).

При выполнении схемы выбора значение селектора s последовательно сравнивается с детекторами d1, d2,..., dn. В случае, если найден детектор di, значение которого равно s или он является пустым, то выполняется простая схема $\lceil C_i$.

После этого, если i = n, то выполнение схемы выбора закончено, в противном случае (i<n) дальнейшее выполнение зависит от характера входа (d_{i+1}). В случае строгого входа выполнение схемы выбора закончено. В случае нестрогого входа выполнение схемы выбора продолжается выполнением $C_{(i+1)}$,... Другими словами, выполнение тела схемы выбора начинается с первого подходящего входа и кончается первым вслед за ним строгим входом (или концом схемы, или выходной стрелкой).

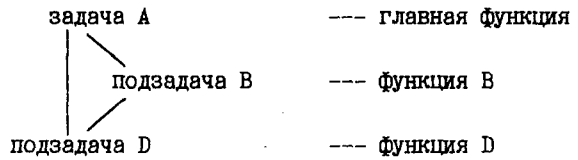
В схематическом алгоритме комментарием считается:

1) часть строки (или целая строка) до конца строки, которая начинается тремя минусами;

2) строки, которые размещены у начала схемы (за ⌜, ⌠), конца схемы (за ⌞, ⌡), у ограничителя заголовка (за ⊢), у стрелки (за ⟵ ,<--).

2. Структура программы

В системе SKC алгоритм задачи представляется в виде схематической программы, которая строится из так называемых функций:



В приведенном примере имеем дело с задачей A, выполнение которой включает как выполнение подзадачи D, так и выполнение подзадачи B. Для решения последней также необходимо решать задачу D.

SKC-программа в основном состоит из описаний функций. Имя главной функции всегда должно быть "main", имена остальных функций выбираются произвольно. Имена функций должны удовлетворять общим требованиям, о которых будет рассказано в следующем пункте.

Общий вид описания функции в SKC:

```
-------------------------------------------
__тип _имя функции(__формальные_параметры)
__описания_формальных_параметров

┌  __описание_локальных переменных
|
|  __действия
|
└
-------------------------------------------.
```

В метаязыке двумя символами подчеркивания (__) начинается название необязательного элемента. Понятие типа вводится позже, вначале рассматриваются целочисленные функции, название типа которых (int) можно опускать.

Описанные в простой схеме локальные переменные и действия называются телом функции, а предшествующая им часть - заголовком функции. Пунктирные линии в начале и конце

описания являются комментариями и используются для лучшего оформления.

Вызов функции описывается следующей строкой:

_имя_функции(__список_аргументов);

Программа решения описанной выше задачи может иметь структуру, указанную на рисунке 10. SKC-программа может размещаться в одном или нескольких файлах. В рассмотренном примере используется один файл. В целях лучшего оформления программы в первой строке желательно писать имя файла (ФАЙЛ1), а заканчивать описание пунктирной линией с точкой в конце. Описание же отдельных функций можно разделять просто пунктирной линией.

```
-----------------------------------ФАЙЛ1-----------
-----------------------------------------------------
main()          --- начало описания главной функции
 ...
 D();           --- вызов функции D
 ...
 B();           --- вызов функции B
 ...

                --- конец описания главной функции
-----------------------------------------------------
B()             --- начало описания функции B
...
 ...
 D();           --- вызов функции D
 ...

                --- конец описания функции B
-----------------------------------------------------
D()             --- начало описания функции D
...
 ...

                --- конец описания функции D
-----------------------------------------------------.
```

Рисунок 10. Пример структуры SKC-программы.

В каждом случае выполнение программы заключается в вычислении главной функции, в ходе которого вызываются и выполняются другие функции. Описания функций не могут содержаться друг в друге.

Вызываться могут и стандартные функции языка Си. Последние будем называть Си-функциями. Одной из Си-функций, к примеру, является функция вывода "printf". В простом отдельном случае этой функции можно задать один аргумент - выводимый текст, заключенный в кавычки. Например,

```
printf("Здравствуйте !");
```

выведет на экран

```
Здравствуйте !
```

Автоматического перевода строки после вывода на печать не происходит. Поэтому следующие два сообщения

```
printf("Вставьте диск.");
printf("Нажмите клавишу.");
```

будут выведены в одной строке:

```
Вставьте диск.Нажмите клавишу.
```

При желании вывода каждого сообщения на отдельную строку, необходимо в текст включить символ перевода строки \n. Тогда после выполнения команд

```
printf("Вставьте диск.\n");
printf("Нажмите клавишу.\n");
```

или

```
printf("Вставьте диск.");
printf("\nНажмите клавишу.\n");
```

или

```
printf("Вставьте диск.\nНажмите клавишу.\n");
```

на экране появится текст, размещенный на двух строках:

```
Вставьте диск.
Нажмите клавишу.
```

Существуют и другие символы, меняющие свой смысл, если перед ними стоит обратная дробная черта:

```
\b - шаг назад
\f - новая страница
\n - новая строка
\t - табуляция
\' - апостроф
\" - кавычки
\\ - обратная косая
\0 - нулевой символ
```

В простом случае программа состоит только из описания главной функции.

```
-------------------------------------------ФАЙЛ2-----
---              функция printf                    ---
-----------------------------------------------------
main()
{
------ 3 раза печатать на одной строке
  {
  * 3
  printf("Okay  !");
  }
------ 4 раза печатать друг под другом
  {
  * 4
  printf("Okay  !\n");
  }
}
-----------------------------------------------.
```

Рисунок 11. Программа, состоящая только из главной функции.

При написании текста программы широко используются специальные макрокоманды. Они не относятся к множеству элементарных конструкций. С помощью макрокоманд можно "определять" текст и "включать" в текст файлы. Команда

#define _имя _определение

обеспечивает пользователю возможность макроподстановки, которая позволяет вынести отдельные фрагменты, например, явные константы, из текста программы и вместо них использовать их символические имена. Команды этого вида называются макроопределениями. Например, если в начале программы написать

#define max(a,b) (((a) > (b)) ? (a) : (b)) ,

то каждое вхождение в программу выражения max(a,b) заменяется на указанное определение, в частности, max(x+1,y-2) заменяется на ((x+1) > (y-2)) ? (x+1) : (y-2).

Макрокомандой можно задавать также имена параметрам программы. Такие параметры традиционно обозначаются большими буквами. К примеру, во многих системах признаком конца файла является -1, но существуют и другие файловые системы. Поэто-

му целесообразно в программу, обрабатывающую файлы, включить макроопределение:

```
#define EOF  -1 ,
```

которое ставит в соответствие обозначению EOF число -1. Макроопределения облегчают переделку программы для работы в другой системе.

Второй очень важной макрокомандой является макровключение:

```
#include <_имя_файла>   или   #include "_имя_файла"
```

Данная команда заменяется транслятором на содержимое указанного файла. Полученный комбинированный текст затем обрабатывается как единый файл с текстом программы. Подобным образом можно "срастить" несколько файлов. Включенные файлы в свою очередь тоже могут содержать макровключения [3]. Если имя файла заключено в двойные кавычки, считается, что файл находится в текущем каталоге. Если же имя указано в угловых скобках, то подразумевается, что файл находится в специальном каталоге.

В Си-системе существует целый ряд стандартных файлов. Обычно имена этих файлов снабжены суффиксом "h" (от английского слова "header"). Одним из наиболее часто используемых является файл "stdio.h", который содержит ряд определений, например, определено значение "EOF", и необходимые стандартные функции ввода/вывода данных. Простые функции ввода/вывода данных приведены в таблицах 1.1 и 1.2 .

Таблица 1.1. Функции ввода данных.

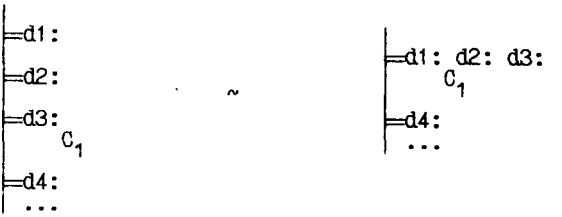
Имя	Назначение	Значение функции
getchar	Ввод следующего символа с экрана	Код введенного символа или EOF, если введено Ctrl+z
getint	Ввод целого числа с экрана	Введенное целое число
getfloat	Ввод с экрана числа с плавающей точкой	Введенное число (тип функции - "float")

Таблица 1.2. Функции вывода данных.

Имя	Аргумент(тип)	Назначение
putchar	Символ (char)	Вывод на экран символа, код которого задан в качестве аргумента
putint	Целое число (int)	Вывод на экран целого числа, которое задано аргументом
putfloat	Число с плавающей точкой (float)	Вывод на экран числа с плавающей точкой(6 мест после точки), которое задано аргументом
putbyte	Код символа (char)	Вывод на экран двоичного числа длиной 8
putxt	"строка" (*char)	Вывод данной строки на экран

В SKC-программе не используется строгий вход в схемах выбора. Строгий вход интерпретируется с помощью стрелок и нестрогого входа:

Разрешается уплотнять пустые варианты:



3. Основные типы данных и идентификаторы

В файле типы программных объектов, то есть переменных или функций, должны быть обязательно описаны. Только в случае целочисленного типа функции описание можно опустить, как чаще всего и поступают.При использовании стандартных файлов нет надобности описывать типы функций, входящих в данный файл. Например, не надо описывать тип функции "getfloat", так как соответствующее описание уже находится в файле "stdio.h". Один и тот же объект нельзя описывать несколько раз.

В простом случае переменные описываются строкой:

_тип _список_переменных .

При этом каждой из перечисленных переменных выделяется определенный объем памяти, размер которого зависит от указанного типа. В таблице 1.3 перечислены основные типы и для каждого типа приведены примеры констант.

Таблица 1.3. Основные типы.

Тип	Имя	Размер (в битах)	Диапазон	Константы
символ	char	8	код символа (-128..127)	'A', '+', '.'
целое число	int	16	целое число (-32768..32767)	1, -45, 654 0xF1, 0137
число с плавающей точкой	float	32	вещественное число (до 6 знаков)	3.14, -0.5, 1.98564e-4
число с двойной точностью	double	64	вещественное число (до 12 знаков)	8.4876396 -0.56398776

Целые константы могут быть:

1) десятичными: цифры 0-9, первой цифрой не должен быть нуль;

2) восьмеричными: цифры 0-7, начинаются с нуля (011=73, 012=10);

3) шестнадцатеричными: цифры 0-9, буквы a-f, A-F для значений 10-15, начинаются с 0x или 0X (0x12=18, 0x2f=47).

Имена переменных и функций должны удовлетворять определенным требованиям. В качестве имен переменных, функций и типов данных используются идентификаторы. Первый символ идентификатора не может быть цифрой. Допустимыми символами являются цифры 0-9, латинские прописные и строчные буквы a-z, A-Z, символ подчеркивания (_). Идентификатор может быть произвольной длины (до 31 символа), но не все символы учитываются. Внешние идентификаторы могут состоять из шести символов.

Для обозначения программных объектов не разрешено использовать ключевые слова:

auto, break, case, char, const, do, double, else, enum, entry, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while.

Примеры описания переменных:

```
int i;              --- переменная i целого типа;
int i, k;           --- переменные i, k целого типа;
char c, st_r;       --- переменные c, st_r символьного
                        типа;
float x, y0, z;     --- переменные x, y0, z - числа с
                        плавающей точкой;
double w;           --- переменная w - число с двойной
                        точностью.
```

Возможно использовать и следующие типы:

```
unsigned char           (0..255),
unsigned int            (0..65535),
long int (или long)     (-2147483648..2147483649),
unsigned long int       (0..4294967296),
long double             (вещественное число до 24
                         знаков).
```

4. Выражения

Центральным понятием языка Си является понятие выражения. Определим его пока как инструкцию для вычисления значений. Величины, которые встречаются в выражениях, называются операндами. Для вычисления новых значений используются выражения, которые строятся из знаков операций, скобок и операндов. Простейший случай выражения - один операнд. Например:

1) константа - значением выражения является значение константы;

2) переменная - значением выражения является значение переменной;

3) функция - значением выражения является найденное значение функции.

Сложные выражения получаются объединением простейших. Таким образом, выражение состоит из одного или большего числа операндов и символов операций.

Операции имеют приоритеты. Поэтому для того, чтобы создавать синтаксически и логически правильные программы, необходимо знать, как записываются выражения и как они интерпретируются, необходимо знать старшинство операций и порядок, в котором они выполняются. Для явного указания порядка операций можно пользоваться круглыми скобками.

В языке Си операция присваивания, символ которой " = ", обладает существенной особенностью. Кроме того, что она выполняет присваивание, в качестве результата она поставляет присвоенное значение, которое можно снова использовать в других целях. Поэтому запись

```
y = (x = 1) + 7;
```

является корректной. Переменной x присваивается значение 1, а переменной y - значение 8. Вполне допустимо писать и кратные присваивания:

```
x = y = z = 0; .
```

По сравнению с другими языками программирования язык Си позволяет использовать некоторые своеобразные знаки присваивания. Например, присваивания в левом столбце удобнее заменить на соответствующие присваивания, указанные в правом

столбце:

```
i = i + 1;          i++; ( или ++i;)
j = j - 1;          j--; ( или --j;)
x = x - y;          x -= y;
k = k / 2;          k /= 2;
z = z * 4;          z *= 4;
```

Отметим, что выражения "i++" и "++i" имеют разные значения. В первом случае значением выражения является значение i ДО увеличения, во втором - значение i ПОСЛЕ увеличения, то есть его новое значение. Аналогично определяются и значения "j--" и "j++".

При сравнении двух значений выражений используется знак " == " (не " = "). Побитовые операции сдвига не могут производиться над вещественными операндами. Значением выражений, содержащих операции отношения или логические операции, является 0 (Ложь) или 1 (Истина). Знаки бинарных операций при написании выражений лучше слева и справа отделять пробелами.

4.1. Бинарные операции

Бинарная операция - операция с двумя операндами.

В используемых ниже выражениях через a и b обозначаются операнды. Операции расположены последовательно по убыванию приоритета. Горизонтальными сплошными линиями в отдельные группы выделяются операции, имеющие одинаковый приоритет.

Умножение и деление:

a * b	произведение a и b;
a / b	деление a и b (без остатка, если a и b - целые числа);
a % b	остаток от деления a на b (a и b должны быть целыми числами).

Сложение и вычитание:

a + b	сумма a и b;
a - b	разность a и b.

Сдвиг:

a << b	двоичное представление a сдвигается влево на b разрядов, освободившиеся справа разряды заполняются нулями;
a >> b	двоичное представление a сдвигается вправо на b разрядов, освободившиеся слева разряды заполняются нулями в случае положительного a и единицами, если a < 0.

Сравнение:

a < b	значение выражения равно 1, если a < b, иначе 0;
a > b	значение выражения равно 1, если a > b, иначе 0;
a <= b	значение выражения равно 1, если a ⩽ b, иначе 0;
a >= b	значение выражения равно 1, если a ⩾ b, иначе 0;

Сравнение:

a == b	1, если a = b, иначе 0;
a != b	1, если a ≠ b, иначе 0.

Побитовая логическая операция И:

a & b	значение выражения содержит 1 во всех разрядах, в которых a и b содержат 1, и 0 во всех остальных разрядах.

Побитовая операция исключающее ИЛИ:

a ^ b	содержит 1 только в тех разрядах, в которых a и b содержат разные двоичные значения.

Побитовая логическая операция ИЛИ:

a | b	содержит 1 во всех разрядах, в которых a или b содержит 1, и 0 во всех остальных разрядах

Логическая операция И:

a && b	вначале проверяется значение a, значение b проверяется в том случае, если значение a≠0, значением выражения является 1, если значения a и b ≠ 0 (иначе 0).

Логическая операция ИЛИ:

a || b	вначале проверяется значение a, значение b проверяется в случае, если a=0, значением выражения является 0, если a и b = 0 (иначе 1).

Присваивание:

	присваивается значение переменной a;
a = b	значением выражения является b;
a ⊕= b	значением выражения является a⊕b, где ⊕ ∈ { +, -, %, *, /, <<, >>, &, |, ^ }.

Запятая:	
a, b	вначале вычисляется значение a, затем b. значением выражения является b.

Как правило, бинарные операции выполняются слева направо, однако операция присваивания происходит справа налево.

4.2. Унарные операции

Все унарные операции имеют один и тот же приоритет. По отношению к бинарным операциям их приоритет выше.

