ШКОЛА – КОЛЛЕДЖ – УНИВЕРСИТЕТ

Н. И. КОСТЮКОВА

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ СИ

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ И ЗАДАЧИ ПО ПРОГРАММИРОВАНИЮ

СИБИРСКОЕ УНИВЕРСИТЕТСКОЕ ИЗДАТЕЛЬСТВО

Серия: **Школа – Колледж – Университет**

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ВЫСШИЙ КОЛЛЕДЖ ИНФОРМАТИКИ

Н. И. Костюкова

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ СИ

Методические рекомендации
и задачи по программированию

СИБИРСКОЕ УНИВЕРСИТЕТСКОЕ ИЗДАТЕЛЬСТВО
НОВОСИБИРСК • 2017

УДК 681.3.06(075)
ББК 32.973-01я73
К72

Рецензенты:
д.ф.-м.н, профессор НГУ,
ст. науч. сотр. Института вычислительной математики
и математической геофизики СО РАН РФ *В. А. Огородников*
ст. преп. кафедры информатики ВКИ НГУ,
мл. науч. сотр. Института информационных технологий *А. И. Куликов*

Костюкова Н. И.

К72 Программирование на языке Си. Методические рекомендации и задачи по программированию. — Новосибирск: Сиб. унив. изд-во, 2017. — 158 с. — (Школа – Колледж – Университет)

ISBN 978-5-379-02016-3

Данное пособие предназначено для обучения программистов-практиков. Материал ориентирован на повседневную работу за терминалом ЭВМ. Все конструкции языка Си, независимо от частоты их использования, синтаксической и семантической сложности, описаны исчерпывающе, неформально, но довольно строго, проиллюстрированы краткими примерами. Материал составлен на основе классических работ по языку программирования Си. Реализация сложных задач представлена в приложении.

Для преподавателей информатики, студентов высших и средних специальных учебных заведений, а также учителей информатики средних школ.

УДК 681.3.06(075)
ББК 32.973-01я73

ISBN 978-5-379-02016-3

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Язык программирования Си был разработан и реализован в 1972 году сотрудником фирмы AT&T Bell Laboratories Деннисом Ритчи. Хотя язык Си появился относительно недавно, популярность его росла очень быстро, и в настоящее время компиляторы этого языка созданы для многих машин. Растущая популярность языка Си обусловлена следующими двумя важными причинами. Во-первых, язык Си очень гибок: его можно относительно просто использовать в различных областях приложений. Во-вторых, большая часть программного обеспечения популярной операционной системы (ОС) UNIX написана на языке Си, который является основным языком программирования в этой системе.

Язык Си обеспечивает программисту большую свободу. Эта свобода — источник большой выразительности и один из главных источников силы языка Си, делающих его эффективным, универсальным и простым в использовании для различных областей применения.

Язык Си является процедурным языком, доминирующим в области промышленного программирования. Развитие языка Си шло по пути повышения его надежности и улучшения средств диагностики ошибок за счет усиления типизации в нем. Вседозволенность в языке Си является следствием желания расширить как можно больше область его применения. Язык Си удалось сделать относительно маленьким языком программирования за счет того, что в его состав не были включены ввод-вывод и средства для работы со строками. Практический опыт использования языка Си показал правильность такого подхода. Язык Си является языком со слабой типизацией данных. В нем допускается неявное преобразование типов для всех базовых типов и указателей. Однако мобильный транслятор с языка Си выводит предупреждение о каждом встречающемся в программе случае неявного преобразования типов, в котором участвует указатель.

1. ОБЩИЙ СИНТАКСИС

1.1. Формат

Пробелы, символы табуляции, перевода на новую строку и перевода страницы используются как разделители. Можно использовать любое количество таких символов. Для повышения читабельности текста рекомендуется использовать символы табуляции.

1.2. Комментарии

Комментарии начинаются парой символов /* и заканчиваются парой символов */. Разрешены везде, где допустимы пробелы. Комментарий может начинаться на одной строке текста программы и заканчиваться на другой строке.

Примеры:

```
/*Однострочный комментарий*/
/* Многострочный
* комментарий
*/в программе
```

1.3. Идентификаторы

Идентификаторы используются как имена переменных, функций и типов данных. Допустимые символы: цифры 0–9, латинские прописные и строчные буквы a–z, A–Z, символ подчеркивания (_). Первый символ не может быть цифрой.

Идентификатор может быть произвольной длины, но в некоторых ЭВМ не все символы учитываются компилятором и загрузчиком.

Примеры:
NAME1
name1
Total_5
Paper

Число значимых символов и виды букв (прописные/строчные) в идентификаторах могут различаться даже на однотипных ЭВМ в зависимости от используемых компиляторов и загрузчиков.

1.4. Зарезервированные слова

Типы данных	*Классы памяти*	*Операторы*
char	auto	break
double	extern	case
enum	register	continue
float	static	default
int		do
long		else
short		for
struct		goto
union		if
unsigned		return
void		switch
sizeof		while
typedef		

Замечание. sizeof — это операция, выполняющаяся во время компиляции. Описание typedef используется для определения сокращенной формы описания существующего типа данных. В некоторых реализациях, кроме того, зарезервированы слова asm и fortran.

2. ОСНОВНЫЕ ТИПЫ ДАННЫХ

К основным типам данных относятся целые числа (int, short, long, unsigned), символы (char) и числа с плавающей точкой (float,

double). На их основе строятся производные типа данных. В этом разделе описаны синтаксис констант и объем памяти, занимаемой основными типами данных.

2.1. Целые константы

Десятичные: цифры 0–9;
первой цифрой не должен быть 0.

Примеры:

```
12
111
956
1007
```

Замечание. Если значение превышает наибольшее машинное целое со знаком, то оно представляется как длинное целое.

Восьмеричные: цифры 0–7;
начинаются с 0.

Примеры:

```
012 = 10 /*десятичное*/;
0111 = 73 /*десятичное*/;
076 = 62 /*десятичное*/;
0123 = 1*64+2*8+3 = 83 /*десятичное*/;
```

Замечание. Если значение превышает наибольшее машинное целое со знаком, то оно представляется как длинное целое.

Шестнадцатеричное: цифры 0–9, буквы a–f или A–F
для значений 10–15;
начинаются с 0x или 0X.

Примеры:

```
0x12 = 18 /*десятичное*/;
0X12 = 18 /*десятичное*/;
0x2f = 47 /*десятичное*/;
0XA3 = 163 /*десятичное*/;
0x1B9 = 1*256+11*16+9 = 441 /*десятичное*/;
```

Замечание. Если значение превышает наибольшее машинное целое со знаком, то оно представляется как длинное целое.

2.2. Длинные целые константы

Длинная целая константа явно определяется латинской буквой l или L, стоящей после константы.

Примеры:

Длинная десятичная: 12l = 12 /*десятичное*/;
956L = 956 /*десятичное*/;
Длинная восьмеричная: 012l = 10 /*десятичное*/;
076L = 62 /*десятичное*/;
Длинная шестнадцатеричная: 0x12l =18 /*десятичное*/;
0XA3L = 163 /*десятичное*/;

2.3. Константы с плавающей точкой

Константа с плавающей точкой всегда представляется числом с плавающей точкой двойной точности, т. е. как имеющая тип double, и состоит из следующих частей:

целой части — последовательности цифр;

десятичной точки;

дробной части — последовательности цифр;

символа экспоненты e или E;

экспоненты в виде целой константы (может быть со знаком − или +).

Любая часть (но не обе сразу) из нижеследующих пар может быть опущена:

целая или дробная часть;

десятичная точка или символ e (E) и экспонента в виде целой константы.

Примеры:

345. = 345 /*десятичное*/;
3.14159 = 3.14159 /*десятичное*/;
2.1E5 = 210000 /*десятичное*/;
.123E3 = 123 /*десятичное*/;
4037e-5 = .04037 /*десятичное*/;

2.4. Символьные константы

Символьная константа состоит из одного символа кода ASCII, заключенного в апострофы (').

Примеры:

'A'
'a'
'7'
'$'

Специальные (управляющие) символьные константы

Новая строка (перевод строки)	HL (LF)	'\n'
Горизонтальная табуляция	HT	'\t'
Вертикальная табуляция	VT	'\v'
Возврат на шаг	BS	'\b'
Возврат каретки	CR	'\r'
Перевод формата	FF	'\f'
Обратная косая	\	'\\'
Апостроф	'	'\''
Кавычки	"	'\"'
Нулевой символ	NUL	'\0'

Кроме этого, любой символ может быть представлен последовательностью трех восьмеричных цифр: '\ddd'.

Замечание. Символьные константы считаются данными типа int.

2.5. Строковые константы

Строковая константа представляется последовательностью символов кода ASCII, заключенной в кавычки ("…"). Она имеет тип char[].

Примеры:

"This is character string"
"Это строковая константа"
"A" "1234567890" "0" "$"

В конце каждой строки компилятор помещает нулевой символ '\0', отмечающий конец данной строки.

Каждая строковая константа, даже если она идентична другой строковой константе, сохраняется в отдельном месте памяти.

Если необходимо ввести в строку символ кавычек ("), то перед ним надо поставить символ обратной косой (\). В строку могут быть введены любые специальные символьные константы, перед которыми стоит символ \. Символ \ и следующий за ним символ новой строки игнорируется.

2.6. Перечислимые константы

Имена, указанные в описании перечислимых констант, трактуются как целые константы.

3. ОПЕРАЦИИ И ВЫРАЖЕНИЯ

3.1. Выражения

Выражение состоит из одного или большего числа операндов и символов операций.

Примеры:

```
a++;
b = 10;
x = (y*z)/w;
```

Замечание. Выражение, заканчивающееся точкой с запятой, является оператором.

3.2. Метаобозначения операндов

Некоторые операции требуют операндов определенного вида. Вид операнда обозначается одной из следующих букв:

e — любое выражение;

v — любое выражение, ссылающееся на переменную, которой может быть присвоено значение. Такие выражения называются адресными.

Префикс указывает тип выражения. Например, ie обозначает произвольное целое выражение. Ниже описываются все возможные префиксы:

i — целое число или символ;

a — арифметическое выражение (целое число, символ или число с плавающей точкой);

p — указатель;

s — структура или объединение;

sp — указатель на структуру или объединение;

f — функция;

fp — указатель на функцию.

Обозначение smem указывает на имя элемента структуры или объединения.

Замечание. Если в выражении должно быть несколько операндов, то они отличаются номерами, например: ae1 + ae2.

3.3. Арифметические операции

\+ Использование: ae1 + ae2

Сумма значений ae1 и ae2.

Пример:

```
i = j + 2;
```

Устанавливает i равным j плюс 2.

\+ Использование: pe + ie

Адрес переменной типа pe, больший на ie адреса, заданного указателем pe.

Пример:

```
last = arname + arsize - 1;
```

Присваивает переменной last адрес последнего элемента массива arname.

\- Использование: ae1 - ae2

Разность значений ae1 и ae2.

Пример:

```
i = j - 3;
```

\- Использование: pe - ie

Адрес переменной типа pe, меньший на ie адреса, заданного указателем pe.

Пример:

```
first = last - arsize + 1;
```

- Использование: pe1 - pe2

  Число переменных типа pe в диапазоне от pe2 до pe1.

Пример:

```
arsize = last - first;
```

- Использование: -ae

  Изменение знака ae.

Пример:

```
x = -x;
```

* Использование: ae1 * ae2

  Произведение значений ae1 и ae2.

Пример:

```
z = 3 * x;
```

/ Использование: ae1 / ae2

  Частное от деления ae1 на ae2.

Пример:

```
i = j / 5;
```

% Использование: ae1 % ae2

  Остаток от деления (деление по модулю) ae1 на ae2.

Пример:

```
minutes = time % 60;
```

++ Использование: iv++

Увеличение iv на 1. Значением этого выражения является значение iv до увеличения.

Пример:

```
j = i++;
```

++ Использование: pv++

Увеличение указателя pv на 1, так что он будет указывать на следующий объект того же типа. Значением этого выражения является значение pv до увеличения.

Пример:

```
*ptr++ = 0;
```

Присвоить значение 0 переменной, на которую указывает ptr, затем увеличить значение указателя ptr так, чтобы он указывал на следующую переменную того же типа.

++ Использование: ++iv

Увеличение iv на 1. Значением этого выражения является значение iv после увеличения.

Пример:

```
i = ++j;
```

++ Использование: ++pv

Увеличение pv на 1. Значением этого выражения является значение pv после увеличения.