Отрицание:	
!a	0, если a ≠ 0 и 1, если a = 0.
Побитовое отрицание:	
~a	дополнение до единицы значения a, значение выражения содержит 1 во всех разрядах, в которых a содержит 0, и 0 во всех разрядах, в которых a содержит 1.
Изменение на единицу:	
++a	a += 1;
a++	значение a до увеличения, выполняется a+=1;
--a	a -= 1;
a--	значение a до уменьшения, выполняется a-=1.
Изменение знака:	
-a	изменение знака a.
Преобразование типа:	
(ТИП)a	значение a, преобразованное в тип данных ТИП.
Адрес объекта:	
&a	значением выражения есть адрес a (a должно быть переменной или указателем адреса).
Обращение по адресу:	
*a	значением выражения есть переменная, адресуемая указателем a.
Определение размера в байтах:	
sizeof a	число байт, требуемых для размещения объекта a;
sizeof(ТИП)	число байт, требуемых для размещения объекта типа ТИП.

4.3. Условные выражения

Рассмотрим выражения a, b и c. Выражением будет считаться и запись:

```
c ? a : b
```

Значение этого выражения есть a, если $c \neq 0$, и b, если $c = 0$.

Например, функцию

$$sign(a) = \begin{cases} 1, & \text{если } a > 0 \\ -1, & \text{если } a < 0 \\ 0, & \text{если } a = 0 \end{cases}$$

можно задать выражением

```
(a > 0) ? 1 : ((a < 0) ? -1 : 0) .
```

Но если учитывать, что значением выражения может быть 0, если выражение ложно, и 1, если истинно, то данную функцию можно записать еще более компактно:

```
(a < 0) ? -1 : a > 0 .
```

4.4. Преобразование типов

Преобразование типов понадобится в тех случаях, когда нужно "смешивать" в выражениях различные типы и требуется знать тип результата. Имена типов удобно перечислять в таком порядке:

```
char, int, long, float, double.
```

В этом списке типы представлены в порядке увеличения их "размера памяти". Речь идет об объеме памяти, необходимой для хранения значения некоторого типа. Этот список помогает освоиться с правилами неявного преобразования типов, приведенными ниже.

При вычислении выражений используются только целочисленные значения (int) и значения с двойной точностью (double). Для выражений, включающих одну бинарную операцию, например,

```
a + b ,
```

тип результата определяется по типам операндов. Действует

такое правило: значения типа char преобразуются в int, а float преобразуются в double. Если в результате преобразования любой из операндов (a или b) оказывается типа double, то второй также преобразуется в double. После этого либо оба операнда будут типа double, в таком случае результат будет типа double, либо один или оба операнда будут типа int или long. Если один из операндов - long, то другой преобразуется к long и результат - long. Если оба операнда - int, то результат - int. Поэтому можно говорить, что неявные преобразования всегда идут от "меньших" объектов к "большим" [3]. Результаты преобразований типов суммированы в таблице 1.4.

Таблица 1.4. Результаты преобразований типов.

a	b	Результат
char int	char int	int
char int	long	long
char int	float double	double
long	long	long
long	float double	double
float double	float double	double

4.5. Элементарные компоненты

Элементарная инструкция - описание или выражение, заканчивающееся точкой с запятой. В одну строку можно записывать несколько элементарных инструкций.

Условие записывается в отдельной строке и имеет вид:

выражение ? .

Условие удовлетворяется (истинно), если выражение не равно нулю, и не удовлетворяется (ложно), если выражение равно нулю.

Длина строки, а следовательно, и длина элементарных компонент и заголовков, в SKC-программе ограничена шириной экрана. Перевод инструкции допускается, но только в том случае, если не нарушается баланс парных символов (скобки, апострофы) в экранной строке.

5. Последовательности

Последовательность - одномерный массив, состоящий из элементов, каждый из которых имеет порядковый номер, называемый индексом элемента. Все элементы массива имеют один и тот же тип.

Последовательность описывается строкой:

```
тип имя_массива [число_элементов];
```

Число элементов или длина последовательности задается константой. В языке Си индексация начинается с нуля. Поэтому описание

```
float a[10];
```

резервирует память для вещественных переменных

```
a[0], a[1], ... ,a[9].
```

6. Связь между функциями через глобальные переменные

Как было сказано выше (см. п.2), любая программа на языке Си может состоять из нескольких функций. Выполнение программы всегда начинается с функции с именем "main". Для

передачи информации между функциями могут использоваться глобальные переменные.

Глобальные переменные описываются перед всеми функциями и доступны любой функции файла программы. Глобальные переменные в обмене информацией между функциями играют роль общей "полки", откуда функции получают исходные данные и куда возвращают результаты своей работы. Такой способ называется "неявной" передачей информации в отличие от другой, "явной", передачи через параметры функций, которая рассматривается в следующем пункте.

7. Связь между функциями через параметры

Функция может получить исходные данные через свои параметры. Для этого в описании функции перечисляются формальные параметры. В теле функции все действия производятся с этими параметрами. Параметры являются по существу локальными переменными, но отличаются от них тем, что при обращении к функции им присваиваются значения соответствующих фактических параметров (аргументов при вызове функции).

Конечно, возможно использование связи через глобальные переменные и через параметры в одной функции. Основное различие в том, что в функции значение глобальной переменной может изменяться. При использовании параметров исходные значения остаются неизменными. При этом результат работы функции можно получить через возвращаемое значение.

8. Значения функций

Функции можно описать так, чтобы всякий раз при их вызове получать и значения функций. Для передачи значения используется оператор возврата. В этом случае работа функции заканчивается выполнением выходной стрелки. Соответствующее значение, следующее в скобках за стрелкой, считается значением функции. Например, описание:

```
float y(x)
float x;
┌
│ x > 0 ?
│
└──(x)  положительный аргумент остается и значением
   --- функции
  x < 0 ?
└──(-x) значение функции в случае отрицательного
   --- аргумента
└──(0)  если аргумент равен нулю
└
```

соответствует описанию функции, которое выдает абсолютное значение аргумента:

$$y = \begin{cases} x, & \text{если } x > 0 \\ -x, & \text{если } x < 0 \\ 0, & \text{если } x = 0 \end{cases}$$

Решение этой задачи можно записать и более компактно:

```
float y(x)
float x;
┌
└──((x > 0) ? x : ((x < 0) ? -x : 0))
└
```

В практической работе используется уже существующая стандартная функция "fabs" нахождения абсолютной величины вещественного числа, которая описана в файле "math.h".

Но создаются и используются многие функции, значения которых не существенны или не определены, например, значение функции "putxt" (функция вывода строки на экран). В случаях, когда в теле функции не предусматривается возврат значения выражения, тип функции обозначается через "void". Например, из описания:

```
-------------------------
void p(x)
int x;
{...
-------------------------.
```

следует, что функции с именем p передается в качестве аргумента целое число x, но сама функция никакого значения не возвращает.

При использовании функций, возвращающих значения, необходимо заботиться о том, чтобы их тип был заранее описан (до вызова функции). Тип функции описывается предложением:

_тип _имя_функции(); .

Разрешается опускать описания функций целочисленного типа.

В функциях, оперирующих с вещественными числами, параметры желательно описывать типом "double", не "float".

9. Цикл с параметрами

Цикл с параметрами - общий случай огражденного цикла. Счет числа циклов требует трех "служебных" действий: инициации счетчика, увеличения счетчика и проверки, не достигнуто ли конечное значение. Естественно эти три действия соединить вместе. В качестве обобщения предлагается Си-цикл, заголовок которого состоит из трех выражений, отделенных друг от друга точкой с запятой:

```
* __выражение_1;__выражение_2;__выражение_3
C
```

где, например,

выражение_1 - инициация счетчика,

выражение_2 - условие продолжения счета,

выражение_3 - увеличение счетчика.

Любое из трех или все три выражения в цикле могут отсутствовать, но разделяющие их точки с запятыми опускать нельзя.

Семантика Си-цикла определяется с помощью неограниченного цикла:

```
__выражение_1;          ---инициация
╔═
│__выражение_2 ?        ---проверка параметра(ов)
    C
│__выражение_3;         ---изменение параметра(ов)
╚═
```

Си-цикл используется в случаях, когда:

1) в заголовке параметр - вещественное число:

```
┌
│* x = a; x <= b; x += a
│
│ C
└                              ;
```

2) скорость критична:

```
┌
│* i = 1; i <= n; n++
│
│ C
└                              ;
```

3) отсутствует ограничение параметра:

```
┌
│* i = 1; ; i++
│
│ C
└                              ;
```

4) имеется несколько параметров:

```
┌
│* i = 1, j = n; i <= n/2; i++, j--
│
│ C
└                              ,
```

(каждое из трех выражений в заголовке может состоять из нескольких выражений, объединенных оператором запятая).

10. Однострочные схемы

Простые схемы

```
┌
│t ?
│
│ оператор_1;..;оператор_n;
└
```

```
┌
│t ?
│ оператор_1;..;оператор_n;
│ ←
│
│ оператор_1';..;оператор_n';
└
```

и огражденный цикл

```
* p
оператор_1;..;оператор_n;
```

можно записывать в одну строку, соответственно:

```
?(t) оператор_1;..;оператор_n;
```

```
?(t) оператор_1;..;оператор_n;: оператор_1';..;оператор_n';
```

```
*(p) оператор_1;..;оператор_n;
```

11. Форматированный вывод

Функция вывода данных "printf" в пункте 2 применялась в простейшем варианте. Печатая строки текста, эта функция преобразует внутреннее представление величин в вид, подходящий для печати. Общий вид обращения к функции можно представить так:

printf(_управляющая строка __,список аргументов); .

Управляющая строка может содержать символы, которые следует напечатать, управляющие символы, перед которыми стоит обратная косая черта (\), и спецификации преобразования.

Каждой спецификации преобразования должен ставиться в соответствие некоторый аргумент из списка параметров. "Минимальный" вид спецификации - знак процента, за которым следует один из нескольких вполне определенных символов.

Общий вид спецификации таков:

%[flag][width][.prec][long]type .

Квадратными скобками здесь выделены элементы, которых может и не быть.

Обозначения:

type - символ преобразования (см. табл. 1.5);

flag - указатель на способ выдачи (см. табл. 1.6);

width- строка цифр (число), задающая минимальный размер поля, или '*' , если следующий аргумент в списке аргументов есть размер;

Таблица 1.5. Символы преобразования.

Символы преобразования	Тип аргумента	Преобразование
c	char	Единственный символ
d или i	int	Десятичное со знаком
u	int	Десятичное без знака
o	int	Восьмеричное без знака
x или X	int	Шестнадцатеричное без знака (используется a-f или A-F, соответственно)
f	float, double	Десятичное со знаком в виде [-]dddd.ddd
e или E	float, double	Десятичное со знаком в виде [-]d.ddd e [+/-]ddd или [-]d.ddd E [+/-]ddd
g или G	float, double	Наиболее короткое представление из e (или E) и %f
s	char *	Аргумент выводится как строка

Таблица 1.6. Указатели на способ выдачи.

Указатели	Способ выдачи
отсутствует	Выравнивание вправо
-	Выравнивание влево
+	Всегда указывается знак + или - вначале
пробел	Указывается знак только для отрицательных
#	В преобразовании используется альтернативная форма:
	c, s, d, i, u — не используется
	o — перед ненулевыми ставится 0
	x или X — впереди добавляется 0x или 0X
	e, E, f — гарантируется вывод десятичной точки
	g или G — также как ранее, но нет замыкающих нулей

.prec- строка цифр, указывающая в преобразованиях e, E, f, g, G, сколько цифр напечатается после десятичной точки, если речь идет о строках, то задает число печатаемых из строки символов. Вместо цифр может использоваться '*', если следующий аргумент в списке аргументов есть размер;

long - спецификация размера аргумента:
h - в преобразованиях d, i, o, u, x, X аргумент есть short int;

l - в преобразованиях d, i, o, u, x, X аргумент есть long int;

L - в преобразованиях e, E, f, g, G аргумент есть long double;

Список аргументов состоит из разделенных запятыми выражений, значения которых надо вывести. Например, если переменным

```
int pos;
double sumpos;
```

присвоить значения 9 и 185.46, то оператором

```
printf("\nЧисло элементов %d;\nих сумма =%f.", pos, sumpos);
```

выведется на экран сообщение:

```
Число элементов 9;
их сумма =185.460000.
```

Если не использовать функцию "printf", а довольствоваться ранее рассмотренными функциями вывода, то тот же результат можно получить следующим образом:

```
putxt("\nЧисло элементов ");putint(pos);putxt(";");
putxt("\nих сумма =");putfloat(sumpos);putxt(".");
```

Для того, чтобы программа выдавала информацию в удобном виде, необходимо свободно манипулировать всеми перечисленными возможностями при обращении к функции "printf". Лучший способ освоиться со всем их разнообразием - это использовать их в реальных условиях и "экспериментировать" с различными спецификациями преобразования. В этом помогут и примеры, сосредоточенные в таблице 1.7 [3].

Таблица 1.7. Примеры форматированного вывода.

Значение	Преобразование	Выдача
360	%10d	360
360	%-10d	360
360	%-10x	168
-1	%-101x	ffffffff
360	%10o	550
3.14159265	%10f	3.141593
3.14159265	%10.3f	3.142
3.14159265	%10.0f	3
programmer	%10.7s	program
programmer	%-10.7s	program

12. Форматированный ввод

В языке Си существует функция "scanf", которая является симметричной к "printf", то есть построенная аналогичным образом. Общий вид обращения к "scanf" выглядит так:

scanf(_управляющая строка __,список аргументов); .

Внутри управляющей строки пробелы, символы табуляции и переходы на новую строку игнорируются. Если в управляющей строке появляется какой-либо символ, кроме тех, которые относятся к спецификации преобразования, то считается, что он должен совпадать с первым непустым символом из входного потока. *Примеры спецификаций преобразования приводятся в* таблице 1.8.

Таблица 1.8. Примеры спецификаций преобразования.

Символы преобразования	Тип аргумента
c	Ссылка на char
d, i, o, u, x, X	Ссылка на int
e, E, f, g, G	Ссылка на float
s	Ссылка на массив char

Для функции "scanf" общий вид спецификации преобразования следующий:

%[*][width][long]type .

Обозначения:

type - символ преобразования;

* - допустимый признак гашения присваивания;

width- цифровая строка, задающая максимальный размер поля.

long - спецификация размера аргумента:

h - в преобразованиях d, i, o, u, x, X аргумент есть ссылка на short int;

l - в преобразованиях d, i, o, u, x, X аргумент есть ссылка на long int;

l - в преобразованиях e, E, f, g, G аргумент есть ссылка на double;

L - в преобразованиях e, E, f, g, G аргумент есть ссылка на long double;

Функция "scanf" возвращает в обратившуюся программу число опознанных и присвоенных элементов данных. Если уже первый символ входной строки не совпадает с первым из элементов управляющей строки, то выдается значение нуль. По достижению же конца файла, выдается значение EOF [3].

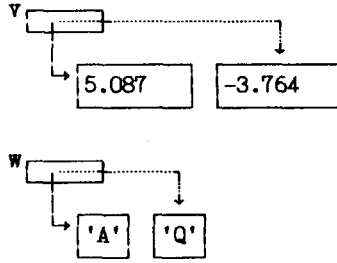
Простейшие функции ввода чисел getint() и getfloat() реализованы именно на основе функции "scanf" :

```
-----------------------------------------
getint()
┌
| int x;
|
|  scanf("%d", &x); ---&x - адрес x
←—(x)
└
-----------------------------------------
-----------------------------------------
float getfloat()
┌
| float x ;
|
|  scanf("%f", &x);
←—(x)
└
-----------------------------------------.
```

(Понятие адреса переменной объясняется в следующем пункте).

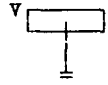
13. Адреса и ссылки

Адрес данных - выражение, значением которого является адрес памяти, где находятся данные, которыми манипулирует программа. В языке программирования Си предусмотрен легкий и удобный способ получения адресов данных. И, соответственно, если есть адрес данных, то легко получить их значение. Механизм этот прост и "короток". Адрес любого элемента данных получается при указании перед именем префикса & (амперсенда). То есть, если x - какая-либо описанная в программе переменная, то &x - адрес этой переменной во время выполнения программы. Для манипуляции адресами надо иметь возможность определять величины, значения которых - ссылки или адреса. Каждая ссылка связана с определенным типом. Например, если ссылка v указывает на переменную вещественного типа, а w - на символьную переменную, то значением v может быть адрес размещения вещественного числа в памяти, а значением w - адрес символа:



Различия между v и w, связанными с разными типами, видны прежде всего при операции изменения их значений: "v++;" означает изменение v на длину вещественного числа (4 байта), а "w++;" меняет значение w на 1. В примере, приведенном выше, соответствующие новые значения v и w обозначены пунктирной стрелкой.

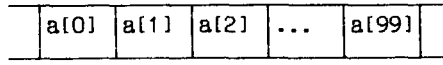
Ссылкам может быть присвоено специальное значение, которое обозначается через NULL и является пустой ссылкой. Она не указывает на какой-либо объект. Случай, когда v==NULL, изображен на рисунке:



Фактически, ссылки тесно связаны с массивами. Так, в случае описания

```
float a[100];
```

в памяти резервируется место для ста вещественных чисел



В этом случае вместо выражения &(a[0]), определяющего адрес первого элемента массива, допускается использование просто имени a.

Для описания ссылок применяется специально предусмотренный символ (*). Описания

```
float *v;
char  *w;
```

совершенно индентичны описаниям v[] и w[]. В этом случае ссылки v и w можно использовать вместе с индексами:

```
w[1] = 'Q';  .
```

К ссылкам можно применять унарную операцию *:

```
*(w + 1) = 'Q'; ---то же самое, что и w[1] = 'Q';
putfloat(*v);   ---вывод вещественного числа с адресом v
```

С помощью введенных понятий можно обеспечить связь любой функции с вызвавшей ее программой. Ведь, если программа передает функции адреса некоторых величин, то именно эти адреса и сохраняются в локальной области функции. Адрес величины функция изменить не может, но она может изменить содержимое этого адреса, что и требуется.

14. Строки

Строка - последовательность символов. В памяти место для строки резервируется символьным массивом. Например, при описании

```
char s[80];
```

последовательность s[0]...s[79] может использоваться в качестве места для хранения строки. Отдельного типа для строк не существует. Поэтому все операции над строками описываются через операции над символами, конечно, используя при этом стандартные Си-функции. Признаком конца строки является символ '\0', то есть непосредственно за последним символом строки следующий байт должен быть нулевым (содержать нуль). Признак конца не считается "рабочим" символом. Он не выводится на печать, и длиной строки считается число байтов от ее начала до нулевого байта, не включая последнего. Следовательно, char s[80] описывает строку, максимальная длина которой 79 символов. Строка-константа - последовательность символов, заключенная в двойные кавычки. На самом деле, строка-константа в программе транслятором трактуется как ссылка на эту строку. Поэтому строку-константу можно записывать там, где допускается адрес символа (и наоборот). В

частности, описанная выше строка s печатается командой putxt(s). Подобно другим данным, строки можно читать, записывать и печатать. Для этого существует стандартный файл "string.h", в котором описаны функции обработки строк.

Примеры таких функций:

```
-----------------------------------------

strlen(s)           --- длина строки
char *s;
┌
│ int l;
│ ┌
│ │ * l = 0; ; l++
│ ├
│ │ s[l] == '\0' ?
└─┘
←──(l)
└┘
-----------------------------------------


-----------------------------------------

char *strcpy(s, t) ---копирование s <== t
char *s, *t;
┌
│ char *s0 = s;
│ ┌
│ │ *(s++) = *(t++) ?
│ └
└
←──(s0)
└
-----------------------------------------.
```

15. Структуры и объединения

При желании в программе целый комплект переменных разного типа можно рассматривать как один объект. Например, может оказаться целесообразным положение точки на плоскости задавать и рассматривать тройкой чисел, состоящей из двух вещественных координат и одного целого числа, соответствующего номеру четверти, в которой находится точка. В таких случаях используют определенный тип данных - структуру.

Структура соединяет логически связанные данные разных типов. Структурный тип данных определяется следующим описанием:

```
struct _имя_структуры
{
  _описание_элемента_1;
  ...
  _описание_элемента_n;
};
```

Элементы структуры описываются как переменные. Например, описание

```
struct punkt
{
  float x;       --- абсцисса
  float y;       --- ордината
  int veerand;   --- номер четверти
};
```

определяет структуру с именем "punkt". В дальнейшем структурные переменные описываются с помощью структурного типа:

```
struct punkt A, B;
```

В памяти резервируется место для переменных A и B, каждое из которых состоит из трех элементов, соответственно:

```
A.x, A.y, A.veerand
```

и

```
B.x, B.y, B.veerand.
```

Существуют и другие способы описания структур. К примеру, структурные переменные A и B не обязательно описывать отдельно:

```
struct punkt
{
  float x;
  float y;
  int veerand;
} A, B;
```

Структура может входить в структуру:

```
struct kolmnurk          --- треугольник
{
                  --- вершины треугольника
  struct punkt A;
  struct punkt B;
  struct punkt C;
};
struct ring              --- окружность
{
  struct punkt keskpunkt;     --- центр
  float  raadius;             --- радиус
};
struct kolmnurk k;    --- треугольник k
struct ring r1, r2;      --- окружности
                         --- r1 и r2
```

Объединение описывает переменную, которая может иметь любой тип из некоторого множества типов. Определение объединенного типа данных аналогично определению структурного типа данных:

```
union _имя_объединения
{
  _описание_элемента_1;
  ...
  _описание_элемента_n;
};
```

Например:

```
union sone
{
  int x;
  struct
  {
    char c1;
    char c2;
  }
  y;
};
```

Данные типа union sone занимают память, необходимую для размещения наибольшего из своих элементов. Переменная объединенного типа описывается строкой:

```
union sone w;  .
```

Над структурами и объединениями можно производить операции [4]. Пусть sv - структура или объединение, spe - ссылка на структуру или объединение и smem - элемент структуры или объединения, тогда:

1) sv.smem - значением выражения является элемент smem структуры или объединения sv, например, r1.raadius ;

2) spe -> smem - значением выражения является элемент smem структуры (или объединения), на которую(ое) указывает spe, это значение эквивалентно значению выражения (*spe).smem, например, если описано

struct ring *p; ---p есть ссылка на объект типа ring ,
то справедливы выражения

p -> keskpunkt и (*p).keskpunkt .

16. Текстовый экран

Половина, если не две трети, привлекательности персональной ЭВМ заключается в возможностях визуального представления информации. Дисплейный адаптер представляет собой довольно сложное устройство, содержащее процессор и память. Процессору предоставляется возможность записывать информацию в адаптерную память (видеопамять), за которой закреплен определенный диапазон адресов. Процессор адаптера все время читает эту память и отображает прочитанное на экране монитора. Процессор, а значит и программа, может влиять на представленную на экране картинку двумя способами. Во-первых, он может записать любую информацию в видеопамять. Во-вторых, он имеет возможность изменить параметры схемы, которая формирует изображение. Иными словами, можно задавать различные режимы преобразования видеопамяти в картинку. Режимы пронумерованы. Текстовому режиму соответствуют номера от 0 до 3 и номер 7.

Рассмотрим третий режим. Именно этот режим обычно устанавливается сразу после включения машины.На экране можно изображать только символы. Размеры символов, места, в которых они могут изображаться, и сам набор изображаемых символов жестко заданы: экран поделен на 25 строк, в каждой стро-

ке может изображаться по 80 символов, то есть всего имеется 2000 мест для изображения символов. В видеопамяти каждому месту на экране соответствует два последовательных байта. (Следовательно, экран занимает 4 Кбайт.) В первом байте хранится код символа (возможно 256 различных символов). Информация о виде данного символа на экране, включающая цвет, фон и мигание, называется атрибутом символа и помещается во второй байт. На экране одновременно могут присутствовать символы с разными атрибутами. Разберемся теперь, как устроен байт атрибута символа. Для этого придется записать его в двоичной форме и разбить на три части:

B7	B6 B5 B4	B3 B2 B1 B0
мигание	цвет фона	цвет символа

Символ будет мигать, если B7 = 1. Цвет символа, как видим, может быть записан четырехразрядным двоичным числом, или, что то же самое, десятичным числом от 0 до 15:

0 черный, 4 красный, 8 темно-серый, 12 ярко-красный,
1 синий, 5 розовый, 9 ярко-синий, 13 ярко-розовый,
2 зеленый, 6 коричневый, 10 ярко-зеленый, 14 желтый,
3 голубой, 7 светло-серый, 11 ярко-голубой, 15 белый.

Так же определяется и цвет фона, только он может меняться лишь до 7.

Объем видеопамяти занимает 16 Кбайт. Так что для построения изображения в этом режиме одновременно требуется лишь четверть всей видеопамяти. Эту часть видеопамяти принято называть видимой страницей. Общее число страниц, следовательно, равно четырем. Страницы имеют номера от 0 до 3. Первоначально, при задании этого режима, в качестве видимой страницы выбирается страница номер 0. Содержимое остальных страниц при этом не влияет на вид экрана. Можно сделать видимой любую из страниц.

Режим 1 отличается от режима 3 вдвое меньшим количеством символов на экране, но зато вдвое большим количеством страниц: от 0 до 7.

Специальная часть схемы адаптера занимается изображением на экране курсора. Эта часть никак не связана с видеопамятью - изображение курсора и текст смешиваются лишь на экране [11].

17. Графический экран

Графическому режиму соответствуют номера от 4 до 6 и от 10 до 16. Рассмотрим режим 4. В этом режиме на экране изображается 200 строк точек, по 320 точек в каждой строке. При этом одновременно на экране могут быть изображены точки четырех цветов. В видеопамяти каждой экранной точке отводится два бита. Следовательно, изображение требует одновременно все 16 Кбайт видеопамяти.

Цвет каждой точки может меняться от 0 до 3. Сразу после включения режима 4 устанавливается такое соответствие:

0 - черный; 1 - голубой; 2 - розовый; 3 - белый.

Пользователю дана возможность заставить адаптер раскрашивать точки со значением 0 в 16 различных цветов. Второй вариант преобразования:

1 - зеленый; 2 - красный; 3 - коричневый.

Эти два варианта называются палитрами (с номерами 0 и 1, соответственно).

Рассмотрим другие графические режимы. Режим 6 отличается от режима 4 тем, что в каждой строке вместо 320 точек имеется 640. Достигается это тем , что каждая точка кодируется не двумя, а одним битом видеопамяти. Из-за этого число цветов сокращается до двух - черного и белого.

Команда "напечатать" ("printf") работает в любом из перечисленных режимов. При этом печать всегда начинается с позиции курсора, хотя в графических режимах он не виден. На экране (относительно курсора) по-прежнему 25 строк и в каждой строке умещается 40 (в 4-ом режиме) и 80 (в 6-ом режиме) символов [11].

18. Использование средств базовой системы ввода/вывода (BIOS)

Одним из "секретов" эффективного программирования на языке Си является максимальное использование возможностей окружающей среды - стандартного набора функций операционной системы. Эти функции собраны в специальный блок - "базовую систему ввода/вывода" (Английская аббревиатура "BIOS" означает: "Basic-Input-Output-System").

Операционная система имеет большое количество встроенных функций, к которым программист может обращаться из программы на языке Си. Обращение к функции операционной системы автоматически вызывает прерывание.

Прерывание - это некоторый сигнал, заставляющий компьютер прервать выполнение текущей программы и переключиться на выполнение другой, называемой подпрограммой обслуживания прерывания. Каждому прерыванию поставлен в соответствие код типа прерывания, представляющий собой беззнаковое целое число от 0 до 255 и определяющий переход на соответствующую подпрограмму обработки прерывания. По окончании работы подпрограммы прерывания, как правило, выполнение текущей программы продолжается.

Для организации прерываний в языке Си используется функция int86. Обращение к ней имеет следующий формат:

int86(into, inregs, outregs);

Здесь into - это код типа прерывания; inregs и outregs - ссылки на объединения входных и выходных параметров, соответственно. Через входные параметры, в частности, передается код запрашиваемой функции, через выходные - результат работы по прерыванию. Таким образом, за одним кодом прерывания может скрываться множество функций. Входные и выходные параметры объединены в inregs и outregs с помощью типа union REGS :

```
union REGS
{
  struct WORDREGS x;
  struct BYTEREGS h;
};
```

Структуры WORDREGS и BYTEREGS выглядят так:

```
struct WORDREGS
{
 unsigned int ax, bx, cx, dx, si, di, cflag, flags;
};
struct BYTEREGS
{
 unsigned char al, ah, bl, bh, cl, ch, dl, dh;
};
```

Прототип функции int86 и описания типов union REGS, struct WORDREGS, struct BYTEREGS содержатся в стандартном файле "dos.h".

Использование обращения к функциям BIOS через прерывания позволяет, в частности, управлять графическим экраном. Для этого удобно пользоваться вспомогательными функциями a86 и v86bh:

```
-------------------------------------

unsigned int a86( i, ah, al, cx, dx )
int i, ah, al, cx, dx;
{
 union REGS inr, outr;
 inr.h.ah = ah; inr.h.al = al;
 inr.x.bx = 0;
 inr.x.cx = cx; inr.x.dx = dx;
 int86( i, &inr, &outr);
<------(outr.x.ax)
}
-------------------------------------

unsigned int v86bh( ah, bh, bl)
int ah, bh, bl;
{
 union REGS inr, outr;
 inr.h.ah = ah;
 inr.h.bh = bh; inr.h.bl = bl;
 int86( 16, &inr, &outr);
<------(outr.h.bh)
}
-------------------------------------.
```

Последняя функция использует прерывание только с кодом 16, которое реализует графические возможности BIOS.

Теперь (зная коды прерываний и функций [11]) легко

создать необходимые средства управления экраном:

```
---------------------------------------------------------

---              запрос режима экрана                 ---
#define s_mode()      (a86(16, 15, 0, 0, 0) & 255)
---            установка режима экрана                ---
#define s_setmode(m) a86(16, 0, (m), 0, 0)
---               рисование точки                     ---
#define s_drawdot(c,l,clr)  a86(16, 12,(clr),(c),(l))
---              запрос цвета в точке                 ---
#define s_dot(c,l)  (a86(16, 13, 0, (c), (l)) & 255)
---              установка цвета фона                 ---
#define s_setback(n)  v86bh(11, 0, (n))
---               установка палитры                   ---
#define s_setpal(n)  v86bh(11, 1, (n))
---------------------------------------------------------.
```

Описанные определения функций управления экраном целесообразно поместить в отдельный файл с именем graphics. Тогда в программу, использующую графические средства, нужно включить нишу:

```
graphics
--- функции управления экраном
```

Используя эти элементарные средства, можно создать более сложные графические функции. Например, следующий фрагмент описывает рисование линии из отдельных точек (между точками (ax, ay) и (bx, by) цвета clr):

```
---------------------------------------------------------

s_drawline(ax, ay, bx, by, clr)
int ax, ay, bx, by, clr;
{
  int xdev = 0, ydev = 0, dx, dy, d, xstep, ystep, i;
  dx = bx - ax; dy = by - ay;
  xstep = (dx < 0)? -1 : (dx > 0);
  ystep = (dy < 0)? -1 : (dy > 0);
  dx = abs(dx); dy = abs(dy);
  d = (dx > dy)? dx : dy;
  {
  * d + 2
  }
   s_drawdot(ax, ay, clr);
   xdev += dx; ydev += dy;
   [? (xdev > d) xdev -= d; ax += xstep;
   [? (ydev > d) ydev -= d; ay += ystep;
  ==
}
---------------------------------------------------------.
```

19. Матрицы

Матрица - таблица или двумерный массив, состоящий из элементов одного и того же типа. Описывается строкой

```
_тип _a[m][n];
```

где m и n - натуральные постоянные величины. Описанием резервируется место в памяти для матрицы с именем a, в которой m строк, n столбцов и все элементы имеют указанный тип. Элемент матрицы a, находящийся в i-ой строке j-ом столбце, обозначается в программе через

```
a[i][j]
```

где i и j - целые числа. Первый индекс матрицы, то есть номер строки i, имеет значения от 0 до (m-1), а второй индекс, то есть номер столбца j, изменяется от 0 до (n-1):

```
a[0][0]    a[0][1] ... a[0][n-1]
a[1][0]    a[1][1] ... a[1][n-1]
...
a[m-1][0]  a[m-1][1]... a[m-1][n-1]
```

Выражения a и &(a[0][0]) имеют одинаковые значения, а именно, адрес первого элемента массива a. Адрес начала i-ой строки &(a[i][0]) можно записать через a[i]. Матрица располагается в памяти по строкам, благодаря чему элементы матрицы можно обрабатывать и последовательно. Например, если ссылке b, описанной через "char *b", присвоить значение a, то

```
*(b + 1) --- второй элемент матрицы, то есть a[0][1],
*(b + n) --- элемент a[1][0]
```

и так далее.

20. Файлы

Файл - определенная совокупность данных с общим именем, хранящаяся на внешнем носителе памяти (на магнитном диске). Текстовым файлом назовем файл, состоящий из последовательности символов, то есть из их кодов. Во время обработки файла в памяти хранится информация о нем, которую используют все функции, связанные с файлом. Подробно о структуре этой

информации программисту знать необязательно, так как достаточно использовать специальную ссылку на нее - ссылку на файл. Ссылка на файл - идентификатор, который описывается следующим образом:

```
FILE *_ссылка_на_файл;
```

Описание не должно находиться внутри какой-либо функции, если и другие функции ссылаются на этот (глобальный) файл. Но ссылка на файл может передаваться и в качестве параметра.