Пример:

```
*++ptr = 0;
```

-- Использование: iv--

Уменьшение iv на 1. Значением этого выражения является значение iv до уменьшения.

Пример:

```
j = i--;
```

-- Использование: pv--

Уменьшение указателя pv на 1 так, что он будет указывать на предыдущий объект того же типа. Значением этого выражения является значение pv до уменьшения.

Пример:

```
arrpos = p--;
```

-- Использование: --iv

Уменьшение iv на 1. Значением этого выражения является значение iv после уменьшения.

Пример:

```
i = --j;
```

-- Использование: --pv

Уменьшение pv на 1. Значением этого выражения является значение pv после уменьшения.

Пример:

```
prepos = --p;
```

Замечание. При выполнении операций ++ и -- появляется побочный эффект — изменяется значение переменной, используемой в качестве операнда.

3.4. Операция присваивания

Замечание. Значением выражения, в которое входит операция присваивания, является значение левого операнда после присваивания.

= Использование: v = e
Присваивание значения e переменной v.

Пример:

```
x = y;
```

Замечание. Следующие операции объединяют арифметические или побитовые операции с операцией присваивания.

+= Использование: av += ae
Увеличение av на ae.

Пример:

```
y += 2;
```

Увеличение переменной y на 2.

+= Использование: pe += ie
Увеличение pe на ie.

Пример:

```
p += n;
```

-= Использование: av -= ae
Уменьшение av на ie.

Пример:

```
x -= 3;
```

-= Использование: pv -= ie
Уменьшение pv на ie.

Пример:

```
ptr -= 2;
```

*= Использование: av *= ae
Умножение av на ae.

Пример:

```
tamesx *= x;
```

/= Использование: av /= ae
Деление av на ae.

Пример:

```
x /= 2;
```

%= Использование: iv %= ie

Значение iv по модулю ie.

Пример:

```
x %= 10;
```

>>= Использование: iv >>= ie

Сдвиг двоичного представления iv вправо на ie бит.

Пример:

```
x >>= 4;
```

<<= Использование: iv <<= ie

Сдвиг двоичного представления iv влево на ie бит.

Пример:

```
x <<= 1;
```

&= Использование: iv &= ie

Побитовая операция И двоичных представлений iv и ie.

Пример:

```
remitems &= mask;
```

^= Использование: iv ^= ie

Побитовая операция исключающее ИЛИ двоичных представлений iv и ie.

Пример:

```
control ^= seton;
```

|= Использование: iv |= ie

Побитовая операция ИЛИ двоичных представлений iv и ie.

Пример:

```
additems |= mask;
```

3.5. Операции отношения

Замечание. Логическое значение Ложь представляется целым ненулевым значением.

Значением выражений, содержащих операции отношения или логические операции, является 0 (Ложь) или 1 (Истина).

== Использование: ie1 == ie2

Истина, если ie1 равно ie2; иначе Ложь.

Пример:

```
if (i == 0)
break;
```

== Использование: pe1 == pe2

Истина, если значение указателей pe1 и pe2 равны.

!= Использование: ie1 != ie2

Истина, если ie1 не равно ie2.

Пример:

```
while (i != 0)
i = func;
```

!= Использование: pe1 != pe2

Истина, если значения указателей pe1 и pe2 не равны.

Пример:

```
if (p != q)
break;
```

< Использование: ae1 < ae2

Истина, если ae1 меньше, чем ae2.

Пример:

```
if (x < 0)
printf ("negative");
```

< Использование: pe1 < pe2

Истина, если значение pe1 (т. е. некоторый адрес) меньше, чем значение pe2.

Пример:

```
while (p < q)
*p++ = 0;
```

Пока адрес, заданный p, меньше, чем адрес, заданный q, присваивать значение 0 переменной, на которую указывает p, и увеличивать значение p так, чтобы этот указатель указывал на следующую переменную.

<= Использование: ae1 <= ae2

Истина, если ae1 меньше или равно ae2.

<= Использование: pe1 <= pe2
Истина, если pe1 меньше или равно pe2.

> Использование: ae1 > ae2
Истина, если ae1 больше, чем ae2.

Пример:

```
if (x > 0)
printf("positive");
```

> Использование: pe1 > pe2
Истина, если значение pe1 (т. е. некоторый адрес) больше, чем значение pe2.

Пример:

```
while (p > q)
*p-- = 0;
```

>= Использование: ae1 >= ae2
Истина, если ae1 больше или равно ae2.

>= Использование: pe1 >= pe2
Истина, если значение pe1 больше или равно значению pe2.

3.6. Логические операции

! Использование: !ae или !pe
Истина, если ae или pe ложно.

Пример:

```
if (!good)
printf ("no good");
```

|| Использование: e1 || e2
Логическая операция ИЛИ значений e1 и e2. Вначале проверяется значение e1; значение e2 проверяется только в том случае, если значение e1 — Ложь. Значением выражения является Истина, если истинно значение e1 ИЛИ e2.

Пример:

```
if (x<A || x>B)
printf ("out of range");
```

&& Использование: e1 && e2
Логическая операция И значений e1 и e2. Вначале проверяется значение e1; значение e2 проверяется только в том случае, если

значение e1 — Истина. Значением выражения является Истина, если истинно значение e1 И e2.

Пример:

```
if (p != NULL && *p>7)
n++;
```

Если p — не нулевой указатель и значение переменной, на которую указывает p, больше чем 7, то увеличить n на 1.

Обратите внимание: если значение указателя p равно NULL (0), то выражение *p не имеет смысла.

3.7 Побитовые операции

~ Использование: ~ie

Дополнение до единицы значения ie. Значение выражения содержит 1 во всех разрядах, в которых ie содержит 0, и 0 во всех разрядах, в которых ie содержит 1.

Пример:

```
opposite = ~mask;
```

>> Использование: ie1 >> ie2

Двоичное представление ie1 сдвигается вправо на ie2 разрядов. Сдвиг вправо может быть арифметическим (т. е. освобождающиеся слева разряды заполняются значением знакового разряда) или логическим в зависимости от реализации, однако гарантируется, что сдвиг вправо целых чисел без знака будет логическим и освобождающиеся слева разряды будут заполнены нулями.

Пример:

```
x = x >> 3;
```

<< Использование: ie1 << ie2

Двоичное представление ie1 сдвигается влево на ie2 разрядов; освобождающиеся справа разряды заполняются нулями.

Пример:

```
fourx = x << 2;
```

& Использование: ie1 & ie2

Побитовая операция И двоичных представлений ie1 и ie2. Значение выражений содержит 1 во всех разрядах, в которых и ie1, и ie2 содержат 1, и 0 во всех остальных разрядах.

Пример:

```
flag = ((x & mask) != 0);
```

| Использование: ie1 | ie2

Побитовая операция ИЛИ двоичных представлений ie1 и ie2. Значение выражений содержит 1 во всех разрядах, в которых ie1 или ie2 содержит 1, и 0 во всех остальных разрядах.

Пример:

```
attrsum = attr1 | attr2;
```

^ Использование: ie1 ^ ie2

Побитовая операция исключающее ИЛИ двоичных представлений ie1 и ie2. Значение выражений содержит 1 в тех разрядах, в которых ie1 и ie2 имеют разные двоичные значения, и 0 во всех остальных разрядах.

Пример:

```
diffbits = x ^ y;
```

3.8. Адресные операции

& Использование: &v

Значением выражения является адрес переменной v.

Пример:

```
intptr = &n;
```

* Использование: *pe

Значением выражения является переменная, адресуемая указателем pe.

Пример:

```
*ptr = c;
```

* Использование: *fpe

Значением выражения является функция, адресуемая указателем fpe.

Пример:

```
fpe = funcname;
(*fpe) (arg1,arg2);
```

3.9. Операции над массивами

[] Использование: pe[ie]

Значением выражения является переменная, отстоящая на ie переменных от адреса, заданного pe. Это значение эквивалентно значению выражения *(pe + ie).

Пример:

```
arname[i]=3;
```

Присвоить значение 3 i-му элементу массива arname.

Обратите внимание: первый элемент массива описывается выражением arname[0].

3.10. Операции над структурами и объединениями

. Использование: sv.smem

Значением выражения является элемент smem структуры (или объединения) sv.

Пример:

```
product.p_revenue = 50;
```

Присвоить значение 50 элементу p_revenue структурной переменной product.

-> Использование: spe->smem

Значением выражения является элемент smem структуры (или объединения), на которую(ое) указывает spe. Это значение эквивалентно значению выражения (*spe).smem.

Пример:

```
prodptr->p_revenue = 2;
```

Присвоить значение 2 элементу p_revenue структурной переменной, на которую указывает prodptr.

3.11. Другие операции

?: Использование: ae ? e1 : e2 или pe ? e1 : e2

Если истинно ae или pe, то выполняется e1; иначе выполняется e2. Значением этого выражения является значение выражения e1 или e2.

Пример:

```
abs = (i<=0) ? -i : i;
```

, Использование: e1,e2

Сначала выполняется выражение e1, потом выражение e2. Значением всего выражения является значение выражения e2.

Пример:

```
for (i=A, j=b; i<j; i++, j--)
p[i] = p[j];
```

sizeof Использование: sizeof(e)

Число байт, требуемых для размещения данных типа e. Если e описывает массив, то в этом случае e обозначает весь массив, а не только адрес первого элемента, как во всех остальных операциях.

sizeof Использование: sizeof(*тип*)

Число байт, требуемых для размещения объектов типа «*тип*».

Пример:

```
n = sizeof(arname)/sizeof(int);
```

Число элементов в массиве целых чисел определяется как число байт в массиве, поделенное на число байт, занимаемых одним элементом массива.

(*тип*) Использование: (*тип*)e

Значение e, преобразованное в тип данных «*тип*».

Пример:

```
x = (float)n/3;
```

Целое значение переменной n преобразуется в число с плавающей точкой перед делением на 3.

() Использование: fe(e1,e2,...,eN)

Вызов функции fe с аргументами e1, e2, … , eN. Значением выражения является значение, возвращаемое функцией.

Обратите внимание: порядок выполнения выражений e1, e2, … , eN не гарантируется.

Пример:

```
x = sqrt(y);
```

3.12. Приоритеты и порядок выполнения операций

Для каждой группы операций в нижеследующей таблице приоритеты одинаковы. Чем выше приоритет группы операций,

тем выше она расположена в таблице. Порядок выполнения определяет группировку операций и операндов (слева направо или справа налево), если отсутствуют скобки и операции относятся к одной группе.

Примеры:

Выражение a*b / c эквивалентно выражению (a*b) / c, так как операции выполняются слева направо.

Выражение a + b / c не эквивалентно выражению (a + b) / c, так как операции выполняются справа налево.

Слева направо

() Вызов функции

[] Выделение элемента массива

. Выделение элемента структуры или объединения

-> Выделение элемента структуры (объединения), адресуемой(го) указателем

Справа налево

! Логическое отрицание

~ Побитовое отрицание

- Изменение знака

++ Увеличение на единицу

-- Уменьшение на единицу

& Определение адреса

* Обращение по адресу

(*тип*) Преобразование типа

sizeof Определение размера в байтах

Слева направо

* Умножение

/ Деление

% Деление по модулю

+ Сложение

- Вычитание

<< Сдвиг влево

>> Сдвиг вправо

< Меньше
<= Меньше или равно
> Больше
>= Больше или равно
== Равно
!= Не равно
& Побитовая операция И
^ Побитовая операция исключающее ИЛИ
| Побитовая операция ИЛИ
&& Логическая операция И
|| Логическая операция ИЛИ
, Операция запятая

Справа налево
?: Условная операция
= Присваивание
*= /= %= += -= Арифметические операции
<<= >>= &= ^= |= Операции отношения

3.13. Порядок обработки операндов

Для четырех операций (&& || ?: ,) гарантируется, что левый операнд будет обрабатываться первым. Для остальных операций порядок обработки может быть разным на разных компиляторах. Это означает, что если программа, отлаженная на некоторой ЭВМ, зависит от негарантированного порядка обработки операндов, то на другой ЭВМ с другим компилятором она может выполняться неправильно.

Пример:

```
v = (x = 5) + (++x);
```

Если порядок обработки операндов в операции + слева направо, то переменная v получит значение 11 (5 + 6) и значение x будет равно 6.

Если порядок справа налево, то значение v зависит от значения x, которое эта переменная имела до выполнения выражения; например, если значение x было равно 0, то значение v станет равным 6 (5 + 1), а значение x — равным 5.