Для открытия файла используется функция "fopen":

```
_ссылка_на_файл = fopen(_имя_файла, _вид_файла); .
```

При этом происходит инициализация информации названного файла, и ее адрес присваивается ссылке на файл. Если файл не был открыт по какой-либо причине, то ссылке присваивается значение NULL. Поэтому при открытии файла целесообразно сразу же проверить, не имеет ли ссылка значение NULL. Пока файл открыт, пока идет обработка его данных, значение ссылки изменять нельзя. Если файл больше не требуется, то его надо закрыть. Это одна из "бытовых" обязанностей. Для этого существует функция "fclose". Обращение к ней выглядит следующим образом:

```
fclose(_ссылка_на_файл);  .
```

При нормальном окончании программы все открытые файлы закрываются автоматически.

При открытии файла аргумент _вид _файла - строка, указывающая, с какой целью открывается файл. Этот аргумент может иметь вид:

"r" - открыть файл для чтения;
"w" - создать файл и открыть для записи;
"a" - открыть файл для добавления;
"r+" - открыть файл для чтения и записи;
"w+" - создать файл и открыть для чтения и записи;
"a+" - открыть файл для чтения и добавления.

Функция чтения следующего элемента из файла вызывается следующей строкой:

```
fgetc(_ссылка_на_файл);  .
```

Выдается значение кода соответствующего символа или значение EOF, если файл больше не содержит символов, то есть достигнут конец файла.

Добавление символа в конец файла происходит при вызове функции:

fputc(_символ,_ссылка_на_файл);

Добавлять можно и целые слова, используя форматированный вывод:

fprintf(_ссылка_на_файл,_формат,_список_аргументов);

Эта функция определяется аналогично функции "printf". Формат - управляющая строка, на основе которой формируется последовательность выводимых функцией fprintf символов.

ГЛАВА 2. РАБОТА В СИСТЕМЕ SKC

1. Команды системы SKC

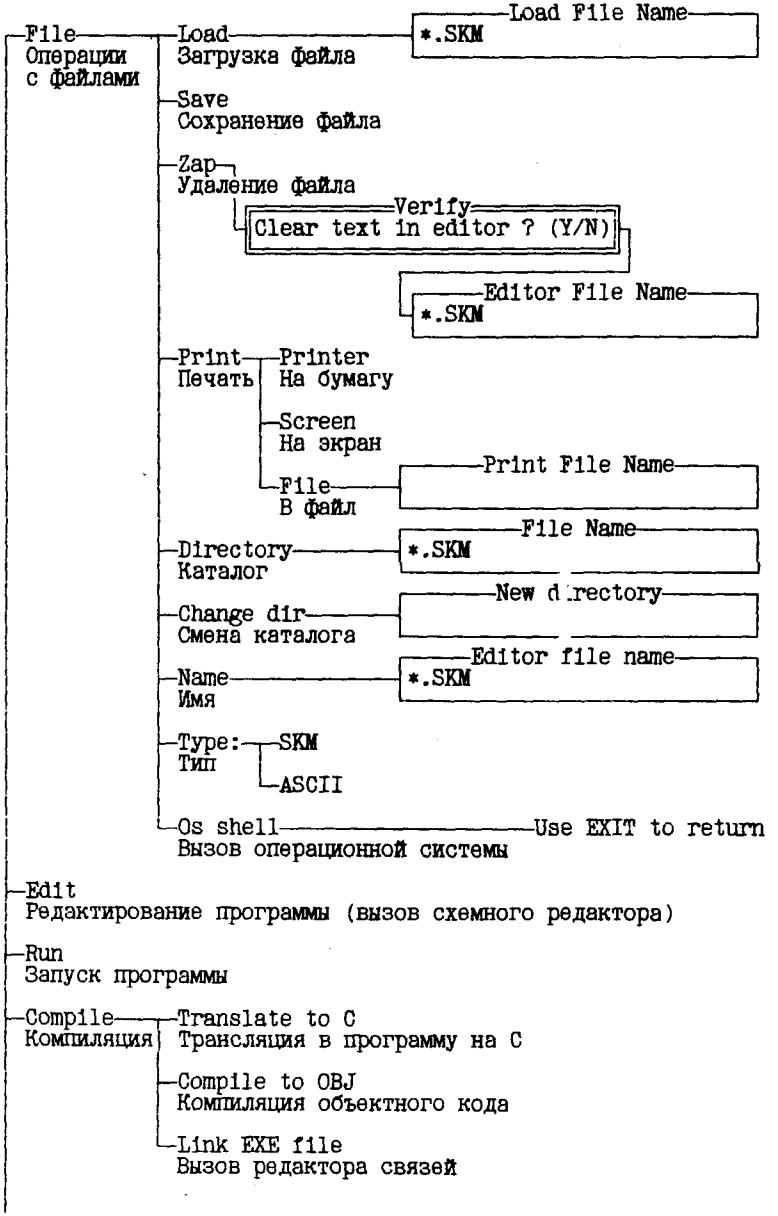
1.1 Меню команд

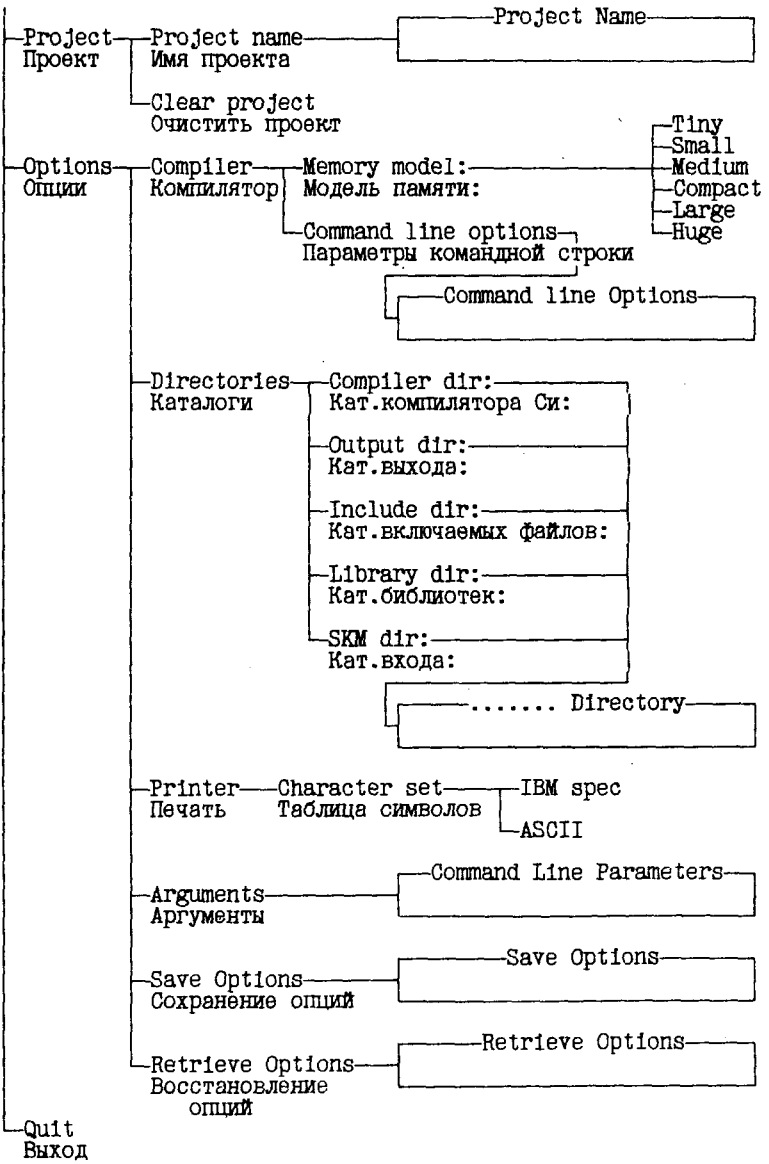
Вверху экрана находится меню команд. Вы получаете доступ к нему сразу после запуска программы. Тогда светлый прямоугольник выделит первую команду. Вы можете передвигать светлый прямоугольник клавишами передвижения курсора [Влево] и [Вправо]. Для выполнения команды, передвиньте светлый прямоугольник на нее и нажмите [Return]; вместо этого, Вы можете также нажать первую заглавную букву в имени команды. Имена команд более или менее искусно выбраны так, чтобы все начинались с разных букв. Например, есть два пути запуска команды "Run" :

- передвижением светлого прямоугольника поверх команды "Run" и нажатием [Return];
- нажатием [R].

Систему меню, такую как в SKC, часто называют деревом меню. После запуска программы Вы попадаете в корень дерева, от которого отходят несколько ветвей ("File", "Edit", "Run", ...). Следуя по одной из ветвей, Вы получаете еще один указатель на ветвь и, в конце концов, получаете лист дерева, то есть фактически команду (такие как "Save", "Os shell"). Пройти через дерево меню к выполнению любой команды Вы можете передвигая (с помощью клавиш управления курсором) светлый прямоугольник на нужную ветвь и переходя на один уровень ниже в структуре дерева клавишей [Return]. Клавиша [Esc] возвращает Вас на один уровень в дереве выше.

Ниже предлагается система меню, преобразованная в дерево. Названия команд системы SKC приведены на английском языке и сопровождаются комментариями на русском языке.


File
Операции с файлами
Load
Загрузка файла
Load File Name
*.SKM
Save
Сохранение файла
Zap
Удаление файла
Verify
Clear text in editor ? (Y/N)
Editor File Name
*.SKM
Print
Печать
Printer
На бумагу
Screen
На экран
File
В файл
Print File Name
Directory
Каталог
File Name
*.SKM
Change dir
Смена каталога
New directory
Name
Имя
Editor file name
*.SKM
Type:
Тип
SKM
ASCII
Os shell
Вызов операционной системы
Use EXIT to return
Edit
Редактирование программы (вызов схемного редактора)
Run
Запуск программы
Compile
Компиляция
Translate to C
Трансляция в программу на С
Compile to OBJ
Компиляция объектного кода
Link EXE file
Вызов редактора связей


Project
Проект
Project name
Имя проекта
Project Name
Clear project
Очистить проект
Options
Опции
Compiler
Компилятор
Memory model:
Модель памяти:
Tiny
Small
Medium
Compact
Large
Huge
Command line options
Параметры командной строки
Command line Options
Directories
Каталоги
Compiler dir:
Кат.компилятора Си:
Output dir:
Кат.выхода:
Include dir:
Кат.включаемых файлов:
Library dir:
Кат.библиотек:
SKM dir:
Кат.входа:
..... Directory
Printer
Печать
Character set
Таблица символов
IBM spec
ASCII
Arguments
Аргументы
Command Line Parameters
Save Options
Сохранение опций
Save Options
Retrieve Options
Восстановление опций
Retrieve Options
Quit
Выход

1.2 Как начать писать программу

Для того, чтобы начать создавать программу, Вы должны вызвать схемный редактор. Доступ в схемный редактор из меню команд обеспечивает команда "Edit". Для возврата в меню команд используется клавиша [Esc]. Описание схемного редактора приведено ниже, в пункте 2.

1.3 Как вызвать ранее созданную программу

Если Вы хотите использовать уже готовую или закончить ранее начатую программу, то выберите через меню команд команду "File Load". Система SKC предложит Вам ввести имя программы. Допустимо использовать в имени символ "*", который заменяет неизвестные символы. В этом случае система предложит Вам выбрать вызываемую программу из тех, которые она сможет найти.

1.4 Как ввести имя программы

Ввод имени программы осуществляется по команде "File Name". Система SKC предложит Вам ввести имя. По этой команде можно и переименовать любую программу. Имя программе необходимо присваивать, так как под этим именем она будет храниться на диске, и по имени ее можно будет потом вызвать. Как правило, расширение .SKM можно опускать.

1.5 Как перейти к другой программе

Часто бывает, что за один сеанс работы на машине создаются несколько программ. Чтобы перейти к другой программе, следует сохранить предыдущую на диске по команде "File Save". Затем используйте команду "File Zap". Система SKC спросит у Вас разрешения на удаление из оперативной памяти предыдущей программы и, если Вы ответите утвердительно ("Y"), предложит Вам ввести имя новой программы.

1.6 Просмотр каталога

Иногда требуется просмотреть содержимое текущего каталога диска. Для этого используйте команду "File Directory". Система SKC предложит Вам ввести имя файла, который Вас интересует. Для просмотра всего каталога введите имя "*.*". Заменить текущий каталог можно по команде "File Change dir". В этом случае Вы получите предложение ввести имя нового каталога.

1.7 Распечатка текста программы

Готовую программу Вы можете распечатать на имеющемся в комплекте с компьютером принтере (печатающем устройстве). Выберите в меню команд команду "File Print Printer". Не забудьте предварительно вставить в принтер лист чистой бумаги!

1.8 Компиляция и запуск программы

Когда Вам кажется, что программа уже готова, попробуйте запустить ее по команде "Run". Система SKC проделает последовательно четыре операции:

- транслирует схематическую программу в программу на языке C;
- вызывает компилятор, который создаст из программы на языке C файл с объектным кодом;
- вызывает сборщик, который создаст из файла с объектным кодом выполняемую программу;
- запускает полученную выполняемую программу.

Первые три операции Вы можете проделать поэтапно с помощью команд (соответственно):

- "Compile Translate to C";
- "Compile Compile to OBJ";
- "Compile Link EXE file".

1.9 Как сохранить программу и выйти из системы SKC

Для сохранения Вашей программы в файле на диске используйте команду "File Save". Она сохранит программу в том же файле, откуда она была прочитана. В этом случае запасная копия оригинальной программы будет создана с расширением ".BAK". Запасная копия может реально пригодиться, когда непредвиденно случится "несчастье" с Вашей программой. Вы могли бы, следовательно, не записывать программу дважды "для большей уверенности". Если Вы все-таки записали дважды, то Вы имеете на диске текст программы и идентичную ему копию.

Если Вы нуждаетесь в записи текста программы в различные файлы, Вы можете изменить имя файла перед записью с помощью команды "File Name".

Выберите команду "Quit" для выхода из системы SKC и возврата в DOS. Система SKC проверит, не изменяли ли Вы программу после последнего ее сохранения. Если нет, то система сразу выйдет в DOS. Если же Вы делали изменения, то система SKC предложит сначала сохранить программу.

2. Схемный редактор

Доступ в схемный редактор из меню команд обеспечивает команда "Edit". Для возврата в меню команд используется клавиша [Esc].

Схемный редактор разработан в духе известных текстовых редакторов (WordStar, TurboPascal и др.), но существенной отличительной чертой его является структурность, ориентированность на работу со схемами. Например, пустая схема порождается и уничтожается только целиком. Поэлементно, по строкам и символам, можно редактировать лишь входящие в схему тексты - элементарные инструкции, заголовки, условия, комментарии.

Схемный редактор допускает определенные отклонения от стандарта при изображении графических элементов на экране. Например, по стандарту такой элемент схемы как условие выглядит следующим образом:

B ? ;

на экране же он изображается

? B .

Существуют различия и в изображении сильного и слабого терминаторов, ниши, неогражденного и огражденного циклов.

Строгий вход схемы выбора не предусмотрен. Длина строки обрабатываемой программы ограничивается шириной экрана.

2.1. Назначение функциональных клавиш

Основные "схемные" операции редактора присвоены функциональным клавишам F2, F3, ..., F10. В таблице 2.1 приведена сводка этих операций, причем под уровнем 0 подразумевается область вне схем. Как правило, нажатие на функциональную клавишу порождает новый объект, который вставляется после текущей строки (где находится курсор). Исключение составляют клавиши F7, F8, изменяющие лишь статус текущей строки.

Выходная стрелка (терминатор) порождается клавишей F9, последующие нажатия на F9 увеличивают длину стрелки. Для уменьшения протяженности стрелки предусмотрена клавиша "стирание слева".

В общем для уничтожения объектов используется пара Ctrl+Y. В частности, таким способом удаляется и схема, но перед этим ее необходимо опустошить.

Функциональная клавиша F1 вызывает на экран меню-справочник, который содержит краткий обзор всех операций.

При выделении частей схематической программы в качестве блоков не допускается нарушение целостности схем. Выделенный блок членов C может быть превращен в простую схему, простая схема - в неогражденный цикл и неогражденный цикл - в последовательность членов C. Каждый из этих переходов осуществляется нажатием ^KN или Alt+F2 [9].

Таблица 2.1. Назначение функциональных клавиш.

Клавиша	Допустимое местонахождение курсора	Порождаемый объект
F1	Любое	Меню-справочник
F2	Уровень 0 или тело схемы	Простая схема
F3	Уровень 0 или тело схемы	Неогражденный цикл
F4	Уровень 0 или тело схемы	Огражденный цикл
	Начало неогражденного цикла	Заголовок огражденного цикла
	Начало огражденного цикла	Дополнительный заголовок
F5	Уровень 0 или тело схемы	Схема выбора
F6	Вход схемы выбора	Дополнительный вход
F7	текст	? текст
	? текст	[? текст
	[? текст	текст
F8	текст	[* текст
	[* текст	текст
F9	Тело схемы	Слабая стрелка
	Слабая стрелка	Сильная стрелка
	Сильная стрелка	Слабая стрелка
F10	Уровень 0 или тело схемы	Ниша

ГЛАВА 3. СТИЛЬ ПРОГРАММИРОВАНИЯ

Программирование - это искусство, постигаемое в результате опыта работы и знакомства с чужими работами. Способ представления программы во многом зависит от вкуса программиста. Часто приходится выбирать между краткостью и изяществом. В этой главе выделяются те особенности представления и организации программы, которые делают ее визуально более привлекательной и более понимаемой. Рассматриваются ясность, краткость и эффективность - часто противоречивые цели программирования [3].

1. Ясность

На ясность программы влияют два основных фактора: способ визуального представления программы и способ употребления конструкций языка программирования. На первый фактор уже неоднократно обращалось внимание. Наряду с тем, что схематическое программирование само по себе позволяет создавать удобочитаемые программы с помощью использования схем разного вида, полезно использовать средства автоматического "облагораживания" путем введения различных отступов, пустых и пунктирных строк и тому подобных приемов выделения строения программы. Безусловно, полезно широкое применение комментариев.

Вторая составляющая ясности программы - использование возможностей самого языка. Здесь не существует одного "верного" пути. К хорошему решению можно прийти по-разному. При частом использовании языка уменьшается количество нелепых конструкций, появляется умение пользоваться различными "трюками", которые допускает любой язык программирования. Разумеется, речь идет о таких "трюках", которые не уменьшают ясности программы.

2. Краткость

Си - краткий язык, он вдохновляет на сжатое выразительное представление сложных идей. Но мощностью его следует пользоваться осмотрительно. Часто бывает трудно сказать, а не стала ли "краткость" "неясностью". Должен быть очень значительный выигрыш в скорости выполнения, чтобы оправдать включение в любую программу сложных операторов.

3. Эффективность

Улучшение эффективности программ - не всегда легкое или даже желательное дело. Для небольших программ эффект может оказаться незаметным. Но если программа используется интенсивно, то внимание, которое уделяется критическим по времени местам, стремление уменьшить время выполнения программы могут окупиться значительным улучшением производительности.

С точки зрения эффективности всего процесса программирования, наиболее важными следует считать ясность и логическую стройность, читабельность составляемых программ. Остальные характеристики, как правило, являются второстепенными, их улучшения можно добиться путем модификации готовых отлаженных программ.

ГЛАВА 4. ПРОГРАММЫ

1. Целые числа

1.1. Наибольший общий делитель

В программе реализован алгоритм Евклида нахождения наибольшего общего делителя (НОД) двух неотрицательных целых чисел [1]. Алгоритм основан на следующих свойствах этой величины. Пусть m и n - одновременно не равные нулю целые неотрицательные числа и пусть $m \geqslant n$. Тогда, если n = 0, то НОД(m, n) = m, а если $n \neq 0$, то для чисел m, n и r, где r - остаток от деления m на n, выполняется равенство

НОД(m, n) = НОД(n, r).

Отсюда следует идея алгоритма: рассматриваемую пару значений (m, n), где $m \geqslant n$, будем повторно заменять на пару (m', n'), где m' = n, n' - остаток от деления m на n. Такую замену производим до тех пор, пока второй член не станет равным нулю. Ответом будет соответствующее значение m. Ввод данных в программе идет в теле первого неограниченного цикла, условием выхода из которого является ввод допустимых чисел.

Пример решения:

```
D:\SKM>ln1
 m > 0 :  n > 0 :
34        85
 НОД( 34,85 ) = 17
```

```
---      Наибольший общий делитель  чисел m, n                ---
# include <stdio.h>
main()
{
 int r, n, n1, m, m1;
 ===---ввод чисел
  putxt (" m > 0 :  n > 0 : \n ");
  m1 = m = getint(); n1 = n = getint();
 m <= 0 || n <= 0 ?
 ===
 [? (n > m) r = m; m = n; n = r;
 ===---вычисление
 n != 0 ?
  r = m ; m = n;  n = r % n;
 ===
 putxt(" НОД( %d,%d ) = ", m1, n1); putint(m);
}
```

1.2. Разложение на простые множители

Разложение целого числа a на простые множители производится последовательным делением a на 2 и ряд нечетных чисел b (b = 3, 5, 7, ...) по формуле

$$a_i = a_{i-1} / b$$

до тех пор, пока соблюдается условие

$$\sqrt{a_i} + 1 < b .$$

Это условие является условием продолжения выполнения тела последнего неограниченного цикла.

В программе дополнительно к стандартному файлу ввода/вывода данных <stdio.h> включается стандартный файл <math.h>, в котором определена необходимая при решении функция нахождения квадратного корня (функция sqrt()).

```
---     Разложение целого числа на простые множители      ---
-----------------------------------------------------------
# include <stdio.h>
# include <math.h> ---функции abs(), sqrt()
-----------------------------------------------------------
main ()
{
  int a,b;
  do
  {
    putxt(" Целое число a = "); a = getint();
  }
  a == 0 ?
  a = abs(a);
  putxt(" Множители : ");
  do
  {
    a % 2 == 0 ?
    printf(" %d   ", 2);
    a /= 2;
  }
  b = 3;
  do
  {
    b >= (sqrt(a) + 1) ?
  <----
    a % b == 0 ?
    printf(" %d   ", b);
    a /= b;
  <---
    b += 2;
  }
  putint(a);
}
-----------------------------------------------------------.
```

Примеры решений:

```
D:\SKM>ln2
 Целое число a = 30
 Множители :  2   3   5
D:\SKM>ln2
 Целое число a = -68740
 Множители :  2   2   3   3   89
D:\SKM>ln2
 Целое число a = 3080
 Множители :  2   2   2   5   7   11
D:\SKM>
```

1.3. Решето Эратосфена

Легко увидеть, что у кратного числа n найдется делитель $\leqslant \sqrt{n}$. Следовательно, если целое положительное число $n \neq 1$ не делится ни на одно положительное простое число, не большее $\sqrt{n}$, то оно простое.

В последовательности чисел 2,3, ...,$\sqrt{n}$ последовательно вычеркиваем каждое второе число после 2. Первое незачеркнутое число - простое (3). Далее вычеркиваем каждое третье число после 3. Первое незачеркнутое число - простое (5). Затем вычеркиваем каждое пятое число после 5 и так далее до тех пор, пока не дойдем до числа, большего $\sqrt{n}$. Все числа, которые остаются, являются простыми. Такой метод нахождения простых чисел называется решетом Эратосфена [5].

Количество простых чисел r не превышает:

$[1.6n / \ln n + 1]$, $n \leqslant 200$,

$[n / (\ln n - 2) + 1]$, $n > 200$.

В программе используется функция floor(), описанная в стандартном файле <math.h>, значением которой является целая часть от аргумента.

Примеры решений:

```
D:\SKM>ln19
 N (>1) = 30
 Простые числа от 1 до 30 : 1 2 3 5 7 11 13 17 19 23 29
D:\SKM>ln19
 N (>1) = 130
 Простые числа от 1 до 130 : 1 2 3 5 7 11 13 17 19 23 29
 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103
 107 109 113 127
D:\SKM>
```

```
---------------------------------------------------------------
---      Нахождение всех простых чисел от 1 до N            ---
---------------------------------------------------------------
#include <stdio.h>
#include <math.h>
---------------------------------------------------------------
main()
{
 float s;
 int   n, r, p[400], i, j, k;
 putxt(" N (>1) = "); n = getint();

 ----оценка количества простых чисел
 n > 200 ?
  r = floor(n / (log(n) - 2) + 1);
 <----
  r = floor(1.6 * n / log(n) + 1);


 p[1] = 1; p[2] = 2; p[3] = 3;

 [* (i = 4..r) p[i] = 0;

 ---нахождение остальных простых чисел
 j = 3;
 * k = 3..n; 2

  s = sqrt(k);

  * i = 2..n

  p[i] > s ?
   p[j++] = k;
  <----
  --- проверка, делится ли k на p[i] без остатка
  (k % p[i]) == 0 )?
  <----
  <=

 printf(" Простые числа от 1 до %d : ", n);

 * i = 1..r

 [? (p[i] != 0) printf(" %d ", p[i]);
}
---------------------------------------------------------------.
```

2. Массивы

2.1. Сортировка

В программе рассмотрен массив целых чисел a_0, ..., a_{n-1}. Задача сортировки или упорядочения массива по неубыванию, которая решена в этой программе, состоит в необходимости переставить элементы в следующем порядке:

$a_0 \leqslant a_1 \leqslant \ldots \leqslant a_{n-1}$.

В этих целях используется алгоритм сортировки выбором: элемент массива a[j], имеющий наименьшее значение, переставляется на первое место (a_0), затем то же самое проделывается, начиная со второго элемента, то есть рассматривается массив a_1, ..., a_{n-1}, и так далее [1]. Этот метод сортировки отличается краткостью алгоритма, но по скорости работы уступает другим простым методам сортировки [1].

Пример решения:

```
D:\SKM>ln4
 Количество чисел n = 13
 Введите массив a[15] : 0 1 4 3 2 6 -10 -23 4 4 4 5 56
-23  -10  0  1  2  3  4  4  4  4  5  6  56
```

```
---------------------------------------------------------
---          Сортировка массива выбором               ---
---------------------------------------------------------
# include <stdio.h>
---------------------------------------------------------
main()
{
 int a[100];
 int n, r, i, j;
 putxt(" Количество чисел n = "); n = getint();
 putxt(" Введите массив a[%d] :  ", n);
 (* (i = 0..n-1) a[i] = getint();
 {
  * i = 0..n-1

  {
   * j = i..n-1
   {
    (? (a[j] < a[i]) r = a[j]; a[j] = a[i]; a[i] = r;
   }
  printf("%d  ", a[i]);
  }
}
---------------------------------------------------------.
```

2.2. Таблица числа вхождений

Программа выводит на экран таблицу, в которой для каждого значения из введенной последовательности целых чисел a_1, ... , a_n, указывается количество его вхождений.

Элемент a_i последовательно сравнивается со всеми элементами массива, начиная с первого. Сравнение и подсчет числа вхождений производится во внутреннем цикле с управляющей переменной j. Алгоритм усложняется тем, что необходимо пропускать члены, значения которых встречались раньше в последовательности. Эта ситуация проверяется первым условием, при выполнении которого цикл прерывается и начинается обработка следующего элемента.

Пример решения:

```
D:\SKM>ln3
 Количество чисел n = 6
 Введите массив a[6] : 2 2 1 3 5 1
 2   2
 1   2
 3   1
 5   1
```

Таблица входов

```
# include <stdio.h>
main()
{
 int a[100];
 int n, i, j, k;
 putxt(" Количество чисел n = "); n = getint();
 putxt(" Введите массив a[%d] : ", n);
 for (i = 0..n-1) a[i] = getint();
 {
 * i = 0..n-1
 k = 0;
  {
  * j = 0..n-1
  {
  ( a[i] == a[j] ) && ( j < i ) ?
  <-------
   [? ( a[i] == a[j] )  k++;
  }
 putxt("\n %d   %d", a[i], k);
 }
}
```

3. Геометрия

3.1. Площадь треугольника

В программе пользователем вводятся три длины a, b и c. В случае, если они не являются длинами сторон треугольника, выдается сообщение:

Треугольник не существует.

Выполнение программы при этом заканчивается. В противном случае вычисляется площадь треугольника по формуле Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

где p - полупериметр,

$$p = (a + b + c) / 2 .$$

Примеры решений:

```
D:\SKM>ln
 Длины сторон a:   b:   c:
               3    4    9
 Треугольник не существует.
D:\SKM>ln
 Длины сторон a:   b:   c:
               3    4    4
 Площадь треугольника S = 5.562149
D:\SKM>ln
 Длины сторон a:   b:   c:
               4    3    5
 Площадь треугольника S = 6.000000
D:\SKM>
```

```
-----------------------------------------------------------
---                  Площадь треугольника                 ---
-----------------------------------------------------------
# include <stdio.h>
# include <math.h>  ---функции fabs(), sqrt()
-----------------------------------------------------------
main()
{
 float a, b, c, p;
 putxt  (" Длины сторон a:  b:  c:  \n ");
 a = getfloat(); b = getfloat(); c = getfloat();
 --- проверка правильности ввода
a < 0 || b < 0 || c < 0 || a + b < c || fabs(a - b) > c ?
   putxt (" Треугольник не существует.");
<----
 p = (a + b + c) / 2;
 putxt (" Площадь треугольника S = ");
 putfloat( sqrt(p * (p - a) * (p - b) * (p - c)) );
}
-----------------------------------------------------------.
```

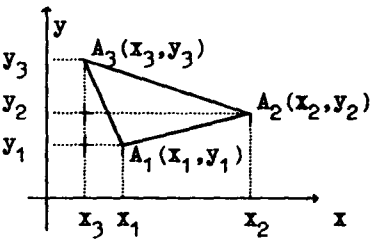
3.2. Площадь n-угольника

Если координаты вершин n-угольника на плоскости есть (x_i, y_i), i=1,...,n , то площадь многоугольника вычисляется по формуле :

$$S = 0.5 \left| \sum_{i=0}^{n-1} (x_n + x_{n+1})(y_i - y_{i+1}) \right| ,$$

где $(x_0, y_0) = (x_n, y_n)$ [5].

Структуру этой формулы можно понять на примере треугольника:



Действительно,

$$S_{A_1A_2A_3} = S_{y_1A_1A_2y_2} + S_{y_2A_2A_3y_3} - S_{y_1A_1A_3y_3};$$

а, например, площадь трапеции $y_1A_1A_2y_2$ равна :

$$S = 0.5 \left|(y_2-y_1)(x_1+x_2)\right|.$$

Программа состоит из двух функций: "main", "asisse". Назначение второй из них - ввод координат. Данная функция не возвращает значения, следовательно, имеет тип "void".

Число вершин и координаты вершин описаны как глобальные переменные, так как к ним обращаются обе функции.

Примеры решений:

```
D:\SKM>ln13
 n = 3
 x1, y1 : 0  0
 x2, y2 : 0  3
 x3, y3 : 4  0
 Площадь 3-угольника =  6.00
D:\SKM>ln13
 n = 2
 x1, y1 : 1  2
 x2, y2 : 5  4
 x3, y3 : 3  5
 x4, y4 : -2  3
 Площадь 4-угольника =  9.50
D:\SKM>
```

```
---                    Площадь  n-угольника                    ---
#include <stdio.h>
#include <math.h>
int n;                          --- число вершин
float x[10], y[10];             --- координаты вершин
main()
{
  int i;
  float s;
  void asisse();

  asisse();                     --- ввод координат
  s = 0;
  * i = 0..(n-1)
    s += (x[i] + x[i+1]) * (y[i] - y[i+1]);
  s = fabs(s) / 2;
  printf(" Площадь %d-угольника = %5.2f ", n, s);
}
void asisse()
{
  int i;
  putxt(" n = "); n = getint();
  * i = 1..n
    printf(" x%d, y%d : ", i, i);
    x[i] = getfloat(); y[i] = getfloat();
  x[0] = x[n]; y[0] = y[n];
}
```

3.3. Положение точки относительно n-угольника

Программа определяет, лежит ли точка с координатами (x_0,y_0) внутри n-угольника, заданного координатами вершин (x_i,y_i), i=1,...,n . Программа получена переводом на язык Си программы на языке Бейсик [5].