Предупреждение. Если вы присваиваете переменной значение в любом выражении (включая вызов функции), то не используйте эту переменную снова в том же выражении. Например, если в предыдущем примере необходим порядок обработки слева направо, сделайте так:

```
x = 5;
v = x + (x + 1);
++x;
```

3.14. Арифметические преобразования в выражениях

Прежде всего каждый операнд типа char или short преобразуется в значение типа int и операнды типа unsigned shar или unsigned short преобразуются в значение типа unsigned int.

Затем, если один из операндов имеет тип double, то другой преобразуется в значение типа double, и результат будет иметь тип double.

Иначе, если один из операндов имеет тип unsigned long, то другой преобразуется в значение типа unsigned long, и таким же будет тип результата.

Иначе, если один из операндов имеет тип long, то другой преобразуется в значение типа long, и таким же будет тип результата.

Иначе, если один из операндов имеет тип long, а другой — тип unsigned int, то оба операнда преобразуются в значение типа unsigned long, и результат будет иметь тип unsigned long.

Иначе, если один из операндов имеет тип unsigned, то другой преобразуется в значение типа unsigned, и результат будет иметь тип unsigned.

Иначе оба операнда должны быть типа int, и таким же будет тип результата.

4. ОПЕРАТОРЫ

4.1. Формат и вложенность

Формат: один оператор может занимать одну или более строк. Два или большее количество операторов могут быть расположены на одной строке.

Вложенность: операторы, управляющие порядком выполнения (if, if-else, switch, while, do-while и for), могут быть вложены друг в друга.

4.2. Метка оператора

Метка может стоять перед любым оператором, чтобы на этот оператор можно было перейти с помощью оператора goto.

Метка состоит из идентификатора, за которым стоит двоеточие (:). Областью определения метки является данная функция.

Пример:

```
ABC2: x = 3;
```

4.3. Составной оператор

Составной оператор (блок) состоит из одного или большего числа операторов любого типа, заключенных в фигурные скобки ({}). После закрывающейся фигурной скобки не должно быть точки с запятой (;).

Пример:

```
{x = 1; y = 2; x = 3}
```

4.4. Оператор-выражение

Любое выражение, заканчивающееся точкой с запятой (;), является оператором. Ниже приведены операторы выражения.

Оператор присваивания

```
Идентификатор = выражение;
```

Пример:

```
x = 3;
```

Оператор вызова функции

```
Имя функции (аргумент1, … , аргументN);
```

Пример:

```
fclose (file);
```

Пустой оператор

Состоит только из точки с запятой (;). Используется для обозначения пустого тела управляющего оператора.

4.5. Оператор завершения break

break;

Прекращает выполнение ближайшего вложенного внешнего оператора switch, while, do или for. Управление передается оператору, следующему за заканчивающимся. Одно из назначений этого оператора — закончить выполнение цикла при присваивании некоторой переменной определенного значения.

Пример:

```
for (i = 0; i < n; i++)
if ((a[i] = b[i]) == 0)
break;
```

4.6. Оператор продолжения continue

continue;

Передает управление в начало ближайшего внешнего оператора цикла while, do или for, вызывая начало следующей итерации. Этот оператор по действию противоположен оператору break.

Пример:

```
for (i = 0; i < n; i++){
if (a[n] != 0)
continue;
a[i] = b[i];
k++;
}
```

4.7. Оператор возврата return

return;

Прекращает выполнение текущей функции и возвращает управление вызвавшей программе.

return *выражение*;

Прекращает выполнение текущей функции и возвращает управление вызвавшей программе с передачей значения выражения.

Пример:

```
return x+y;
```

4.8. Оператор перехода goto

goto *метка*;

Управление безусловно передается на оператор с меткой «*метка*».

Используется для выхода из вложенных управляющих операторов. Область действия ограничена текущей функцией.

Пример:

```
goto ABC;
```

4.9. Условный оператор if-else

if (*выражение*)
**   *оператор***

Если выражение истинно, то выполняется оператор.

Если выражение ложно, то ничего не делается.

Пример:

```
if (a == x)
temp = 3;
temp = 5;
```

if (*выражение*)
***оператор*1**
else
***оператор*2**

Если выражение истинно, то выполняется *оператор*1 и управление передается на оператор, следующий за *оператором*2 (т. е. *оператор*2 не выполняется).

Если выражение ложно, то выполняется *оператор*2.

Часть else оператора может опускаться. Поэтому во вложенных операторах if с пропущенной частью else может возникнуть неоднозначность. В этом случае else связывается с ближайшим предыдущим оператором if в том же блоке, не имеющем части else.

Примеры:

Часть else относится ко второму оператору if:

```
if (x > 1)
    if (y == 2)
        z = 5;
```

```
    else
z = 6;
```

Часть else относится к первому оператору if:

```
if (x > 1) {
    if (y == 2)
        z = 5;
} else
        z =6;
```

Вложенные операторы if:

```
if (x == 'a')
    y = 1;
else if (x == 'b') {
    y = 2;
    z = 3;
} else if (x == 'c')
    y = 4;
else
    printf("ERROR");
```

4.10. Оператор-переключатель switch

switch (*выражение*) {
case *константа*: *операторы*
case *константа*: *операторы*
. . . .
default: *операторы*
}

Сравнивает значение выражения с константами во всех вариантах case и передает управление оператору, который соответствует значению выражения. Каждый вариант case может быть помечен целой или символьной константой либо константным выражением. Константное выражение не может включать переменные или вызовы функций.

Примеры:

Правильно: case 3 + 4:

Неправильно: case X + Y:

Операторы, связанные с меткой default, выполняются, если ни одна из констант в операторах case не равна значению выражения. Вариант default не обязательно должен быть последним.

Если ни одна константа не соответствует значению выражения и остается только вариант default, то не выполняется никаких действий.

Ключевое слово case вместе с константой служит просто меткой, и если будут выполняться операторы для некоторого варианта case, то далее будут выполняться операторы всех последующих вариантов до тех пор, пока не встретится оператор break. Это позволяет связывать одну последовательность операторов с несколькими вариантами.

Никакие две константы в одном операторе-переключателе не могут иметь одинаковые значения.

Пример:

```
switch (x) {
                case 'A':
                        printf ("CASE A\n");
                        break;
                case 'B':
                case 'C':
                        printf("CASE B or C\n");
                        break;
                default:
                        printf("NOT A, B or C\n");
                        break;
}
```

Наиболее общая синтаксическая форма оператора switch:

switch (*выражение*) *оператор*

Пример:

```
switch (x)
case 2:
case 4:
y = 3;
```

4.11. Оператор цикла while

while (*выражение*)
**   *оператор***

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

Если выражение ложно, то управление передается следующему оператору.

Замечание. Значение выражения определяется до выполнения оператора. Следовательно, если выражение ложно с самого начала, то оператор вообще не выполняется.

Пример:

```
while (k < n) {
    y = y*x;
    k++;
}
```

4.12. Оператор цикла do-while

do
**   *оператор***
while (*выражение*);

Сначала выполняется оператор; если выражение истинно, то вычисляется значение выражения. Это повторяется до тех пор, пока выражение не станет ложным.

Если выражение ложно, то управление передается следующему оператору.

Замечание. Значение выражения определяется после выполнения оператора. Поэтому оператор выполняется хотя бы один раз.

Оператор do-while проверяет условие в конце цикла, в отличие от оператора while, в котором условие проверяется в начале цикла.

Пример:

```
x = 1;
do
    printf("%d\n", power(x,2));
while (++x <= 7);
```

4.13. Оператор цикла for

for (*выражение*1;
** *выражение*2;**
** *выражение*3)**
** *оператор***

*Выражение*1 описывает инициализацию цикла.

*Выражение*2 — проверка условия завершения цикла. Если оно истинно, то выполняется *оператор* тела цикла for, выполняется *выражение*3, все повторяется, пока *выражение*2 не станет ложным. Если оно ложно, цикл заканчивается и управление передается следующему оператору.

*Выражение*3 вычисляется после каждой итерации.

Оператор for эквивалентен следующей последовательности операторов:

```
выражение1;
while (выражение2) {
    оператор
    выражение3;
}
```

Пример:

```
for (x = 1; x <= 7; x++)
    printf("%d\n", power(x,2));
```

Любое из трех или все три выражения в операторе for могут отсутствовать, однако разделяющие их точки с запятыми (;) опускать нельзя.

Если опущено *выражение*2, то считается, что оно постоянно истинно. Оператор for (;;) представляет собой бесконечный цикл, эквивалентный оператору while(1).

Каждое из *выражений*1-3 может состоять из нескольких выражений, объединенных оператором запятая (,).

Пример:

```
for (i=0, j=n-1; i<n; i++, j--)
a[i] = a[j];
```

5. ФУНКЦИИ

5.1. Определение функции

Функция определяется описанием типа результата, формальных параметров и составного оператора (блока), описывающего выполняемые функцией действия.

Пример:

```
double linfunc (x, a, b)    тип результата, имя функции,
                                список параметров
double x;
double a;                   описание параметров
double b;
{
return (a*x + b);  возвращаемое значение    составной оператор
}
```

Оператор return может не возвращать никакого значения или возвращает значение выражения, стоящего в этом операторе.

Значение выражения при необходимости преобразуется к типу результата функции.

Функция, которая не возвращает значение, должна быть описана как имеющая тип void.

Пример:

```
void
errmesq(s)
char *s;
{
    printf("***%s\n",s);
}
```

5.2. Вызов функции

Существует два способа вызова функции:

имя функции (e1, e2, … , eN)

(**указатель на функцию*) (e1, e2, … , eN)

Здесь *указатель на функцию* — это переменная, содержащая адрес функции. Адрес функции может быть присвоен указателю оператором

указатель на функцию = имя функции;

Аргументы (фактические параметры) передаются по значению, т. е. каждое выражение e1, e2, … , eN вычисляется и значение передается функции, например, загрузкой в стек.

Порядок вычисления выражений и порядок загрузки значений в стек не гарантируются.

Во время выполнения не производится проверка числа или типа аргументов, переданных функции. Такую проверку можно произвести с помощью программы lint до компиляции.

Вызов функции — это выражение, значением которого является значение, возвращаемое функцией.

Описанный тип функции должен соответствовать типу возвращаемого значения. Например, если функция linfunc возвращает значение типа double, то эта функция должна быть описана до вызова:

```
extern double linfunc();
```

Замечание. Такое описание не определяет функцию, а только описывает тип возвращаемого значения; оно не нужно, если функция определена в том же файле до ее вызова.

Примеры:

Правильно:
```
extern double linfunc();
float y;
y = linfunc (3.05, 4.0, 1e-3);
```

Значение функции перед присваиванием переменной y преобразуется из типа double в тип float.

Неправильно:
```
float x;
float y;
x = 3.05;
y = linfunc (x, 4, 1e-3);
```

Тип аргументов не соответствует типу параметров, описанных в определении функции, а именно: константа 4 имеет тип int, а не double. В результате аргументы, загруженные в стек, имеют непра-

вильные тип и формат, поэтому значения, выбираемые из стека, бессмысленны и значение, возвращаемое функцией, не определено. Кроме того, если тип функции не описан, то считается, что возвращаемое значение имеет тип int. Поэтому, даже если функция linfunc возвращает правильное значение типа double, выражение, представляющее вызов функции, получит бессмысленное значение типа int (например, старшая половина значения double).

5.3. Функция main

Каждая программа начинает работу с функции main(). Во время выполнения программы можно представить аргументы через формальные параметры argc и argv функции main. Переменные среды языка оболочки shell передаются программе через параметр env.

Пример:

```
/*
программа печатает значения фактических параметров,
а затем переменных среды
*/
main (argc, argv, envp)
int argc;        /* число параметров */
char **argv;     /* вектор параметров-строк */
char **envp;     /* вектор переменных среды */
{
register int i;
register char **p;
                 /* печать значений параметров */
for (i = 0; i < argc; i++)
    printf("arg %i:%s\n", i, argv [i]);
                 /* печать значений переменных среды */
for (p = envp; *p != (char*)0; p++)
    printf("%s\n", *p);
}
```

Замечание. Параметры argv и envp могут быть описаны также следующим образом:

```
char *argv [];
char *envp [];
```

6. ОПИСАНИЯ

Описания используются для определения переменных и для объявления типов переменных и функций, определенных в другом месте. Описания также используются для определения новых типов данных на основе существующих типов. Описание не является оператором.

6.1. Основные типы

Примеры:

```
char c;
int x;
```

Основными типами являются:

char — символ (один байт);

int — целое (обычно одно слово);

unsigned — неотрицательное целое (такого же размера, как целое);

short — короткое целое (слово или полуслово);

long — длинное целое (слово или двойное слово);

float — число с плавающей точкой (одинарной точности);

double — число с плавающей точкой (двойной точности);

void — отсутствие значения (используется для нейтрализации значения, возвращаемого функцией).

Символы (char) в зависимости от компилятора могут быть со знаком или без знака. Рассматриваемые как целые, символы со знаком имеют значения от -127 до 128, а символы без знака — от 0 до 256. Некоторые реализации допускают явный тип unsigned char.

Данные целого типа int могут иметь такой же диапазон, как данные типа long или short.

Описание типа unsigned эквивалентно описанию типа unsigned int. Описание unsigned может сочетаться с описанием типа char, short или long, формируя описание типов unsigned char, unsigned short, unsigned long.

Описание типов short и long эквивалентны описаниям типов short int и long int. Диапазон данных типа long обычно в два раза больше диапазона данных типа short.

6.2. Указатели и массивы

Допустимо бесконечно большое число различных типов указателей и массивов.

Указатель на основной тип

Пример:

```
char *p;
```

Переменная p является указателем на символ, т. е. этой переменной должен присваиваться адрес символа.

Указатель на указатель

Пример:

```
char **t;
```

Переменная t — указатель на указатель символа.

Одномерный массив

Пример:

```
int a[50];
```

Переменная a — массив из 50 целых чисел.

Двумерный массив

Пример:

```
char m[7][50];
```

Переменная m — массив из семи массивов, каждый из которых состоит из 50 символов.

Массив из семи указателей

Пример:

```
char *r[7];
```

Массив r состоит из 7 указателей на символы.

Указатель на функцию

Пример:

```
int (*f)();
```

f — указатель на функцию, возвращающую целое значение.

6.3. Структуры

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

```
struct имя структуры {
Описание элементов
};
```

Пример:

```
struct dinner {
    char *place;
    float cost;
    struct dinner *next;
};
```

Структурная переменная описывается с помощью структурного типа.

Примеры:

```
struct dinner week_days [7]; /* массив структур */
struct dinner best_one;      /* одна структурная переменная */
sruct dinner *p      /* указатель на структурную переменную */
```

6.4. Поля бит в структурах

Поле бит — это элемент структуры, определенный как некоторое число бит, обычно меньше, чем число бит в целом числе. Поля бит предназначены для экономного размещения в памяти данных небольшого диапазона.

Пример:

```
struct bfeg {
    unsigned int bf_flg1 : 10;
    unsigned int bf_flg2 : 6;
};
```

Данная структура описывает 10-битовое поле, которое для вычислений преобразуется в значение типа unsigned int, и 6-битовое поле, которое обрабатывается как значение типа unsigned int.

6.5. Объединения

Объединение описывает переменную, которая может иметь любой тип из некоторого множества типов. Определение объеди-

ненного типа данных аналогично определению структурного типа данных:

union *имя объединения* {
    Описание элементов
};

Пример:

```
union bigword {
    long bg_long;
    char *bg_char [4];
};
```

Данные типа union bigword занимают память, необходимую для размещения наибольшего из своих элементов, и выравниваются в памяти к границе, удовлетворяющей ограничениям по адресации как для типа long, так и для типа char *[4].

Описание переменной объединенного типа

Пример:

```
union bigword x;
union bigword *p;
union bigword a [100];
```

6.6. Перечисления

Данные перечислимого типа относятся к некоторому ограниченному множеству данных.

Определение перечислимого типа данных

enum *имя перечислимого типа* {
    Список значений
};

Каждое значение данного перечислимого типа задается идентификатором.

Пример:

```
enum color {
    red, green, yellow
};
```

Описание переменной перечислимого типа

Пример:

```
enum color chair;
enum color suite [40];
```

Использование переменной перечислимого типа в выражении

Пример:

```
chair = red;
suite [5] != yellow;
```

6.7. Переименование типов

Формат:

typedef *старый тип новый тип*

Примеры:

```
typedef long large;
    /* определяется тип large, эквивалентный типу long */
typedef char *string;
    /* определяется тип string, эквивалентный типу char * */
```

Переименование типов используется для введения осмысленных или сокращенных имен типов, что повышает понятность программ, и для улучшения переносимости программ (имена одного типа данных могут различаться на разных ЭВМ).

6.8 Определение локальных переменных

Замечание 1. Постоянные переменные, сохраняемые в некоторой области памяти, инициализируются нулем, если явно не заданы начальные значения. Временные переменные, значения которых сохраняются в стеке или регистре, не получают начального значения, если оно не описано явно.

Замечание 2. Все описания в блоке должны предшествовать первому оператору.

Автоматические переменные

Пример:

```
{
int x; /* x - это автоматическая переменная */
}
```

Автоматическая переменная является временной, так как ее значение теряется при выходе из блока. Областью определения является блок, в котором эта переменная определена. Переменные, определенные в блоке, имеют приоритет перед переменными, определенными в охватывающих блоках.

Регистровые переменные

Пример:

```
{
register int y;
…
}
```

Регистровые переменные являются временными, их значения сохраняются в регистрах, если последние доступны. Доступ к регистровым переменным более быстрый. В регистрах можно сохранять любые переменные, если размер занимаемой ими памяти не превышает разрядности регистра. Если компилятор не может сохранить переменные в регистрах, он трактует их как автоматические. Областью действия является блок, в котором определена эта переменная. Операция получения адреса & не применима к регистровым переменным.

Формальные параметры

Примеры:

```
int func(x);
int x;
{
…
}

int func(x);
register int x;
{
…
}
```

Формальные параметры являются временными, так как получают значение фактических параметров, передаваемых функции. Областью действия является блок функции. Формальные парамет-

ры должны отличаться по именам от внешних переменных и локальных переменных, определенных внутри функции. В блоке функции формальным параметрам могут быть некоторые значения.

Статические переменные

Пример:

```
{
static int flag;
...
}
```

Статические переменные являются постоянными, так как их значения не теряются при выходе из функции. Любые переменные в блоке, кроме формальных параметров функции, могут быть определены как статические. Областью действия является блок, в котором эти переменные определены.

6.9. Определение глобальных переменных

Глобальные переменные

Пример:

```
int global_flag;
```

Глобальные переменные определяются на том же уровне, что и функции, т. е. они не локальны ни в каком блоке. Постоянные глобальные переменные инициализируются нулем, если явно не задано другое начальное значение. Областью действия является вся программа. Они должны быть описаны во всех файлах программы, в которых к ним есть обращения.

Замечание. Некоторые компиляторы требуют, чтобы глобальные переменные были определены только в одном файле и описаны как внешние в других файлах, где они используются. Глобальные переменные должны быть описаны в файле до первого использования.

Статические глобальные переменные

Пример:

```
 static int File_flag;
```

Постоянные глобальные переменные

Областью действия является файл, в котором данная переменная определена. Должны быть описаны до первого использования в файле.

6.10. Инициализация переменных

Любая переменная, кроме формальных параметров или автоматических массивов, структуры или объединения, при определении может быть инициализирована. Любая постоянная переменная инициализируется нулем (0), если явно не задано другое начальное значение. В качестве начального значения может использоваться любое константное выражение.

Основные типы

Примеры:

```
int I = 1;
float x = 3.145e-2;
```

Массивы

Примеры:

```
int a [] = {1,4,9,16,25,36};
char s [20] = {'a','b','c'};
```

Список значений элементов массива должен быть заключен в фигурные скобки. Если задан размер массива, то значения, не заданные явно, равны 0. Если размер массива опущен, то он определяется по числу начальных значений.

Строки

Пример:

```
char s [] = "hello";
```

Это описание эквивалентно описанию:

```
char s [] = {'h','e','l','l','o','\0'};
```

Структуры

Пример:

```
struct person {
    int height;
    char genfer;
};
```

```
struct person x = {70,'Y'};
struct person family [] = {
    {73,'X'},
    {68,'Y'},
    {50,'X'}
};
```

Список значений для каждой структурной переменной должен быть заключен в фигурные скобки, хотя, если число значений соответствует числу структуры, это не обязательно.

Значения присваиваются элементам структуры в порядке размещения элементов в определении структурного типа. Список значений может быть неполным, в этом случае неинициализированные элементы получают в качестве значения 0. Инициализация структуры в приведенном выше примере будет иметь вид:

```
struct person people [10] = {
    {68},
    {71},
    {74}
};
```

Элементам height первых трех структурных переменных массива присваиваются явные значения; остальные переменные получают значение 0.

6.11. Описание внешних объектов

Тип внешних объектов (т. е. переменных или функций), определенных в другой компоненте программы, должен быть явно описан. Отсутствие такого описания может привести к ошибкам при компиляции, компоновке или выполнении программы.

При описании внешнего объекта используется ключевое слово extern.

Примеры:

```
extern int gloal_var;
extern char *name;
extern int func ();
```

Можно опускать длину внешнего одномерного массива.

Пример:

```
extern float Num_array [];
```

Поскольку все функции определены на внешнем уровне, то для описании функции внутри блока прилагательное extern избыточно и часто опускается.

Пример:

```
{
…
int func ();
…
}
```

Функция, не возвращающая значения, должна описываться как имеющая тип void. Если тип функции явно не задан, считается, что она имеет тип int. Областью действия описания на внешнем уровне является остаток файла; внутри блока областью действия является данный блок. Обычно внешние описания располагаются в начале файла.

Некоторые компиляторы допускают описания переменных на внешнем уровне без прилагательного extern. Многократные описания внешних переменных компоновщик сводит к одному определению.

7. ПРЕПРОЦЕССОР

Если в качестве первого символа в строке программы используется символ #, то эта строка является командной строкой препроцессора (макропроцессора). Командная строка препроцессора заканчивается символом перевода на новую строку. Если непосредственно перед концом строки поставить символ обратной косой черты (\), то командная строка будет продолжена на следующую строку программы.

7.1. Замена идентификаторов

#define *идентификатор строка*

Пример:

```
#define ABC 100
```

Заменяет каждое вхождение идентификатора ABC в тексте программы на 100.

#undef *идентификатор*

Пример:

```
#undef ABC
```

Отменяет предыдущее определение для идентификатора ABC.

7.2. Макросы

Замечание. Во избежание ошибок при вычислении выражений параметры макроопределения необходимо заключать в скобки.

#define *идентификатор*1 (*идентификатор*2, ...) *строка*

Пример:

```
#define abs(A) (((A) > 0) ? (A) : -(A))
```

Каждое вхождение выражения abs(arg) в тексте программы заменяется на ((arg) > 0) ? (arg) : -(arg), причем параметр макроопределения A заменяется на arg.

Пример:

```
#define nmem (P,N)\
(P) -> p_mem[N].u_long
```

Символ \ продолжает макроопределение на вторую строчку. Это макроопределение уменьшает сложность выражения, описывающего массив объединений внутри структуры.

7.3. Включение файлов

Командная строка #include может встречаться в любом месте программы, но обычно все включения размещаются в начале файла исходного текста.

#include <*имя файла*>

Пример:

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

Процессор заменяет эту строку содержимым файла math.h. Угловые скобки означают, что файл math.h будет взят из некоторого стандартного каталога (обычно это /usr/include). Текущий каталог не просматривается.

#include "*имя файла*"

Пример:

#include "ABC"

Препроцессор заменяет эту строку содержимым файла ABC. Так как имя файла заключено в кавычки, то поиск производится в текущем каталоге (в котором содержится основной файл исходного текста). Если в текущем каталоге данного файла нет, то поиск производится в каталогах, определенных именем пути в опции –1 препроцессора. Если и там файла нет, то просматривается стандартный каталог.

7.4. Условная компиляция

Командные строки препроцессора используются для условной компиляции различных частей исходного текста в зависимости от внешних условий.

#if *константное выражение*

Пример:

#if ABC + 3

Истина, если константное выражение ABC + 3 не равно нулю.

#ifdef *идентификатор*

Пример:

#ifdef ABC

Истина, если идентификатор ABC определен ранее командой #define.

#ifndef *идентификатор*

Пример:

#ifndef ABC

Истина, если идентификатор ABC не определен в настоящий момент.

#else

...

#endif

Если предшествующие проверки # if, # ifdef или # ifndef дают значение **Истина**, то строки от #else до #endif игнорируются при компиляции.

Если эти проверки дают значение **Ложь**, то строчки от проверки до #else (а при отсутствии #else — до #endif) игнорируются.

Команда #endif обозначает конец условной компиляции.

Пример:

```
#ifdef DEBUG
    fprintf(stderr, "location: x = %d\n", x);
#endif
```

7.5. Номер строки и имя файла

#line *целая константа* "*имя файла*"

Пример:

```
#line 20 "ABC"
```

Препроцессор изменяет номер текущей строки и имя компилируемого файла. Имя файла может быть опущено.