Пример решения:

```
D:\SKM>ln14
 Количество вершин n = 4
 Координаты вершин :
 x1, y1 : 0 0
 x2, y2 : 3 0
 x3, y3 : 3 3
 x4, y4 : 0 3
 Точка  (x0,y0): 1.5   1
 Точка   лежит внутри 4-угольника
```

```
---------------------------------------------------------------
---        Положение точки относительно n-угольника         ---
---------------------------------------------------------------
#include <stdio.h>
---------------------------------------------------------------
int n;                          --- число вершин
float x[20], y[20];             --- координаты точки и вершин
---------------------------------------------------------------
main()
{
 int i, b = 1;
 asisse();                      --- ввод координат
 {
 * i = 1..n
 ;
  {
  ((y[0] > y[i]) == (y[0] <= y[i+1])) ?
  (x[0]-x[i]) < (y[0]-y[i])*(x[i+1]-x[i])/(y[i+1]-y[i]) ?
   [? (!b) b = 1; : b = 0;
  }
 }
 [? (!b) b = 1; : b = 0;
 putxt(" Точка ");
 [? (!b) putxt(" не ");
 putxt(" лежит внутри %d-угольника ", n);
}
---------------------------------------------------------------
asisse()
{
 int i;
 putxt(" Количество вершин n = "); n = getint();
 putxt(" Координаты вершин : \n");
 {
 * i = 1..n
 ;
  printf(" x%d, y%d : ", i, i);
  x[i] = getfloat(); y[i] = getfloat();
 }
 printf(" Точка (x0,y0): ");
 x[0] = getfloat(); y[0] = getfloat();
 x[n+1] = x[1]; y[n+1] = y[1];
}
---------------------------------------------------------------
```

4. Многочлены

4.1. Умножение многочленов

В программе умножение многочлена a(x) степени m на многочлен b(x) степени n дает многочлен c(x) степени (m+n). Например, если

$$a(x) = 4x^4 + 3x^3 + 2x^2 + x + 0.5 \ ,$$
$$b(x) = 2x^2 + x + 0.4 \ ,$$

то

$$c(x) = a(x) \times b(x) =$$
$$= 8x^6 + 10x^5 + 8.6x^4 + 5.2x^3 + 2.8x^2 + 0.9x + 0.2 \ .$$

Эта операция выполняется по следующему алгоритму [6]:

1) вводим коэффициенты многочлена a(x);

2) организуем внешний цикл вычислений c_i, задав начальное значение управляющей переменной цикла i=(n+m) и конечное значение m с шагом -1, в начале каждого цикла вводим b_{i-m};

3) организуем внутренний цикл с управляющей переменной j, меняющейся от j=m до 1 с шагом -1, и преобразуем коэффициенты многочлена a(x) по формуле

$$a_{j+m+1} = a_j b_{i-m} + a_{j+m} \ ;$$

4) при выходе из внутреннего цикла вычисляем

$$a_{m+1} = a_0 b_{i-m} \ , \ c_i = a_{2m+1}$$

и завершаем внешний цикл;

5) с помощью цикла с управляющей переменной i, меняющейся от i=m-1 до 0 с шагом -1, вычисляем последние m значений $c_i = a_{i+m+1}$;

6) выводим на печать найденные коэффициенты c[i].

```
---------------------------------------------------------------
---          Умножение многочлена степени  m                ---
---             на многочлен степени  n                     ---
---------------------------------------------------------------
#include <stdio.h>
---------------------------------------------------------------
main()
{
 float a[50], c[50], b;
 int m, n, i, j;
 puts("Введите степени многочленов  m, n:");
 m = getint(); n = getint();
 for (i = (2 * m)..(m - 1); -1) a[i] = 0;
 ---  ввод коэффициентов
 for (i = m..0; -1) printf(" a%d = ",i); a[i] = getfloat();
 {--- вычисление c[]
 * i = (n + m)..m; -1
 {
  printf(" b%d = ", i - m); b = getfloat();
  {---- преобразование коэффициентов a[]
  * j = m..1; -1
  {
   a[j+m+1] = a[j] * b + a[j+m];
  }
  a[m+1] = a[0] * b;
  c[i] = a[2*m+1];
 }
 {--- последние m значений c[]
 * i = (m - 1)..0; -1
 {
  c[i] = a[i+m+1];
 }
 {--- вывод значений c[]
 * i = (n + m)..0; -1
 {
  printf("\n c%d = %5.2f ", i, c[i]);
 }
}
---------------------------------------------------------------
```

Пример решения:

```
D:\SKM>ln9
Введите степени многочленов  m, n:  4  2
 a4 = 4
 a3 = 3
 a2 = 2
 a1 = 1
 a0 = 0.5
 b2 = 2
 b1 = 1
 b0 = 0.4

 c6 =  8.00
 c5 = 10.00
 c4 =  8.60
 c3 =  5.20
 c2 =  2.80
 c1 =  0.90
 c0 =  0.20
```

4.2. Схема Горнера

В программе схема Горнера применяется для вычисления степенного многочлена

$$a(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

в виде:

$$a(x) = (\ldots(a_n x + a_{n-1})x + a_{n-2})x + \ldots + a_1)x + a_0 .$$

Пример решения:

```
D:\SKM>ln22
Введите степень многочлена n: 4
 a4 = 4
 a3 = 3
 a2 = 2
 a1 = 1
 a0 = 0
Введите значение x: 2
Значение многочлена : 98.00
D:\SKM>
```

```
---   Схема Горнера для вычисления степенных многочленов   ---
#include <stdio.h>
main()
{
 float a[50], x, y = 0;
 int   i, n;

 putxt("Введите степень многочлена n:"); n = getint();
 ---ввод коэффициентов
 [* (i = n..0; -1) printf(" a%d = ", i); a[i] = getfloat();
 putxt("Введите значение x:"); x = getfloat();
 ---вычисление значения многочлена
 [* (i = n..1; -1) y = (y + a[i]) * x;
 y += a[0];
 printf("Значение многочлена : %5.2f", y);
}
```

5. Матрицы

5.1. Умножение матриц

Умножение матриц *A* с размерностью (m x n) и *B* с размерностью (n x 1) в программе выполняется по формуле:

$$c_{kj} = \sum_{i=1}^{n} (a_{ki} b_{ij}) ,$$

где j=1,...,l и k=1,...,m . Получаемая матрица *C* имеет размерность (m x l).

В программе дополнительно описаны две функции - "asisse" и "avalja" (ввода матриц и вывода результата), причем первая функция оформлена нишей.

Пример решения:

```
D:\SKM>ln10
 Размерности m, n, l : 3  2  2
 a11 = 2
 a12 = 5
 a21 = 4
 a22 = 1
 a31 = 2
 a32 = 3
**********
 b11 = 6
 b12 = 8
 b21 = 3
 b22 = 1
**********
C(3,2)  :
c11 = 27.00   c12 = 21.00
c21 = 27.00   c22 = 33.00
c31 = 21.00   c32 = 19.00
```

```
-------------------------------------------------------------ln10---
---        Умножение матриц  (m x n) и (n x 1)                  ---
---           Результат : матрица  (m x 1)                      ---
-------------------------------------------------------------------
#include <stdio.h>
-------------------------------------------------------------------
float a[10][10], b[10][10], c[10][10];
int   m, n, l;
-------------------------------------------------------------------
main()
{
  int   i, j, k;
  float s;
  asisse();              --- ввод
  ---- умножение
  * k = 1..m
  * j = 1..l
  {
    s = 0;
    {
      * i = 1..n
      {
        s += a[k][i] * b[i][j];
      }
    }
    c[k][j] = s;
  }
  avalja();              --- вывод
}
-------------------------------------------------------------------
avalja()  --- вывод
{
  int i, j;
  printf(" C(%d,%d) : \n", m, l);
  {
    * i = 1..m
    {
      {
        * j = 1..l
        {
          printf(" c%d%d = %5.2f ", i, j, c[i][j]);
        }
      }
      printf("\n");
    }
  }
}
ln10_1
--- asisse()  ввод
-------------------------------------------------------------------
```

```
----------------------------------------------------ln10_1---
---                    Ввод матриц                          ---
-------------------------------------------------------------
asisse()
{
  int i, j;
  putxt(" Размерности m, n, l : ");
  m = getint(); n = getint(); l = getint();
  {
  * i = 1..m
  * j = 1..n
  }
    printf(" a%d%d = ", i, j); a[i][j] = getfloat();
  printf("**********\n");
  {
  * i = 1..n
  * j = 1..l
  }
    printf(" b%d%d = ", i, j); b[i][j] = getfloat();
  printf("**********\n");
}
-------------------------------------------------------------.
```

5.2. Вычисление определителя

В программе вычисляется определитель матрицы a с размерностью (n x n). Используется метод триангуляции. Триангуляция производится с выбором максимального элемента [5].

Алгоритм вычисления:

1) в первом столбце определяется максимальный по абсолютной величине элемент m1, если он находится в j-ой строке, то первая и j-ая строки переставляются, при m1=0 определитель равен нулю;

2) из элементов a_{ij} ($j \geqslant 2$) вычитаются соответствующие элементы первого столбца, умноженные на a_{1j} / a_{11}, в результате получаем

$$\det a = \Delta_n = a_{11}\Delta_{n-1} ,$$

где элементы новой матрицы a' вычисляются по формуле

$$a'_{ij} = a_{ij} - a_{i1}a_{1j} / a_{11} ;$$

3) к определителю полученной матрицы a'_{ij} применяем тот же прием еще (n-2) раза и окончательно получаем, что

$$\det a = \Delta_n = a_{11}a'_{22}\ldots a'_{nn} .$$

```
-------------------------------------------------------------ln16---
---              Вычисление   определителя                    ---
---                методом  триангуляции                      ---
-------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
-------------------------------------------------------------------
int n, k;     float a[20][20], m1;
-------------------------------------------------------------------
main()
{
 int   i, j, l;
 float d = 1, t;
 asisse();   --- ввод
 {
 * k = 1..n
 {
  m1 = 0;
  {---выбор максимального по модулю элемента в столбце
  * i = k..n
  {
   t = a[i][k];
   [? (fabs(t) > fabs(m1)) m1 = t; j = i;
   ==
  }
  {
  m1 == 0 ?
   d = 0;
 <-----
   --- перестановка  j-  и  k-й  строк
  j != k ?
   d = -d;
   {
   * l = k..n
   {
    t = a[j][l]; a[j][l] = a[k][l]; a[k][l] = t;
    ==
   }
  matrix();  ---  вычисление  элементов  матрицы
  d *= a[k][k];
  ==
 }
 printf(" Определитель  det : %5.2f ", d);
}
-ln16_1
--- asisse()
--- matrix()
-------------------------------------------------------------------
```

```
-------------------------------------------------------------lm16_1---
---          Ввод и вычисление элементов матрицы                  ---
-----------------------------------------------------------------------
asisse()
{
 int i, j;
 putxt(" Размерность матрицы n = "); n = getint();
 {
  * i = 1..n
  * j = 1..n
  {
   printf(" a%d%d = ", i, j); a[i][j] = getfloat();
  }
 }
}
-----------------------------------------------------------------------
matrix()
{
 int   i, j;
 float t;
 {
  * i = (k+1)..n
  {
   t = a[i][k] / m1;
   {
    * j = (k+1)..n
    {
     a[i][j]  -= t * a[k][j];
    }
   }
  }
 }
}
-----------------------------------------------------------------------
```

Пример решения:

```
D:\SKM>lm16
 Размерность матрицы n = 3
 a11 = 1
 a12 = 2
 a13 = 3
 a21 = 4
 a22 = 5
 a23 = 6
 a31 = 7
 a32 = 8
 a33 = -9
Определитель  det : 54.00
D:\SKM>
```

6. Численные методы

6.1. Метод трапеций

Программа вычисляет значение определенного интеграла

$$I = \int_A^B \frac{1}{\sqrt{3x^2 - 1}}\, dx$$

на заданном отрезке [A, B] с точностью 0.00001 по квадратичной формуле трапеций. При каждом шаге итерации отрезок разбивается на n частей (n = 3, 5, 9, 17, 33, ...) длиной h = = (B - A) / n , к каждой из которых применяется формула трапеции:

$I \approx h(y_0/2 + y_1 + y_2 + \ldots + y_{n-1} + y_n/2)$,

где y_i - значения интеграла на определенном отрезке разбиения, вычисляемые в описанной отдельно функции "trap", имеющей тип "double".

Каждый последующий шаг итерации происходит внутри неограниченного цикла, выполнение которого прекращается при условии, если разность между двумя найденными значениями интеграла не больше заданной точности 0.00001.

Примеры решений:

```
D:\SKM>ln5

 Отрезок :
 A = 2
 B = 100
 Значение интеграла : 2.2710
D:\SKM>ln5

 Отрезок :
 A = 3
 B = 100
 Значение интеграла : 2.0299
```

```
---------------------------------------------------------------
---        Вычисление определенного интеграла               ---
---        по квадратичной формуле трапеций                 ---
---------------------------------------------------------------
# include <stdio.h>
# include <math.h>
---------------------------------------------------------------
double A, B, h;
int   n;
---------------------------------------------------------------
main()
{
  double I1, I2, or, c;
  double trap();
  c = sqrt(1./3.);
  {
    putxt("\n Отрезок : \n");
    putxt(" A = "); A = getfloat();
    putxt(" B = "); B = getfloat();
    ---  проверка точки, в которой интеграл не определен
    (c >= A) && (c <= B)  ?
  }
  n = 2;
  h = B - A;
  I1 = trap();
  {---  итерация
    I2 = I1;          --- сохранение предыдущего значения
    n = n * 2 - 1;    --- количество частей
    h /= 2;           --- шаг
    I1 = trap();      --- приближенное значение интеграла
    fabs((I1 - I2) / 3) >= 0.00001 ?
  }
  or = (I1 - I2) / 3; ---  оценка  Рунге
  I2 += or;
  printf(" Значение интеграла : %6.4f", I2);
}
---------------------------------------------------------------
double trap() ---  вычисление  интеграла
{
  double I, cf;
  int k;
  I = 0;
  {
    * k = 1..n
    {
      cf = 1;
      [? (k == 1 || k == n) cf = 0.5;
      I += (1 / sqrt(3 * pow(A + (k - 1) * h, 2) - 1)) * cf;
    }
<---(I * h);
}
---------------------------------------------------------------
```

6.2. Метод дихотомии

Программа вычисляет корень уравнения f(x) на заданном пользователем отрезке [A, B] и с заданной погрешностью e. Используется метод дихотомии или метод половинного деления, по которому данный отрезок делится пополам и выбирается тот полуинтервал, на концах которого знаки f(x) разные. Затем процесс деления повторяется до тех пор, пока длина интервала не станет меньше e.

Ввод отрезка поиска корня идет до тех пор, пока значения функции f(x) на концах отрезка не будут иметь разные знаки.

Описанной в программе функции "f" передается в качестве параметра значение x (середина отрезка), функция,в свою очередь, возвращает значение f(x) в этой точке. В данном случае $f(x) = x^2 + x - 2$.

Примеры решений:

```
D:\SKM>ln7
 Введите отрезок  [A, B]: -10  10
 Погрешность вычисления e = 0.001
 На отрезке [-10.000000; 10.000000] корней нет,
 либо их несколько.
 Введите отрезок  [A, B]: -10  0
 Погрешность вычисления e = 0.001

 x = -2.000122, f(x) = 0.000366
D:\SKM.
```

```
---------------------------------------------------------------
---      Вычисление корней квадратного уравнения            ---
---                  методом дихотомии                      ---
---------------------------------------------------------------
# include <stdio.h>
# include <math.h>
---------------------------------------------------------------
main()
{
  double a, b, x, fx, fa, fb, e;
  double f();
  {
    putxt(" Введите отрезок  [A, B]:");
    scanf("%lf%lf", &a, &b);
    [? (b < a) x = b; b = a; a = x;
    putxt(" Погрешность вычисления e = ");
    scanf("%lf", &e);
    fa = f(a);
    fb = f(b);
    {
      !(((fa > 0) && (fb > 0)) || ((fa < 0) && (fb < 0))) ?
<-----
    }
    putxt(" На отрезке [%f; %f] корней нет,\n", a, b);
    putxt(" либо их несколько.              \n");
  }
  {
  ((b - a) > e) ?
    x = (a + b) / 2;
    fx = f(x);
    {
      (fa * fx) > 0 ?
       a = x; fa = fx;
<-----
      b = x; fb = fx;
    }
  }
  fx = f(x);
  printf("\n x = %f, f(x) = %f", x, fx);
}
---------------------------------------------------------------
double f(x)
double x;
{
<-----(pow(x,2) + x - 2)
}
---------------------------------------------------------------
```

6.3. Метод Гаусса

В программе рассматривается система из n линейных уравнений:

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1,$$
$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2,$$
$$\ldots$$
$$a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n .$$

Наиболее употребительным алгоритмом решения систем линейных уравнений является алгоритм Гаусса или алгоритм исключения неизвестных [2]. В общем случае алгоритм предписывает приведение системы n-го порядка к равносильной ей треугольной системе с единичными коэффициентами на диагонали:

$$x_1 + a'_{12}x_2 + a'_{13}x_3 + \ldots + a'_{1n}x_n = b'_1,$$
$$x_2 + a'_{23}x_3 + \ldots + a'_{2n}x_n = b'_2,$$
$$\ldots$$
$$x_n = b'_n .$$

Этот этап называется прямым ходом алгоритма Гаусса, а затем (обратный ход) - вычисление значения неизвестных, начиная с x_n и кончая x_1 .