8. СТРУКТУРА ПРОГРАММЫ

Структура программы предусматривает наличие следующих разделов:

1) включение используемых файлов;
2) описание типов данных;
3) описание функций.

Вход в программу осуществляется из функции main().

Пример:

```
#include <stdio.h>
#define A 100
#define B 200
    int T (void);
    int T (void)
        {
    return(A+B)
        }
void main()
        {
        T();
}
```

9. БИБЛИОТЕКА ВВОДА-ВЫВОДА

Программа, использующая перечисленные ниже функции ввода-вывода, должна включать в себя файл stdio.h с помощью команды препроцессора

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

Файл stdio.h содержит:
определение типа данных FILE;
определение параметров, используемых в макровызовах и вызовах библиотечных функций.
Примеры:
stdin — стандартный файл ввода;
stdout — стандартный файл вывода;
stderr — файл вывода сообщений об ошибках;
NULL — нулевой (0) указатель;
EOF — конец файла.
Замечание. По умолчанию файлы stdin, stdout и stderr связываются с терминалом.

Макроопределения:

putc()	ferror()
getc()	clearer()
putchar()	feof()
getchar()	fileno()

Замечание. Поток ввода-вывода идентифицируется указателем на переменную типа FILE. Средства буферизации включаются в поток как часть стандартного пакета ввода-вывода.

9.1. Доступ к файлам

fopen — открыть поток ввода-вывода.

Определение:
```
FILE *fopen (filename, type)
        char *filename, *type;
```

freopen — закрыть поток stream и открыть файл newfile, используя описание этого потока.

Определение:
```
FILE *freopen (newfile, type, stream)
```

```
                char *newfile, *type;
                FILE *stream;
```

fdopen — связать поток с дескриптором файла, открытым функцией open.

```
Определение:    FILE *fdopen (filfes, type)
                int fildes;
                char *type;
```

fclose — закрыть открытый поток ввода-вывода stream.

```
Определение:    int fclose (stream)
                FILE *stream;
```

fflush — записать символы из буфера в выходной поток stream.

```
Определение:    int fflush (stream)
                FILE *stream;
```

fseek — изменить текущую позицию offset в файле stream.

```
Определение:    int fseek (stream, offset, ptrname)
                FILE *stream;
                long offset;
                int ptrname;
```

rewind — переставить указатель текущего байта в потоке на начало файла.

```
Определение:    void rewind (stream)
                FILE *stream;
```

setbuf — модифицировать буфер потока.

```
Определение:    void setbuf (stream, buf)
                FILE *stream
                char *buf;
```

setvbuf — модифицировать буфер потока.

```
Определение:    int setvbuf (stream, buf, type, size)
                FILE *stream
                char *buf;
                int type, size;
```

9.2. Доступ к каналам

pclose — закрыть поток, открытый функцией popen.

```
Определение:    int pclose (stream)
                FILE *stream;
```

popen — создать поток как канал обмена между процессами.

Определение: FILE *popen (command, type)
char *command, *type;

9.3. Состояние файла

clearer — обнулить признаки ошибки потока.

Определение: void clearer (stream)
FILE *stream;

feof — проверить состояние конца файла в потоке.

Определение: int feof (stream)
FILE *stream;

ferrof — проверить состояние ошибки в потоке.

Определение: int ferror (stream)
FILE *stream;

fileno — связать дескриптор файла, открытого функцией open, с существующим потоком.

Определение: int fileno (stream)
FILE *stream;

9.4. Ввод-вывод строк

fgets — прочитать строку из выходного потока, включая символ новой строки.

Определение: char *fgets (s, n, stream)
char *s;
int n;
FILE *stream;

gets — прочитать строку из стандартного файла ввода stdin.

Определение: char *gets (s)
char *s;

fputs — записать строку в поток stream.

Определение: int fputs (s, stream)
char *s;
FILE *stream;

puts — записать строку в стандартный файл вывода stdout. В конце строк записывается символ новой строки.

Определение: int puts (s)
char *s;

9.5. Ввод символа

fgetc — прочитать следующий символ из выходного потока stream.

Определение: int fgetc (stream)
FILE *stream;

Замечание. Функции getc(), getchar(), ungetc() являются макроопределениям.

getc — прочитать следующий символ из входного потока.

Определение: int getc (stream)
FILE *stream;

getchar — прочитать следующий символ из стандартного файла ввода.

Определение: int getchar()

ungetc — вернуть символ во входной поток.

Определение: int ungetc (c, stream)
int c;
FILE *stream;

9.6. Вывод символа

fputc — записать символ в поток.

Определение: int fputc (c, stream)
int c;
FILE *stream;

Замечание. Функции putc() и putchar() являются макроопределениями.

putc — записать символ в поток.

Определение: int putc (c, stream)
int c;
FILE *stream;

putchar — записать символ в стандартный файл вывода.

Определение: int putchar (c);
int c;

9.7. Блочный ввод-вывод

fread — прочитать из входного потока определенное число байт (символов).

Определение: int fread (ptr, size, nitems, stream)
char *ptr;
int size, nitems;
FILE *stream;

fwrite — записать определенное число байт в выходной поток.

Определение: int fwrite (ptr, size, nitems, stream)
char *ptr;
int size, nitems;
FILE *stream;

10. ОБРАБОТКА СТРОК

Для выполнения описанных в этом разделе функций необходимо включить в программу файл string.h командой

#include <string.h>

strcat — сцепить две строки.

Определение: char *strcat(s1,s2)
char *s1, *s2;

strncat — сцепить две строки, причем из второй строки копировать не более n символов.

Определение: char *strncat(s1,s2,n)
char *s1, *s2;
intn;

strcmp — сравнить две строки в лексикографическом порядке.

Определение: int strcmp(s1,s2)
char *s1, *s2;

strncmp — сравнить первые n символов двух строк.

Определение: int strncmp(s1,s2,n)
char *s1, *s2;
intn;

strcpy — копировать строку s2 в строку s1.

Определение: char *strcpy(s1,s2)
char *s1, *s2;

strncpy — копировать не более n символов строки s2.

Определение: char *strncpy(s1,s2,n)
char *s1, *s2;
intn;

strlen — определить длину строки (число символов без завершающего нулевого символа).

Определение: int strlen(s)
char *s;

strchr — найти в строке первое вхождение символа c.

Определение: char *strchr(s,c)
char *s;
intn;

strrchr — найти в строке последнее вхождение символа c.

Определение: char *strrchr(s,c)
char *s;
int c;

strpbrk — найти в строке s1 любой из множества символов, входящих в строку s2.

Определение: char *strpbrk(s1,s2)
char *s1, *s2;

strspn — определить длину отрезка строки s1, содержащего символы из множества, входящих в строку s2.

Определение: int strspn(s1,s2)
char *s1, *s2;

strcspn — определить длину отрезка строки s1, содержащего символы строки s2.

Определение: int strcspn(s1,s2)
char *s1, *s2;

11. РАСПРЕДЕЛЕНИЕ ПАМЯТИ

malloc — выделение памяти размером size байт.

Определение: char *malloc(size)
unsigned size;

calloc — выделение памяти и обнуление ее.

Определение: char *calloc(nelem,elsize)
unsigned nelem, elsize;

realloc — изменение размера ранее выделенной памяти.

Определение: char *realloc(ptr,size)

```
                char *ptr;
                unsigned size;
```

free — освобождение ранее выделенной памяти.

```
Определение:  void free(ptr)
              char *ptr;
```

12. ФОРМАТИРОВАННЫЙ ВЫВОД

Для описания функций форматированного вывода printf, fprintf, sprintf используются следующие обозначения:

␣ Пробел (символ ␣ на самом деле не печатается!).

{} Используется только один из перечисленных элементов.

[] Используется только один или не используется ни одного из перечисленных элементов.

Для использования функций printf, fprintf, sprintf в программу необходимо вставить команду препроцессора

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

Функции printf, fprintf и sprintf имеют переменное число аргументов. Число и типы аргументов должны соответствовать спецификациям преобразования в форматной (управляющей) строке.

printf — записать аргумент в стандартный файл вывода stdout в соответствии с форматной строкой format.

```
Определение:  int printf (format [,arg] ...)
              char *format;
```

fprintf — записать аргументы в поток stream в соответствии с форматной строкой format.

```
Определение:  int fprintf (stream, format [,arg] ...)
              FILE *stream;
              char *format;
```

sprintf — записать аргументы в массив символов s в соответствии с форматной строкой.

```
Определение:  int sprintf (s, format [,arg])
              char *s, *format;
```

Пример:

printf ("error no.%d:%s", err, mesq);

Печатается значение переменной err как десятичное целое и значение mesq как строка. Результат форматированного вывода будет выглядеть следующим образом (с точностью до значения переменных):

error no. 13: cannot access file.

12.1. Спецификация преобразования

%[*выравнивание*] [*ширина* *] [*дополнительные признаки*] *символ преобразования*

Выравнивание вправо: по умолчанию.

Выравнивание влево: символ -.

Ширина определяет минимальное число выводимых символов. Она может задаваться целым числом; если значение соответствующей переменной превышает явно заданную ширину, то выводится столько символов, сколько необходимо. Символ * обозначает, что число выводимых символов будет определятся текущим значением переменной.

Пример:

printf("%*d", width, number);

12.2. Спецификация вывода символа

%[-] [*ширина*] c

Примеры:

%c A

%3c __A

%-3c A__

12.3. Спецификация вывода строки

%[-] [*ширина*] [*.точность*] s

Точность определяет число печатаемых символов. Если строка длиннее, чем заданная точность, то остаток строки отбрасывается.

Пример:

%10s abcdefghijklmn
%-10.5s abcde_____
%10.5s _____abcde

12.4. Спецификация вывода целого числа со знаком

%[-] [+] [␣] [*ширина*] [l] d

Для отрицательных чисел автоматически выводится знак – («минус»). Для положительных чисел знак + («плюс») выводится только в том случае, если задан признак +; если в спецификации задан знак ␣ («пробел»), то в позиции знака выводится пробел.

Символы преобразования:

l — необходим для данных типа long;

d — определяет вывод данных типа int в десятичном формате со знаком.

Примеры:

%d 43
%+d +43
% d 43

12.5. Спецификация вывода целого числа без знака

%[-] [#] [*ширина*] [l] { u, o, x, X}

Символ # определяет вывод начального нуля в восьмеричном формате или вывод начальных 0x или 0X в шестнадцатеричном формате. Символ l необходим для данных типа long.

Символы преобразования:

u — десятичное без знака;

o — восьмеричное без знака;

x — шестнадцатеричное без знака;

X — шестнадцатеричное без знака с прописными буквами A–F.

Примеры:

%u 777626577
%o 5626321721
%#o 05626321721
%x 2e59a3d1
%X 0X2E59A3D1

12.6. Спецификация вывода числа с плавающей точкой

%[-] { +, ␣ } [#] [*ширина*] [*.точность*] {f, e, E, g, G}

Для отрицательных чисел автоматически выводится знак – («минус»). Для положительных чисел выводится знак + («плюс»), если задан признак +; если в спецификации задан знак ␣ («пробел»), то в позиции знака выводится пробел. Завершающие нули не выводятся, если в спецификацию не включен признак #. Этот признак также обуславливает вывод десятичной точки даже при нулевой точности.

Точность определяет число цифр после десятичной точки для форматов f, e и E или число значащих цифр для форматов g и G. Округление делается отбрасыванием. По умолчанию принимается точность в шесть десятичных цифр.

Символы преобразования и формат вывода по умолчанию:

f [-] ddd.ddd (число с фиксированной точкой);

e [-] d.ddddde{-,+}dd (число в экспоненциальном формате);

E [-]d.ddddE{-,+}dd;

g — наиболее короткий формат из f или e;

G — наиболее короткий формат из f или E.

Типы аргументов float и double не различаются. Числа с плавающей точкой печатаются в десятичном формате.

Примеры:

%f 1234.567890

%1f 12345.6

%E 1.234568E+03

%.3e 1.234e+03

%g 1234.57

Замечание. Чтобы вывести символ %, необходимо в форматной строке задать два символа %%.

Пример:

printf ("%5.2f%%", 99.44);

В результате выполнения данной функции будет напечатано 99.44%.

13. ФОРМАТИРОВАННЫЙ ВВОД

Для описания функций форматированного ввода scanf, fscanf, sscanf используются следующие метаобозначения:

␣ Пробел (символ ␣ на самом деле не печатается!).

{} Используется только один из перечисленных элементов.

[] Используется только один или не используется ни одного из перечисленных элементов.

Для использования функций, описанных в этом разделе, в программу необходимо включить команду препроцессора

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

Функции scanf, fscanf, sscanf могут иметь переменное число аргументов. Число и типы аргументов должны соответствовать спецификациям преобразования в форматной строке.

scanf — ввести данные из стандартного файла ввода stdin в соответствии с форматной строкой format, присваивая значения переменным, заданным указателями pointer.

Определение:
```
int scanf (format [,pointer] ...)
char *format;
```

fscanf — ввести данные из потока stream в соответствии с форматной строкой format.

Определение:
```
int fscanf (stream, format [,pointer] ...)
FILE *stream;
char *format;
```

sscanf — читать данные из строки s в соответствии с форматной строкой format.

Определение:
```
int sscanf (s, format [,pointer] ...)
char *s, *format;
```

Примеры:

Входной поток содержит символы:

```
12.45  1048.73  AE405271  438
```

Вызов функции:

```
float x; char id[8+1]; int n;
scanf ("%f%*f%8 [A-Z0-9]%d", &x, id, &n);
```

Переменной x присваивается значение 12.45, символы 1048.73 пропускаются, переменной id присваивается строка символов «AE405271», переменной n — целое значение 438.

Входной поток содержит символы:
25 5432E-3 Monday
Вызов функции:
```
int I;  float x;  char name[50];
scanf ("%d%f%s", &I, &x, name);
```
Переменной I присваивается значение 25, переменной x — значение 5.432, переменной name — строка «Monday».

Входной поток содержит символ:
56 789 0123 56ABC
Вызов функции:
```
int I; float x; char name[50];
scanf ("%2d%f%*d,%[0-9]", &I, &x, name);
```
Переменной I присваивается значение 56, переменной x — значение 789.0, символы 0123 пропускаются, строка «56» присваивается переменной name. Последующий ввод символа из этого потока функцией getchar дает значение 'A'.

13.1. Спецификация преобразования

%[*] [*ширина*] [*дополнительные признаки*] *символ преобразования*

Символ * обозначает пропуск при вводе поля, определенного данной спецификацией; вводимое значение не присваивается никакой переменной. Ширина определяет максимальное число символов, вводимых по данной спецификации.

13.2. Пустые символы

Пробел или символ табуляции в форматной строке описывает один или более пустых символов. Пустые символы (пробел, символ табуляции, символ новой строки, перевода формата, вертикальной табуляции) во входном потоке в общем случае рассматриваются как разделители полей.

13.3. Литеральные символы

Литеральные символы в форматной строке, за исключением символов пробела, табуляции и символа %, требуют, чтобы во входном потоке появились точно такие же символы.

13.4. Спецификация ввода символа

%[*] [*ширина*] c

Ширина определяет число символов, которые должны быть прочитаны из входного потока и присвоены массиву символов. Если ширина опущена, то вводится один символ. По данной спецификации можно вводить пустые символы.

13.5. Спецификация ввода строки

%[*] [*ширина*] s

Ширина описывает максимальную длину вводимой строки. Строки во входном потоке должны разделятся пустыми символами; ведущие пустые символы игнорируются.

13.6. Спецификация ввода целого числа

%[*] [*ширина*] [l, h] { d, u, o, x }

Буква l определяет тип вводимых данных как long, буква h — как short. По умолчанию принимается тип int.

Символы преобразования:

d — десятичное целое со знаком;

u — десятичное целое без знака;

o — восьмеричное целое без знака;

x — шестнадцатеричное целое без знака.

13.7. Спецификация ввода числа с плавающей точкой

%[*] [*ширина*] [l] { f, e, g }

Буква l определяет тип вводимых данных как double, по умолчанию принимается тип float. Символы преобразования f, e, g являются синонимами.

13.8. Спецификация ввода по образцу

%[*] [*ширина*] *образец*

Образец (сканируемое множество) определяет множество символов, из которых может состоять вводимая строка, и задается строкой символов, заключенной в квадратные скобки.

Примеры:

[abcd]

[A321]

Непрерывный (в коде ASCII) диапазон символов образца описывается первым и последним символами диапазона.

Примеры:

[a-z]

[A-F0-9]

Если на первом месте в образце стоит символ ^, то вводиться будут все символы из входного потока, кроме перечисленных в образце, т. е. допустимое по этой спецификации множество символов будет дополнительным к описанному.

Пример:

[^0-9]

По спецификации преобразования, заданной образцом, вводится строка символов, включая завершающий ее нулевой символ. Пустые символы не пропускаются.

14. МОБИЛЬНОСТЬ ПРОГРАММ НА ЯЗЫКЕ Си

Мобильность программ — это свойство, позволяющее выполнять программы на разных ЭВМ, работающих под управлением разных версий ОС UNIX, с минимальным изменениями.

Изложенные в этом разделе рекомендации не являются строгим правилами. Не существует методики, гарантирующей автоматическое получение мобильной программы, но если вы будете использовать эти рекомендации при разработке своих программ, то ваши программы будут более мобильны, а также лучше организованы, более легки для понимания, изменения и поддержания.

14.1. Верификатор lint

Верификатор (программа семантического контроля) lint обеспечивает строгую проверку типов и выявляет многие конструкции, понижающие мобильность программ, написанных на языке Си. Если использовать верификатор lint на всех этапах разработки, то программный продукт будет гораздо легче переносить на любую версию ОС UNIX. Если вам потребуется переписать старую программу для повышения мобильности, то верификатор lint поможет выявить все сомнительные места.

14.2. Зависимость от компилятора

Некоторые детали в языке Си не стандартизированы. Это может проявиться в небольших различиях при обработке программы разными компиляторами. Какими бы малыми эти различия не были, они могут создать серьезные проблемы при переносе программ с одной вычислительной машины на другую. Поэтому не делайте никаких предположений о реализации свойств языка, которые строго не определены.

14.3. Мобильность файлов данных

Для переноса файлов, содержащих двоичные данные, используйте символьный ввод-вывод. Файлы двоичных данных по сути своей не мобильны, поскольку разные ЭВМ используют разное внутреннее представление данных. К сожалению, не существует простого пути для переноса файлов данных. Порядок байтов в файле может привести к серьезным проблемам при переносе данных с одной ЭВМ на другую по принципу «байт в байтов». Кроме того, коды символов могут быть разными на разных ЭВМ.

Один из способов решения этой проблемы заключается в разработке специальных программ преобразования для конкретных форматов данных. Другой подход заключается в записи байтов, составляющих объект данных, в некотором машинно-независимом порядке. Для передачи символьных данных используйте библиотечные функции printf и scanf, хотя это и непрактично.

ЛИТЕРАТУРА

Болски М. Справочник по Си. — М., 1990.
Джехани Н. С. Программирование на языке Си. — М., 1988.
Кернган Б., Ритчи Д. Язык программирования Си. — М., 1992.
Тондо У., Гемпел Л. Язык Си. — М., 1997.

ПРИЛОЖЕНИЕ

1. ПРОСТЫЕ ЧИСЛА И МАТРИЦЫ

Задача 1.1. Разминка

```
// Определить радиус R1 описанной окружности и радиус
                                                вписанной окружности
// R2 для правильного n-угольника со стороной a.
// Программа написана А. Г. Минаком
#include<stdio.h>
#include<math.h>
const float pi=3.14159;
float r1,r2,n,a;
main()
{
puts("Введите сторону a");
scanf("%f",&a);
puts("Введите количество углов n");
scanf("%f",&n);
r1=a/(2*sin(pi/n));
r2=a/(2*cos(pi/n));
printf("\n Радиус описанной окружности r1=%f",r1);
printf("\n Радиус вписанной окружности r2=%f",r2);
}
```

Задача 1.2. Поиск узлов из простых чисел

Программа отображает целые числа на плоскость некоторым регулярным образом и отмечает на рисунке места, где находятся простые числа.

```
//Построить матрицу A (15 X 15) таким образом: A (7,7)=1,
//затем, по спирали против часовой стрелки, увеличивая
//значение очередного элемента на единицу и выделяя все
//простые числа красным цветом, заполнить матрицу
#include <stdio.h>
#include <conio.h>
void main(void)
{
clrscr();
int mas[15][15];
int n=1,x=6,y=6,k=1;
int i,j;
while(1){
    mas[x][y]=k++;
    switch(n){
        case 1: x++;break;
        case 2: y--;break;
        case 3: x--;break;
        case 4: y++;break;
    }
    if(x==15) break;
    if(x==y && x<6) n=4;
    else if(x+y==12 && x<6) n=1;
    else if(x+y==12 && x>6) n=3;
    else if(x==y+1 && x>6) n=2;
}
for(i=0;i<15;i++){
    for(j=0;j<15;j++){
        textcolor(12);
        if(mas[j][i]>2)
        for(k=2;k<mas[j][i];k++)
        if(mas[j][i]%k==0) textcolor(15);
        cprintf("%3d",mas[j][i]);
    }
        printf("\n");
}
getch();
}
```

Задача 1.3. Определение счастливого билета

Номер билета вводится как положительное целое. Делением на 10 «вырубаем» очередные цифры: три цифры слева должны совпадать с тремя цифрами справа.

```
//Определение счастливого билета
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
void main(void)
{
clrscr();
unsigned long bil;
printf ("Введите номер билета: ");
scanf ("%ld",&bil);
int a,b,c,d,e,f;
a=bil/100000;
b=(bil/10000)%10;
c=(bil/1000)%10;
d=(bil/100)%10;
e=(bil/10)%10;
f=(bil%10);
if((a+b+c)==(d+e+f)) printf("\n счастливый");
else printf("\n несчастливый");
getch();
}
```

Задания для самостоятельного решения

1. Определить радиус и центр окружности, на которой лежит наибольшее число точек заданного на плоскости множества точек.

2. В множестве точек на плоскости найти пару точек с максимальным расстоянием между ними.

3. Задано множество прямых на плоскости (коэффициентами своих уравнений). Подсчитать количество точек пересечения этих прямых.

4. По заданной квадратной матрице размером 10 × 10 построить вектор длиной 19, элементы которого — максимумы элементов, диагоналей, параллельных главной диагонали.

5. Дана целочисленная квадратная матрица порядка n. Найти номера строк:

а) все элементы которых — нули;

б) элементы в каждой из которых одинаковы;

в) все элементы которых четны;

г) элементы каждой из которых образуют монотонную последовательность (монотонно убывающую или монотонно возрастающую);

д) элементы которых образуют симметричные последовательности (полиндромы).

6. Дана целочисленная квадратная матрица порядка 8. Найти наименьшее из значений элементов столбца, который обладает наибольшей суммой модулей элементов. Если таких столбцов несколько, то взять первый из них.

7. Даны натуральные числа i, j, действительная матрица размера 18 × 24 ($1 \le i < j \le 24$). Поменять в матрице местами i-й и j-й столбцы.

8. Дана действительная матрица размером $m \times n$. Определить числа $b_1, \ldots, b_m$, равные соответственно:

а) суммам элементов строк;

б) произведениям элементов строк;

в) наименьшим значениям элементов строк;

г) значениям средних арифметических элементов строк;

д) разностям наибольших и наименьших значений элементов строк.

9. Таблица футбольного чемпионата задана квадратной матрицей порядка n, в которой все элементы, принадлежащие главной диагонали, равны нулю, а каждый элемент, не принадлежащий главной диагонали, равен 2, 1 или 0 (числу очков, набранных в игре: 2 — выигрыш, 1 — ничья, 0 — проигрыш).

а) Найти число команд, имеющих больше побед, чем поражений.

б) Определить номера команд, прошедших чемпионат без поражений.

в) Выяснить, имеется ли хотя бы одна команда, выигравшая более половины игр.

10. Дано натуральное число n; найти $n!$. Использовать программу, включающую рекурсивную функцию вычисления $n!$.

11. Составить функцию вычисления значения целого числа по заданной строке символов, являющейся записью этого числа:

а) в десятичной системе счисления;

б) в шестнадцатеричной системе счисления (шестнадцатеричные цифры — это цифры от 0 до 9 и буквы от A до F).

2. ГРАФЫ (МАТРИЧНОЕ ПРЕДСТАВЛЕНИЕ)

Задача 2.1. Матрица инцидентности

Матрица $I(G)$ размером $n \times m$ (n — число вершин, m — число ребер/дуг графа G), (i, j)-й элемент которой равен 1, если вершина v_i инцидентна ребру e_j в неориентированном графе или если v_i есть начало дуги e_j, равен –1, если вершина v_i есть конец дуги e_j (только для орграфов), и равен 0 в остальных случаях.