Пример решения:

```
D:\SKM>ln8
Введите количество уравнений  n:  3
 a11 = 4
 a12 = 0.24
 a13 = -0.08
 b1 = 8
 a21 = 0.09
 a22 = 3
 a23 = -0.15
 b2 = 9
 a31 = 0.04
 a32 = -0.08
 a33 = 4
 b3 = 20

 x1 =  1.91
 x2 =  3.19
 x3 =  5.04
D:\SKM>
```

```
                                                                  ln8
---          Решение системы линейных уравнений                   ---
---                    методом  Гаусса                            ---
-------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
-------------------------------------------------------------------
double a[50][50], x[50]; int n;
-------------------------------------------------------------------
main()
{
 double mabs, v;
 int i, j, k, l;
 putxt("Введите количество уравнений  n:"); n = getint();
 asisse();    --- ввод
 ------ прямой ход исключения переменных
 * i = 1..n
 {
  --- выбор уравнения с максимальным a[i][i]
  mabs = fabs(a[i][i]); k = i;
  {
   * l = (i+1)..n
   {
   [? (fabs(a[l][i]) > mabs) mabs=fabs(a[l][i]); k=l;
   }
  }
  ------ перестановка i- и k-го уравнений
  k != i ?
   [* (j=1..(n+1)) v=a[i][j]; a[i][j]=a[k][j]; a[k][j]=v;
  --- деление i-го уравнения на коэффициент при x[i]
  v = a[i][i];
  [* (j=1..(n+1)) a[i][j] /= v;
  --- исключение x[i] из l-го уравнения с помощью i-го
  {
   * l = (i+1)..n
   {
    v = a[l][i];
    [* (j = (i+1)..(n+1)) a[l][j] -= a[i][j] * v;
   }
  }
 }
 x[n] = a[n][n+1];
 ------ обратный ход (нахождение x[])
 * i = (n - 1)..1; -1
 {
  x[i] = a[i][n+1];
  [* (j=(i+1)..n) x[i] -= a[i][j] * x[j];
 }
 avalja();    --- вывод
}
ln8_1
--- ввод коэффициентов и свободных членов
--- вывод решения
-------------------------------------------------------------------
```

```
-------------------------------------------------------lm8_1--------
--- ввод коэффициентов и свободных членов, вывод решения ---
------------------------------------------------------------------
asisse()            --- ввод
┌
 int i, j;
 ┌
 * i = 1..n
 ├
 ┌
 * j = 1..n
 ├
  printf(" a%d%d = ",i ,j); scanf("%lf",&a[i][j]);
 ═
 printf(" b%d = ",i); scanf("%lf",&a[i][n+1]);
 ═
└
------------------------------------------------------------------
avalja()            --- вывод
┌
 int i;
 [* (i = 1..n) printf("\n x%d = %5.2f", i, x[i]);
└
------------------------------------------------------------------
```

6.4. Метод простых итераций

В программе решение системы линейных уравнений находится методом простых итераций. При начальных приближениях $x_i(i=1,2,\ldots,n)$ вычисляются последовательные приближения по формуле простых итераций [6]:

$$x_{i(j+1)} = x_{i(j)} - \frac{1}{a_{ii}}\left(\sum_{k=1}^{n} a_{ik}x_{k(j)} - b_i \right)$$

до тех пор, пока

$$x_{i(j+1)} - x_{i(j)} \geqslant e ,$$

где (j) - номер итерации, e - заданная погрешность вычислений. Итерационный процесс сходится, если величина модуля каждого диагонального элемента матрицы коэффициентов системы больше суммы модулей остальных элементов.

Пример решения:

```
D:\SKM>ln11
 Количество уравнений   n =  3
 Погрешность вычислений  e =  0.0001
 a11 = 4
 a12 = 0.24
 a13 = -0.08
 b1 = 8
 a21 = 0.09
 a22 = 3
 a23 = -0.15
 b2 = 9
 a31 = 0.04
 a32 = -0.08
 a33 = 4
 b3 = 20
 Начальные приближения :
 x1 = 1
 x2 = 1
 x3 = 1
 **********************************
 x1 =  1.91  x2 =  3.19  x3 =  5.04
 Количество итераций = 5
```

```
-------------------------------------------------------------ln11---
---            Решение системы n уравнений                     ---
---              методом простых итераций                      ---
-------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
-------------------------------------------------------------------
float a[10][10], b[10], x[10], z[10], e; int n, s;
-------------------------------------------------------------------
main()
{
 int k, i, j;
 s = 0;  ---  количество итераций
 asisse();   --- ввод
 do  --- шаг итерации
 {
  k = 0;
  for
  (* i = 1..n
  )
  {
   x[i] = - b[i];
   for (j = 1..n) x[i] += a[i][j] * z[j];
   if (fabs(x[i] / a[i][i]) >= e) k = 1;
   x[i] = z[i] - x[i] / a[i][i];
  }
  for (i = 1..n) z[i] = x[i];
  s++;
 }
 while (k == 1) ;
 avalja();   --- вывод
}
-ln11_1
---asisse()
---avalja()
-------------------------------------------------------------------
```

```
-------------------------------------------------------lnl1_1---
---          Ввод системы  и  вывод решения                 ---
-------------------------------------------------------------------
asisse()          --- ввод
{
 int i, j;
 putxt(" Количество  уравнений   n =  "); n = getint();
 putxt(" Погрешность вычисления  e =  "); e = getfloat();
 ---- ввод коэффициентов и свободных членов
 * i = 1..n
 {
  * j = 1..n
  {
   printf(" a%d%d = ", i, j); a[i][j] = getfloat();
  }
  printf(" b%d = ", i); b[i] = getfloat();
 }
 putxt(" Начальные приближения : \n");
 * i = 1..n
 {
  printf(" x%d = ", i); z[i] = getfloat();
 }
}
-------------------------------------------------------------------
avalja()          --- вывод
{
 int i;
 putxt(" ********************************* \n");
 * i = 1..n
 {
  printf(" x%d = %5.2f ", i, x[i]);
 }
 printf("\n Количество итераций = %d ", s);
}
-------------------------------------------------------------------
```

6.5. Максимум функции

Программа ищет максимум функции f(x) на заданном отрезке [a, b] с заданной погрешностью вычислений e, используя метод дихотомии или деления интервала поиска пополам. Указанный метод реализуется следующим алгоритмом [6]:

1) проверяем условие $|b - a| < 2e$, если оно выполняется, идем к пункту 5, если нет, - к пункту 2;

2) делим интервал поиска пополам и вычисляем две абциссы, симметрично расположенные относительно точки $x = (a + b) / 2$:

$x_1 = (a + b - e) / 2$ и $x_2 = (a + b + e) / 2$;

3) для этих значений вычисляем $f(x_1)$ и $f(x_2)$;

4) проверяем, если $f(x_1) > f(x_2)$, то полагаем $b = x_2$, идем к пункту 1, иначе берем $a = x_1$ и идем к пункту 1;

5) выводим на печать $x_1 = (a + b) / 2$ и значение функции в точке экстремума $f(x_1)$.

Примеры решений при $f(x) = 0.1x^3 - 2x^2 + 10x$:

```
D:\SKM>ln20
 Отрезок   поиска   [A, B] : -100 10
 Погрешность вычисления  E : 0.00001
 Максимум в точке  X = 3.33
 Значение функции  F( 3.33) = 14.81
D:\SKM>ln20
 Отрезок   поиска   [A, B] : 10 100
 Погрешность вычисления  E : 0.001
 Максимум в точке  X = 100.00
 Значение функции  F(100.00) = 80997.80
D:\SKM>
```

```
---------------------------------------------------------
---            Поиск  максимума  функции               ---
---              методом  дихотомии                     ---
---------------------------------------------------------
#include <stdio.h>
#include <math.h>
---------------------------------------------------------
main()
{
  float a, b, e;
  double x1, x2, f1, f2;
  double func();
  putxt(" Отрезок   поиска   [A, B] : ");
  a = getfloat(); b = getfloat();
  putxt(" Погрешность вычисления  E : ");
  e = getfloat();
  {
   fabs(b - a) >= (2 * e) ?
    x1 = (a + b - e) / 2; x2 = (a + b + e) / 2;
    f1 = func(x1);        f2 = func(x2);
    [? (f1 > f2)  b = x2; : a = x1;
  }
  x1 = (a + b) / 2;
  printf(" Максимум в точке  X = %5.2f \n", x1);
  printf(" Значение функции  F(%5.2f) = %5.2f ", x1, func(x1));
}
---------------------------------------------------------
double func(x)
double x;
{
<------(0.1 * pow(x, 3) - 2 * pow(x, 2) + 10 * x)
}
---------------------------------------------------------
```

7. Комбинаторика

В программе вычисляется число перестановок, размещений и сочетаний. Факториал при целом n≥ 1 вычисляется по формуле:

$n! = n(n-1)(n-2)\ldots 2\cdot 1$;

число перестановок:

$P(n, n) = n!$;

число размещений из n элементов по m:

$P(n, m) = n! / (n-m)!$;

число сочетаний из n элементов по m:

$C(n, m) = n! / ((n-m)!\, m!)$.

Причем, при вводе чисел n и m должны удовлетворяться неравенства:

$n \geqslant 1$ и $1 \leqslant m \leqslant n$.

Примеры решений:

```
D:\SKM>ln6
 n = 10
 m = 5
Число перестановок P(n,n) = 3628800
Число размещений   P(n,m) = 30240
Число сочетаний    C(n,m) = 252
D:\SKM>ln6
 n = 9
 m = 6
Число перестановок P(n,n) = 362880
Число размещений   P(n,m) = 60480
Число сочетаний    C(n,m) = 84
D:\SKM>
```

```
--------------------------------------------------------------
---       Вычисление  числа перестановок, размещений       ---
---                      и  сочетаний                       ---
--------------------------------------------------------------
# include <stdio.h>
--------------------------------------------------------------
unsigned long nf, mf, kf;
int n, m, k;
--------------------------------------------------------------
main()
┌
 unsigned long fakt();
 asisse();
 nf = fakt(n);
 kf = fakt(k);
 mf = fakt(m);
 avalja();
└
--------------------------------------------------------------
asisse()                              ---  ввод чисел
┌
 ┌─
  putxt(" n = "); n = getint();
  putxt(" m = "); m = getint();
 (n < 1) || (m < 1) || (m > n) ?
 └─
 k = n - m;
└
--------------------------------------------------------------
unsigned long fakt(x)                 ---  вычисление факториала
int x;
┌
 long s;
 int t;
 s = 1;
 [* (t = 2..x) s *= t;
<———(s)
└
--------------------------------------------------------------
avalja()                              ---  вывод результатов
┌
 printf("Число перестановок P(n,n) = %lu\n", nf);
 printf("Число размещений   P(n,m) = %lu\n", nf / kf);
 printf("Число сочетаний    C(n,m) = %lu", nf / (kf*mf));
└
--------------------------------------------------------------.
```

8. Комплексные числа

Арифметические действия над комплексными числами, заданными в алгебраической форме, в программе производятся по формулам:

пусть $z_1 = a + ib$, $z_2 = c + id$,

тогда $z_1 + z_2 = (a + c) + i(b + d)$,

$z_1 - z_2 = (a - c) + i(b - d)$,

$z_1 * z_2 = (ac - bd) + i(ad + cb)$,

$z_1 / z_2 = ((ac + bd) + i(bc - ad)) / (c^2 + d^2)$.

Решение реализовано через схему выбора.

Примеры решений:

```
D:\SKM>ln12
 Re Z1, ImZ1 : 49   100
 Re Z2, ImZ2 : 23   -2.5
 +, -, *, / ?  /
 Re Z0 =  1.64, Im Z0 =  4.53
D:\SKM>ln12
 Re Z1, ImZ1 : 3.45  -2.56
 Re Z2, ImZ2 : -1.15  0.06
 +, -, *, / ?  +
 Re Z0 =  2.30, Im Z0 = -2.50
D:\SKM>ln12
 Re Z1, ImZ1 : 0.0023   0.08
 Re Z2, ImZ2 : 78.9     -0.12
 +, -, *, / ?  *
 Re Z0 =  0.19, Im Z0 =  6.31
D:\SKM>
```

```
--- Арифметические действия с комплексными числами в ---
---              алгебраической форме                ---
```

```
#include <stdio.h>
```

```
main()
{
 long float a, b, c, d, l, e, f;
 int  k;
 putxt(" Re Z1, ImZ1 : ");
 a = getfloat(); b = getfloat();
 putxt(" Re Z2, ImZ2 : ");
 c = getfloat(); d = getfloat();
 putxt(" +, -, *, / ? "); getchar();
 k = getchar();
 {
  ? k
  ='+':---сложение
   e = a + c; f = b + d;
  <---
  ='-':---вычитание
   e = a - c; f = b - d;
  <---
  ='*':---умножение
   e = a * c - b * d; f = a * d + c * b;
  <---
  ='/':---деление
   l = c * c + d * d;
   e = (a * c + b * d) / l; f = (b * c - a * d) / l;
 }
 printf(" Re Z0 = %5.2f, Im Z0 = %5.2f", e, f);
}
```

9. Календарь

9.1. День недели

Программа определяет день недели по известной дате. Получена программа переводом на язык Си Бейсик-программы [5].

Примеры решений:

```
D:\SKM>ln17
 Число, месяц, год : 31  1  1987
 31.1.1987 -  суббота.
D:\SKM>ln17
 Число, месяц, год : 15  6  1990
 15.6.1990 -  пятница.
D:\SKM>
```

```
---      Определение дня недели по известной дате          ---
```

```
#include <stdio.h>
#include <math.h>
```

```
main()
{
 long int n;
 int  d, m, y, m1, y1;
 putxt(" Число, месяц, год : ");
 d = getint(); m1 = m = getint(); y1 = y = getint();
 [? (m > 2) m++; : m += 13; y--;
 n = floor(365.25 * y) + floor(30.6 * m) + d - 621050;
 n = n - floor(n / 7) * 7 + 1;
 printf(" %d.%d.%d - ", d, m1, y1);
 {
 ? n
 =1:
  putxt(" понедельник.");
 <---
 =2:
  putxt(" вторник.");
 <---
 =3:
  putxt(" среда.");
 <---
 =4:
  putxt(" четверг.");
 <---
 =5:
  putxt(" пятница.");
 <---
 =6:
  putxt(" суббота.");
 <---
 =7:
  putxt(" воскресенье.");
 }
}
```

9.2. Количество дней

Программа определяет количество дней между двумя датами, вычисляя вначале количество дней от начала летоисчисления [5].

Примеры решений:

```
D:\SKM>ln18
 День, месяц, год : 3 6 1987
 День, месяц, год : 19 8 1965
 количество дней равно 7965.
D:\SKM>ln18
 День, месяц, год : 1 1 1984
 День, месяц, год : 31 12 1984
 количество дней равно 365.
D:\SKM>
```

```
---      Количество дней между двумя датами                    ---
#include <stdio.h>
#include <math.h>
int d, m, y;
main()
{
 long int kld(), n, n1;
 dsisse();   --- ввод
 n1 = kld();
 dsisse();
 n  = kld();
 n  = fabs(n1 - n);
 printf(" Количество дней равно %d. ", n);
}
dsisse()          --- ввод даты
{
 putxt(" День, месяц, год : ");
 scanf(" %d%d%d", &d, &m, &y);
}
long int kld()    --- подсчет количества дней
{
 long int n;
 [? (m > 2) m++; : m += 13; y--;
 n = floor(365.25 * y) + floor(30.6 * m) + d;
<----(n);
}
```

10. Обработка последовательности символов

Программа переводит введенное римское число в арабское. Число вводится строкой. Функцией "touppet" символы преобразуются в верхний регистр. Для обработки символов применяется схема выбора.