В данной задаче задается граф и строится его матрица инцидентности.

```
//Построение матрицы инцидентности
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <iostream.h>
struct elem
{
int num; /* Номер вершины */
int suns; /* Количество сыновей */
char str[20]; /* Строка с номерами сыновей */
elem *next; /* Указатель на следующую вершину */
} *head, *w1, *w2;
```

```
int Connected(int i, int j)
{
int k;
char *str1;
w2 = head;
if(i == j) return 0;
for(k=1; k<i; k++)
     w2 = w2->next;
if( strchr(w2->str, j) ) return 1;
return 0;
}

void main()
{
int tops;
int i,j,k,l;
char *str1;
clrscr();
printf("Введите количество вершин \n");
scanf("%d", &tops);
head = (elem *)malloc(sizeof(elem));
head->num = 1;
head->suns = 0;
head->str[0] = '\0';
head->next = NULL;
w1 = head;
for(i=2;i<=tops;i++)
     {
     w2 = (elem *)malloc(sizeof(elem));
     w2->num = i;
     w2->suns = 0;
     w2->str[0] = '\0';
     w2->next = NULL;
     w1->next = w2;
     w1 = w2;
     }
w1 = head;
for(i=1; i<=tops; i++)
```

```
    {
//      clrscr();
    printf("Введите количество путей из вершины %d\n", i);
    scanf("%d", &k);
    for(j=1; j<=k; j++)
        {
        printf("Введите связь %d\n", j);
        scanf("%d", &l);
        if((l<=0) || (l > tops))
            {
            printf("Такой вершины нет, повторите попытку\n");
            l = 0;
            j--;
            continue;
            }
        w1->str[w1->suns++] = l;
        w1->str[w1->suns] = '\0';
        if(w1->suns == 49)
            {
            printf("Слишком много связей!");
            exit(1);
            }
        }
    w1 = w1->next;
    }
clrscr();
printf("\n\n Матрица инциндентности :\n");
for(i=1; i<=tops; i++)
    {
    printf("\n %d)", i);
    for(j=1; j<=tops; j++)
        {
        printf("%d ", Connected(i, j));
        }
    }
printf("\n\n Нажмите любую клавишу...");
getch();
}
```

Задача 2.2

Граф задается матрицей смежности. Графы можно задавать стеками, списками.

```
// Задание ориентированного графа матрицей смежности
// Реализовано А. Г. Минаком
#include <stdio.h>
#include <alloc.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#define BB 11000

// --- Функция вывода матрицы смежности орграфа(OUT_GRAF) ---
void OUT_GRAF (float ** GR,int r)
{
printf ("\n\n Матрица смежности \n");
for (int i=0;i<r;i++)
    {
    for (int j=0;j<r;j++)
        if (GR[i][j]>10000)
            printf (" - ");
        else
            printf ("%5.2f",GR[i][j]);
    puts(" ");
    }
}

// ------- Функция возвращения индекса первой вершины -------
// ------------------ смежной с вершиной v (FIRST) ------------------
int FIRST (float ** GR,int r,int v)
{
for (int i=0;i<r;i++)
    if ((GR[v][i]>0)&&(GR[v][i]<BB))
        return (i);
return (-1);
}
```

```
// ------ Функция возвращения индекса вершины, смежной с -----
// ---------- вершиной v следующей за индексом i (NEXT) --------
int NEXT (float ** GR,int r,int v,int i)
{
for (int j=i+1;j<r;j++)
    if ((GR[v][j]>0)&&(GR[v][j]<BB))
        return (j);
return (-1);
}

// ------- Функция возвращения вершины с индексом i из --------
// ---------- множества вершин, смежных с v (VERTEX) ----------
int VERTEX (float ** GR,int v,int i)
{
if ((GR[v][i]>0)&&(GR[v][i]<BB))
    return (i);
return (-1);
}

// --------- Функция вывода списка смежности орграфа с ----------
// --------- использованием FIRST, NEXT, VERTEX (OUT) ---------
void OUT (float ** GR,int r)
{
for (int i=0;i<r;i++)
    {
    int j=FIRST(GR,r,i);
    printf ("\n Вершина %i смежна с вершинами:" ,i);
    while (j!=-1)
        {
        int w=VERTEX(GR,i,j);
        if (w!=-1)
            printf ("%4i",w);
        j=NEXT(GR,r,i,j);
        }
    }
}
```

```
// ———————— ОСНОВНАЯ ПРОГРАММА ————————
main ()
{
int n=0,m=0,vs,v;
float cr;
clrscr();
int k=0;
printf ("\n Количество вершин графа n=");
scanf ("%i",&n);
float **G=(float **)malloc(n*sizeof(float *));
while(k!=7)
    {
    puts (" ");
    puts(" 1. Сформатировать орграф в виде матрицы
                                                смежности INSERT()");
    puts(" 2. Вывести на экран матрицу смежности
                                              орграфа OUT_GRAF(G,n)");
    puts(" 3. Возвратить индекс первой вершины, смежной
                                                с заданной FIRST(v)");
    puts(" 4. Возвратить индекс вершины, смежной с v,
                                    следующей за индексом i NEXT(v,i)");
    printf(" 5. Возвратить вершину с индексом i из
                          множества вершин, смежных с v VERTEX(v,i)");
    puts(" 6. Вывести список смежности орграфа
                        с помощью операторов FIRST, NEXT, VERTEX");
    puts(" 7. Выход из программы ");
    scanf("%i",&k);
    switch(k)
        {
        case 1:
            {
            if (m!=0)
                {
                printf ("\n Орграф уже сформирован");
                break;
                }
            m=1;
            for (int i=0;i<n;i++)
```

```
            for (int j=0;j<n;j++)
                G[i][j]=BB;
        for (i=0;i<n;i++)
            G[i][i]=0;
        for (i=0;i<n;i++)
            {
            printf ("\n Вершина: %i",i);
            printf ("\n Количество смежных вершин: ");
            scanf ("%i",&m);
            for (int j=0;j<m;j++)
                {
                printf ("\n %i — > ",i);
                scanf ("%i",&vs);
                printf ("стоимость: ");
                scanf ("%f",&cr);
                G[i][vs]=cr;
                G[vs][i]=cr;
                }
            }
        break;
        }
    case 2 :
        {
        if (m==0)
            {
            printf ("\ Орграф пуст.");
            break;
            }
        OUT_GRAF(G,n);
        break;
        }
    case 3:
        {
        if (m==0)
            {
            printf ("\n Орграф не задан. \n");
            break;
            }
```

```
        printf ("\n Вершина: ");
        scanf ("%i",&v);
        if ((v>n)||(v<0))
            {
            printf ("\n Вершина отсутствует в орграфе");
            break;
            }
        int fsv=FIRST(G,n,v);
        printf ("\n Первая смежная вершина: %i",fsv);
        break;
        }
    case 4:
        {
        if (m==0)
            {
            printf ("\n Орграф не задан. \n");
            break;
            }
        printf ("\n Вершина: ");
        scanf ("%i",&v);
        if ((v>n)||(v<0))
            {
            printf ("\n Вершина отсутствует в орграфе");
            break;
            }
        int i;
        printf ("\n Индекс i : ");
        scanf ("%i",&i);
        int nsv=NEXT(G,n,v,i);
        printf ("\n Индекс вершины, следующая за
                                                индексом i : %i',nsv);
        break;
        }
    case 5:
        {
        if (m==0)
            {
            printf ("\n Орграф не задан. \n");
```

```
                    break;
                    }
                printf ("\n Вершина: ");
                scanf ("%i",&v);
                if ((v>n)||(v<0))
                    {
                    printf ("\n Вершина отсутствует в орграфе");
                    break;
                    }
                int i;
                printf ("\n Индекс i : ");
                scanf ("%i",&i);
                int vg=VERTEX(G,v,i);
                printf ("\n Вершина под индексом i : %i",vg);
                break;
                }
            case 6 :
                {
                if (m==0)
                    {
                    printf ("\ Орграф пуст.");
                    break;
                    }
                OUT(G,n);
                break;
                }
        }
    }
free(G);
}
```

Задания для самостоятельного решения

1. Найти все циклы в графе, заданном матрицей смежности.
2. Найти периметр графа, заданного матрицей смежности.
3. Граф задан матрицей весов, которая есть сеть авиалиний. Найти самый дешевый путь от заданных вершин.

3. СТРУКТУРЫ ДАННЫХ

Решение комбинаторных задач предполагает выделение структур сложного типа с их последующей реализацией средствами выбранного языка программирования. При этом структура данных может не зависеть от конкретных языковых конструкций (абстрактная структура данных).

Под структурой данных понимают совокупность элементов фиксированных типов и набор базисных операций, определяющих организацию и способ обработки данных.

Стеки

Стеком называется структура данных, загрузка или увеличение элементов для которой осуществляется с помощью указателя стека в соответствии с правилом LIFO (Last In, First Out — последним введен, первым выведен).

Указатель стека sp (stack pointer) содержит в любой момент времени индекс (адрес) текущего элемента, который является единственным элементом стека, доступным в данный момент времени для работы со стеком (для случая, когда указатель стека всегда задает ячейку, находящуюся непосредственно над его верхним элементом).

1. Начальная установка:
   ```
   sp=1;
   ```
2. Загрузка элемента x в стек:
   ```
   stack[sp]=x;
   sp=sp+1;
   ```
3. Извлечение элемента из стека:
   ```
   sp=sp-1;
   x=stack[sp];
   ```

4. Проверка на переполнения и загрузка элемента в стек:

```
if(sp<=sd ) {stack[sp]=x;sp=sp+1}
else //переполнение
```

Здесь sd — размерность стека.

5. Проверка наличия элементов и извлечение элемента стека:

```
 if (sp>1){sp=sp-1;x=stack[sp]}
```

//антипереполнение

6. Чтение данных из указателя стека без извлечения элемента:

```
x=stack[sp-1]
```

Очереди

Очередь — одномерная структура данных, для которой загрузка или извлечение элементов осуществляется с помощью указателей начала (head) и конца (tail) очереди в соответствии с правилом FIFO (First In, First Out — первым введен, первым выведен).

1. Начальная установка:

```
Head=1; tail=1;
```

2. Добавление элемента:

```
queue[tail]=x; tail=tail+1
if(tail>qd) tail=1;
```

Здесь qd — размерность очереди.

3. Исключение элемента:

```
x=queue[head]; head=head+1;
if(head>qd) tail=1;
```

4. Проверка переполнения очереди и включение в нее элемента:

```
temp=tail+1;
if(temp=head)
{Переполнение}
else {queue[tail]=x; tail=temp}
```

5. Проверка наличия элементов и исключение элемента x:

```
if(head==tail) {очередь пуста}
else{ x=queue[head]; head=head+1;
    if(head>qd) head=1;}
```

Отметим, что при извлечении элемента из очереди все элементы могут перемещаться на один шаг к ее началу.

Связанные списки

Связанный список представляет собой структуру данных, состоящую из узлов (как правило, записей), содержащих указатели на следующий узел. Указатель, который ни на что не указывает,

снабжается значением NULL. Таким образом, в каждый элемент связанного списка добавляется указатель (звено связи).

Приведем основные базисные операции для работы с однонаправленным связанным списком.

1. Включение элемента q после элемента p:

```
link[q]=link[p];
link[p]=q;
```

Здесь q — индекс элемента, который должен быть вставлен в список после элемента с индексом p.

2. Исключение преемника элемента x:

```
If(link[x] != null) link[x]=[link[x]]
else
{элемент не имеет преемника}
```

Отметим, что элемент, следующий в списке за элементом x, называется преемником элемента x, а элемент, расположенный перед элементом x, называется предшественником элемента x. Если элемент x не имеет преемника, то содержащемся в нем указателю присваивается значение NULL.

3. Включение элемента y перед элементом x:

```
prev=0;
while((link[prev] != NULL) && (link[prev] != x) )do
prev=link[prev];
if (link[prev]=x) {link[prev]=y; link[y]=x }
else
{элемент x не найден};
```

Здесь link[0] является началом списка.

Отметим, что исключение последнего элемента из однонаправленного списка связано с просмотром всего списка.

В двунаправленном связанном списке каждый элемент имеет два указателя (succlink — описывает связь элемента с преемником, predlink — с предшественником). Ниже приведены основные базисные операции для работы с двунаправленным связанным списком.