Примеры решений:

```
D:\SKM>roman
 Введите римское число : IV
 Его арабская запись   : 4
D:\SKM>roman
 Введите римское число : VIII
 Его арабская запись   : 8
D:\SKM>roman
 Введите римское число : XXXI
 Его арабская запись   : 31
D:\SKM>roman
 Введите римское число : XLVI
 Его арабская запись   : 46
D:\SKM>roman
 Введите римское число : XCII
 Его арабская запись   : 92
D:\SKM>roman
 Введите римское число : CDXLI
 Его арабская запись   : 441
D:\SKM>roman
 Введите римское число : ID
 Его арабская запись   : 499
D:\SKM>roman
 Введите римское число : DCXCV
 Его арабская запись   : 695
D:\SKM>roman
 Введите римское число : MCMLXXXIV
 Его арабская запись   : 1984
```

```
--- Перевод чисел из римской ---
--- формы записи в арабскую ---
#include <stdio.h>
main()
{
 char inp_str[50];
 int i, len, n, itog = 0, pred = 0;
 printf(" Введите римское число : ");
 scanf("%50s", &inp_str);
 len = strlen(inp_str); --- определение длины строки
 *  i = (len - 1),.0; -1
 {
  n = 0;
  --- преобразование буквы в верхний регистр
  ? toupper(inp_str[i])
  ='I' :   --- единица
   n = 1;
  <
  ='V' :   --- пять
   n = 5;
  <
  ='X' :   --- десять
   n = 10;
  <
  ='L' :   --- пятьдесят
   n = 50;
  <
  ='C' :   --- сто
   n = 100;
  <
  ='D' :   --- пятьсот
   n = 500;
  <
  ='M' :   --- тысяча
   n = 1000;
  <
  =    :
  <-----
  [? (n < pred) n *= -1; : pred = n;
  itog += n;
 }
 printf(" Его  арабская запись  : %d", itog);
}
```

11. Сравнение файлов

Программа сравнивает два текстовых файла. Для вызова функции необходимо ввести строку:

compfile <файл1> <файл2> ,

где

compfile - имя данной программы,

<файл1>, <файл2> - имена сравниваемых файлов, передаваемые программе в качестве аргументов.

Программа проверяет правильность вызова функцией "InputControl". Далее происходит открытие файлов для сравнения, если оно возможно. Сравнение файлов проводится построчно функцией "CompareFiles". Выдаются следующие результаты: количество проверенных строк (count), в каких строках различия и какой файл меньше, или сообщение о том, что файлы эквивалентны. В конце программы файлы закрываются.

В этом примере демонстрируются некоторые дополнительные возможности языка Си:

1) программа с аргументами: в скобках за именем main указываются число аргументов (argc) и массив ссылок на имена аргументов (*argv[]). Значения этих параметров передаются операционной системой при запуске программы;

2) прототип функции: вместе с типом функции описываются и типы формальных параметров, например:

```
int CompareFiles(char *argv[]);
```

3) типы формальных параметров можно указывать непосредственно за именем функции в заголовке функции:

```
void InputControl(int numarg)
{...
}
```

вместо

```
void InputControl(numarg)
int numarg;
{...
}
```

Примеры решений:

```
D:\SKM>compfile ln8.skm ln8.skm
 Файлы эквивалентны.
  Проверено 55 строк

D:\SKM>compfile c:autoexec.bat a:autoexec.bat
 Различие в строке 1.
 Различие в строке 2.
 Различие в строке 3.
 Различие в строке 4.
 Различие в строке 5.
 Различие в строке 6.
 Различие в строке 7.
 Различие в строке 8.
 Различие в строке 9.
Файл c:autoexec.bat меньше, чем a:autoexec.bat.
  Проверено 9 строк
```

```
-------------------------------------------------------------
---            Сравнение  текстовых  файлов                 ---
-------------------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
-------------------------------------------------------------
FILE *f[2];
------------------ прототипы  функций ----------------------
void InputControl(int numarg);
void OpenFiles(char *argv[]);
int CompareFiles(char *argv[]);
void CloseFiles(char *argv[]);
-------------------------------------------------------------
main(int argc, char *argv[])
{
  int cmp;
  InputControl(argc);       --- проверка ввода
  OpenFiles(argv);          --- открытие файлов
  cmp = CompareFiles(argv);--- сравнение файлов
  printf("  Проверено %d строк  \n", cmp);
  CloseFiles(argv);         --- закрытие файлов
  exit(0);                  --- нормальное завершение работы
}
-------------------------------------------------------------
compare
 --- функция CompareFiles

ic_of_cf
 --- функции InputControl, OpenFiles и CloseFiles

-------------------------------------------------------------
```

--- Сравнение файлов построчно ---

```
int CompareFiles(char *argv[])
{
  int count = 0, diff = 0;
  char s0[80];
  char s1[80];
  {
  ((!feof(f[0])) && (!feof(f[1])))  ?
   fgets(s0, 80, f[0]);
   fgets(s1, 80, f[1]);
   count++;              --- количество проверенных строк
   {
   (strcmp(s0, s1) != 0) ?
    printf(" Различие в строке %d.\n", count);
    diff = 1;         --- есть различие
   }
  }
  {
  (feof(f[0]) && feof(f[1]))  ?
   if (!diff) printf(" Файлы эквивалентны.\n");
<-------(count)
  }
  {
  (feof(f[0]) && !feof(f[1])) ?
   printf("Файл %s меньше, чем %s.\n", argv[1], argv[2]);
<-------(count)
  }
  {
  (!feof(f[0]) && feof(f[1])) ?
   printf("Файл %s меньше, чем %s.\n", argv[2], argv[1]);
<-------(count)
  }
<-----(count)
}
```

```
-------------------------------------------------------------
---   Функции проверки ввода и открытия/закрытия файлов  ---
-------------------------------------------------------------
void InputControl(int numarg)
{
  (numarg != 3) ?
   printf("  Правильный вызов программы : \n");
   printf("  compfile  первый_файл  второй_файл  n");
   exit(1);              --- ненормальное завершение работы .
                         --- (сравнение не произведено)
}
-------------------------------------------------------------
void OpenFiles(char *argv[])
{
  int i;
  * i = 1..2
  {
    (NULL == (f[i-1] = fopen(argv[i], "r")))  ?
     printf(" Невозможно открыть файл  %s\n", argv[i]);
     exit(1);          --- ненормальное завершение работы
                       --- (сравнение не произведено)
  }
-------------------------------------------------------------
void CloseFiles(char *argv[])
{
  int i;
  * i = 0..1
  {
    (0 != fclose(f[i])) ?
     printf("  Невозможно закрыть файл %s\n", argv[i]);
     exit(1);          --- ненормальное завершение работы
  }
}
-------------------------------------------------------------.
```

12. Графика

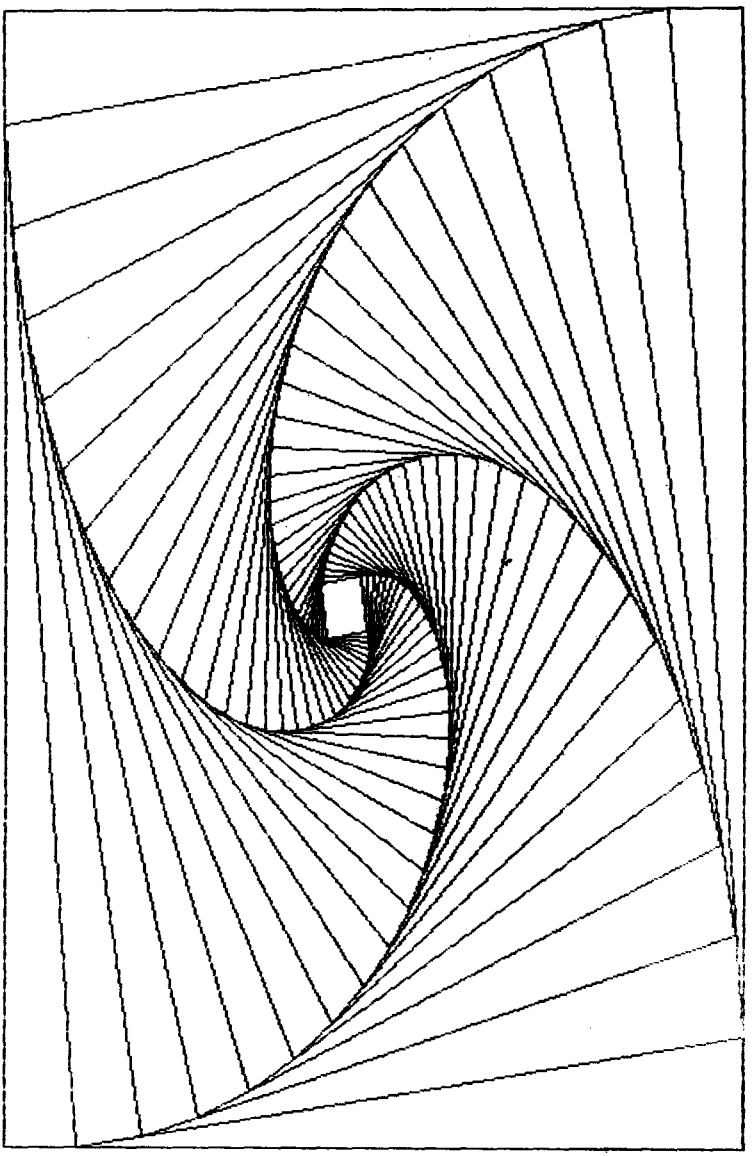
Программа рисует узор, образованный тридцатью вложенными прямоугольниками. В начале программы включается ниша **"graphics"**, содержащая определения функций управления экраном и графику, а также нишу "low_user" обращений к прерываниям ДОС. (Подробнее эти функции описаны в пункте 18 главы 1).

В программе используются макроопределения трех переменных: цвета рисунка и фона, коэффициент сдвига координат вершин следующего прямоугольника. Так что при желании эти значения легко изменить.

Описанная функция "square" рисует прямоугольник. Пересчет координат вершин следующего прямоугольника происходит в функции "nextsquare".

Для задержки изображения на экране используется функция "getch", которая ждет нажатия какой-либо клавиши для продолжения выполнения программы.

Пример решения:



```
---  Узор, образованный вложенными прямоугольниками  ---

graphics
--- функции управления экраном и графика

#define color       2     --- цвет рисунка
#define backcolor  14     --- цвет фона
#define koeff     0.9     --- коэффициент сдвига

--- координаты вершин прямоугольника
int x1, y1, x2, y2, x3, y3, x4, y4;

main()
{
 x1 = 0;   y1 = 0;
 x2 = 319; y2 = 0;
 x3 = 319; y3 = 199;
 x4 = 0;   y4 = 199;
 s_setmode(4);           --- установка графического режима
 s_setback(backcolor); --- установка цвета фона
 s_setpal(1);            --- установка цветовой палитры
 square(color);
 {
  * 30
  {
   nextsquare();
   square(color);
  }
 }
 getch();                --- запрос нажатия клавиши
 s_setmode(3);           --- возврат в текстовой режим
}

square(clr)              --- рисование прямоугольника
int clr;
{
 s_drawline(x1, y1, x2, y2, clr);
 s_drawline(x2, y2, x3, y3, clr);
 s_drawline(x3, y3, x4, y4, clr);
 s_drawline(x4, y4, x1, y1, clr);
}

nextsquare()             --- вычисление новых координат
{
 int px1, py1, px2, py2, px3, py3, px4, py4;
 px1 = (int)(x1 + (x2 - x1) * koeff);
 py1 = (int)(y1 + (y2 - y1) * koeff);
 px2 = (int)(x2 + (x3 - x2) * koeff);
 py2 = (int)(y2 + (y3 - y2) * koeff);
 px3 = (int)(x3 + (x4 - x3) * koeff);
 py3 = (int)(y3 + (y4 - y3) * koeff);
 px4 = (int)(x4 + (x1 - x4) * koeff);
 py4 = (int)(y4 + (y1 - y4) * koeff);
 x1 = px1; y1 = py1; x2 = px2; y2 = py2;
 x3 = px3; y3 = py3; x4 = px4; y4 = py4;
}
```

```
-------------------------------------------------------GRAPHICS---
---       Функции управления экраном и графика              ---
------------------------------------------------------------------
low_user
--- обращения к прерываниям ДОС

------------------------------------------------------------------
---                запрос режима экрана                        ---
#define s_mode()          (a86(16, 15, 0, 0, 0) & 255)

---              установка режима экрана                       ---
#define s_setmode(m)      a86(16, 0, (m), 0, 0)

---                  рисование точки                           ---
#define s_drawdot(c,l,clr)  a86(16, 12, (clr), (c), (l))

---               запрос цвета в точке                         ---
#define s_dot(c,l)        (a86(16, 13, 0, (c), (l)) & 255)

---                установка цвета фона                        ---
#define s_setback(n)      v86bh(11, 0, (n))

---                  установка палитры                         ---
#define s_setpal(n)       v86bh(11, 1, (n))

---                  рисование линии                           ---
s_drawline(ax, ay, bx, by, clr)
int ax, ay, bx, by, clr;
{
 int xdev = 0, ydev = 0, dx, dy, d, xstep, ystep, i;
 dx = bx - ax; dy = by - ay;
 xstep = (dx < 0)? -1: (dx > 0);
 ystep = (dy < 0)? -1: (dy > 0);
 dx = abs(dx); dy = abs(dy);
 d = (dx > dy)? dx : dy;
 {
 * d + 2
 }
  s_drawdot(ax,ay,clr);
  xdev += dx; ydev += dy;
  [?  (xdev > d) xdev -= d; ax += xstep;
  [?  (ydev > d) ydev -= d; ay += ystep;
 }
}
------------------------------------------------------------------.
```

```
-------------------------------------------------------LOW_USER---
---        Функции обращения к прерываниям ДОС                 ---
------------------------------------------------------------------
#include <dos.h>
#include <stdlib.h>
------------------------------------------------------------------
unsigned int a86(i, ah, al, cx, dx)
int i, ah, al, cx, dx;
┌
| union REGS inr, outr;
| inr.h.ah = ah; inr.h.al = al;
| inr.x.bx = 0;
| inr.x.cx = cx; inr.x.dx = dx;
| int86(i, &inr, &outr);
<——(outr.x.ax)
└
------------------------------------------------------------------
unsigned int v86bh(ah, bh, bl)
int ah, bh, bl;
┌
| union REGS inr, outr;
| inr.h.ah = ah;
| inr.h.bh = bh; inr.h.bl = bl;
| int86(16, &inr, &outr);
<——(outr.h.bh)
└
------------------------------------------------------------------
```

Л И Т Е Р А Т У Р А

1. Абрамов С.А., Гнездилова Г.Г., Капустина Е.Н., Селюн М.И. Задачи по программированию. - М. : Наука, 1988. - 224 стр.
2. Абрамов С.А., Зима Е.В. Начала информатики. - М. : Наука, 1989. - 256 стр.
3. Берри Р., Микинз Б. Язык Си. Введение для программистов. - М. : Финансы и статистика, 1988. - 191 стр.
4. Болски М.И. Язык программирования Си. Справочник. - - М. : Радио и связь, 1988 - 96 стр.
5. Гринчишин Я.Т., Ефимов В.И., Лсчакович А.Н. Алгоритмы и программы на Бейсике. - М. : Просвещение, 1988. - 159 стр.
6. Дьяконов В.П. Справочник по алгоритмам и программам на языке Бейсик для персональных ЭВМ. - М. : Наука, 1989. - 240 стр.
7. Кихо Ю.К. Схематическое программирование // Труды выч. центра - 1984. - Вып. 50. - С. 52 - 68.
8. Кихо Ю.К. Схематическая форма записи структур // Труды выч. центра - 1985. - Вып. 52. - С. 55 - 78.
9. Кихо Ю., Куллман К. Стандарт схематического языка и схемный редактор // Труды выч. центра - 1989. - Вып. 57. - С. 93 - 110.
10. Кихо Ю., Сакс Э. Система схематического программирования для ЕС ЭВМ // Труды выч. центра - 1988. - Вып. 55. - С. 65 - 81.
11. Кренкель Т.Э., Коган А.Г., Тараторин А.М. Персональные ЭВМ в инженерной практике. - М. : Радио и связь, 1989. - С.
12. Kiho, J. Arvutiõpetus 2. TÜ. Tartu. 1989.

П Р И Л О Ж Е Н И Е
Список терминов

Настоящее методическое пособие предназначено для студентов математического факультета, изучающих предмет "системное программирование".

Утверждено на заседании совета математического факультета ТУ 23 ноября 1990 года

ОСНОВЫ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ II.
Составители Юри К и х о и др.
На русском языке.
Тартуский университет.
ЭР, 202400, г.Тарту, ул.Юликооли, I8.
Ответственный редактор Т. Роосмаа.
Подписано к печати 3.I2.1990.
Формат 60x84/I6.
Бумага ротаторная.
Машинопись. Ротапринт.
Условно-печатных листов 6,5I.
Учетно-издательских листов 6,3. Печатных листов 7,0.
Тираж 200.
Заказ № 84I.
Цена I руб. 80 коп.
Типография ТУ, ЭР, 202400, г.Тарту, ул.Тийги, 78.