1. Включение элемента y перед элементом x:

```
succlink[y]=x;
predlink[y]=predlink[x];
succlink[predlink[x]]=y;
predlink[x]=y;
```

2. Включение элемента y после элемента x:

```
succlink[y]=succlink[x];
predlink[y]=x;
predlink[succlink[x]]=y;
succlink[x]=y;
```

3. Исключение элемента x:

```
predlink[succlink[x]]=predlink[x];
succlink[predlink[x]]=succlink[x];
```

N-дольные графы

Структура данных — N-дольный граф может быть описана на мнемокоде следующим образом:

```
type вершина = record верш: char;
    шаг1: integer;
        шаг2: integer;
        ……
            шаг N-1: integer;
    первая =array[1..ML1] of вершина;
    вторая = array[1..ML2] of вершина;
    ……
    N-я =array[1..MLN] of вершина;
    Строка1,2=array[1..ML2] of char;
    Таблица1,2=array[1..ML1] of строка1,2;
    Строка1,3=array[1..ML3] of char;
    Таблица1,3=array[1..ML1] of строка1,3;
    ……
    строка2,3=array[1..ML3] of char;
    таблица2,3=array[1..ML2] of строка2,3;
    ……
    строка c2N-1,c2N=array[1..Mc2N] of char;
    таблица c2N-1,c2N=array[1..MLc2N-1] of строка c2N-1,c2N;
        var U1,2: таблица1,2;
        U1,3: таблица1,3;
        ……
        U2,3: таблица2,3;
        Uc2N-1,c2N: таблица c2N-1,c2N;
```

V1:первая;
V2: вторая;
.....
VN: N-я;

Задача 3.1. Все операции со стеком

Структура данных — стек. Принципы выборки и записи в стек описаны выше.

//Работа со стеком. Проверка: пуст ли стек. Добавить в стек.
Выбрать из стека.

//Стек полон

```
#include<stdio.h>
#include<dos.h>
#include<iostream.h>
#include<process.h>
#include<stdlib.h>
#include<conio.H>
#define max_size 200
char s[max_size]; //компоненты стека
int next=0; //позиция стека
int Empty()
{
    return next==0;
}
int Full()
{
    return next==max_size;
}
void Push()
{ if (next==max_size)
    {
    cout<<"Ошибка: стек полон"<<endl;}
    else { next++;cout<<"Добавлен"<<endl;
    cout<<"Что поместить в стек?"<<endl;
    cin>>s[next-1];
    }
}
```

```
void OUTst()
{    int i=0;
     if (next==0)
          {
          cout<<"Стек пуст"<<endl;
          }
     else
          {
          for(i=0;i<next;i++)
          cout<<s[i]<<" "<<endl;
          }
}
void Clear()
{
     next=0;
}
Poz()
{
     return next;
}
void Del()
{
     int a;
     if (next==0) cout<<"Ошибка: стек пуст"<<endl;
          else{next--;cout<<"Удален "<<endl;}
}
void menu()
{
     cout<<"0: распечатать стек"<<endl;
     cout<<"1: добавить в стек"<<endl;
     cout<<"2: удалить из стека"<<endl;
     cout<<"3: узнать номер позиции в стеке"<<endl;
     cout<<"4: узнать пуст ли стек"<<endl;
     cout<<"5: узнать полон ли стек"<<endl;
     cout<<"6: очистить стек"<<endl;
     cout<<"7: выход"<<endl;
}
```

```
main()
{
char c;
clrscr();
textcolor(15);
do {
    menu();
    cin>>c;
    clrscr();
    switch (c) {
        case '0':OUTst();getch();break;
        case '1':Push();break;
        case '2':Del();getch();break;
        case '3':cout<<"Номер "<<Poz()<<endl;getch();break;
        case '4':if (Empty()==1) cout<<"Пуст"<<endl;
                                 else cout<<"Не пуст"<<endl;getch();break;
        case '5':if (Full()==1)cout<<"Полн"<<endl;
                                else cout<<"Не полн"<<endl;getch();break;
        case '6':Clear();cout<<"Стек очистен"<<endl;getch();break;
        case '7':exit(1);
    }
    delay(200);
}
while (c!=7);
return 0;
}
```

Задача 3.2

```
//В список помещаются цифры 1…10
//Вводится число 11: рвется связь между 3 и 4, между ними
                                                вставляется число 11.
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <dos.h>
```

```
struct List
{    int i;
     List*next;
};
List*head=NULL;
void Hed(int i)
{
     if(head==NULL)
          {head=new List;
          head->i=1;
          head->next=NULL;
           }
     else{
          struct List*p,*p1;
          p=head;
          while(p->next!=NULL)
          p=p->next;
          p1=new List;
          p1->i=i;
          p1->next=NULL;
          p->next=p1;
          }
}
int s=0;
void Print(List*p)
{    printf("%d",p->i);
     if(p->next!=NULL)Print(p->next);
}
void delist()
{    List*p;
     while(head!=NULL)
          {
          p=head;
          head=head->next;
          delete(p);
          }
}
```

```
void Vstavka(int i1,int c)
{    List*p=head,*p1;
     while(p->i!=i1)
     p=p->next;
     p1=new List;
     p1->i=c;
     p1->next=p->next;
     p->next=p1;
}
void main()
{    clrscr();
     for(int i=1;i<=10;i++)
     Hed(i);
     Print(head);
     Vstavka(10,11);
     printf("\n");
     Print(head);
     Vstavka(3,11);
     printf("\n");
     Print(head);
     delist();
     getch();
}
```

Задача 3.3. Задача с абстрактной структурой данных — список

```
// Реализация списка с использованием статического массива
//Программа написана А. Г. Минаком
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>
#define max_lenght 100
#define LIST struct list
#define TRUE 1
#define FALSE 0
```

```
#define MUSIC struct music
FILE * f1;
MUSIC
    {
    int i_position;
    char name[20];
    char ispolnitel[20];
    float time;
    int kol;
    float priese;
    };
LIST
    {
    MUSIC elements[max_lenght];
    int last;
    } ;
// --- Функция определения последней позиции списка (END) ---
int END (LIST L)
{
return (L.last);
}
// -------- Функция определения - пуст ли список (EMPTY) ---------
int EMPTY (LIST L)
{
if (END(L)==0)
    return (TRUE);
else
    return (FALSE);
}
// --------- Функция вставки элемента списка (INSERT) ----------
LIST INSERT ( MUSIC x, LIST L )
{
L.last=END(L)+1;
L.elements[L.last]=x;
return(L);
}
```

```
// -------- Функция удаления элемента списка (DELETE) ----------
LIST DELETE ( LIST L)
{
int i_position;
if (EMPTY(L))
    {
    puts ("\n Список пуст.");
    return(L);
    }
else
    {
    L.last=END(L)-1;
    for (i_position=1; i_position<=END(L); i_position++)
        L.elements[i_position]=L.elements[i_position+1];
    return(L);
    }
}
// -- Функция возвращения значения списка по заданной
                                                позиции (RETRIEVE) --
char * RETRIEVE (int position, LIST L)
{
if (EMPTY(L))
    {
    puts ("\n Список пуст.");
    }
else if (position<0)
        {
        puts ("\n ОШИБКА. Позиция задана неверно");
        return (L.elements[0].name);
        }
    else if (position>END(L))
        {
        puts ("\n ОШИБКА. Позиция задана неверно");
        return (L.elements[END(L)].name);
        }
return(L.elements[position].name);
}
```

```
// --- Функция возвращения позиции элемента x списка(LOCATE) ---
int LOCATE (char * x, LIST L)
{
int i_position;
if (EMPTY(L))
     {
     puts ("\n Список пуст.");
     return(0);
     }
else
     {
     for (i_position=0; i_position<=END(L) ; i_position ++)
          if (strcmp(L.elements[i_position].ispolnitel,x)==0)
          return (i_position);
     puts ("\n Элемент не найден.");
     return (END(L)+1);
     }
}
// -------- Функция, делающая список пустым (MAKENULL) --------
LIST MAKENULL (LIST L)
{
int i_position;
for (i_position=0;i_position<=END(L); i_position ++)
     {
     strcpy(L.elements[i_position].ispolnitel,"");
     strcpy(L.elements[i_position].name,"");
     L.elements[i_position].time=0;
     L.elements[i_position].kol=0;
     L.elements[i_position].priese=0;
     }
L.last=0;
puts ("\n Список пуст.");
return (L);
}
// ---------------------- Функция вывода шапки ----------------------
void Head(void)
{
```

```
puts("—— T———— T————————T——T——T——¬");
puts("| Номер | Альбом | Исполнитель | Время | Кол-во | Цена |");
puts("+——+————-+————————+——+——+——+");
return;
}
// ----- Функция вывода элементов списка (OUTPUT_LIST) -----
void OUTPUT_LIST(LIST L)
{
int i_position=0;
int s_pos=1;
clrscr();
Head();
while(TRUE)
    {
    if (s_pos>END(L))
        {
        printf ("L——+———-+————+——+——+——\n");
        getch();
        return;
        }
    if(i_position>18)
        {
        i_position=0;
        printf ("L——+———+————+——+——+——\n");
        getch();
        clrscr();
        Head();
        }
    printf ("| %2i | %11s | %14s | % 2.2f | %4i | %3.2f |\n",s_pos,
                    L.elements[s_pos].name,L.elements[s_pos].ispolnitel,
                            L.elements[s_pos].time,L.elements[s_pos].kol,
                                                    L.elements[s_pos].priese);
    i_position++;
    s_pos++;
    }
}
```

```
// ——— Функция удаления пустого символа из конца строки
char * DEL_LAST(char * stroka)
{
int k_simb=strlen(stroka);
stroka[k_simb-1]=stroka[k_simb];
return (stroka);
}
// ———- Чтение из файла ——————————-
LIST INPUT_LIST (LIST L)
{
MUSIC now;
char a[80];
char filename[25];
printf ("\n Введите имя файла для чтения списка: ");
scanf ("%s",&filename);
f1=fopen(filename,"r");
fgets (a,80,f1);
while (!feof(f1))
    {
    fgets(a,80,f1);
    now.i_position=atoi(a);
    fgets(a,80,f1);
    if (feof(f1))
        {
        fclose(f1);
        return(L);
        }
    strcpy(now.name,DEL_LAST(a));
    fgets(a,80,f1);
    strcpy(now.ispolnitel,DEL_LAST(a));
    fgets(a,80,f1);
    now.time=atof(a);
    fgets(a,80,f1);
    now.kol=atoi(a);
    fgets(a,80,f1);
    now.priese=atof(a);
    L=INSERT(now,L);
    }
```

```
fclose (f1);
return(L);
}
// ———— Ввод в файл ————-
void FOUTPUT_LIST (LIST L)
{
char filename[25];
printf ("\n Введите имя файла для записи списка: ");
scanf ("%s",&filename);
f1=fopen(filename,"w");
for (int i_position=1;i_position<=END(L);i_position++)
    {
    fprintf (f1,"\n%i\n%s\n%s\n%f\n%i\n%f",i_position,
          L.elements[i_position].name,L.elements[i_position].ispolnitel,
                L.elements[i_position].time,L.elements[i_position].kol,
                                          L.elements[i_position].priese);
    }
fprintf (f1,"\n");
fclose (f1);
}
// —— Создать новый список ————
LIST NEW_LIST (LIST L)
{
MUSIC M;
textcolor(11);
clrscr();
Head();
for (int y=1;y<21;y++)
    puts ("|  |  |  |  |  |");
printf ("L—+——+———+——+——+——\n");
int n=0;
while (1)
    {
    if(n>18)
        {
        n--;
        gotoxy(0,21);
```

```
            puts ("|  |  |  |  |  |");
            gotoxy(0,22);
            puts ("L—+—+———+——+—+—\n");
            gotoxy(1,1);
            Head();
            }
        gotoxy(2,4+n);
        scanf("%i",&M.i_position);
        if (M.i_position==0)
            break;
        gotoxy (11,4+n);
        scanf("%s",&M.name);
        gotoxy (27,4+n);
        scanf("%s",&M.ispolnitel);
        gotoxy (48,4+n);
        scanf("%f",&M.time);
        gotoxy (57,4+n);
        scanf("%i",&M.kol);
        gotoxy (64,4+n);
        scanf("%f",&M.priese);
        L=INSERT(M,L);
        n++;
        }
return (L);
}
//————————Функция для подсчета суммы————————
void SUMM_LIST (LIST L)
{
float summ=0,vr=0,k=0;
for (int i_position=1;i_position<=END(L);i_position++)
        {
        summ+=L.elements[i_position].priese;
        k++;
        vr+=L.elements[i_position].time;
        }
printf("\n Всего %f элемента",k);
```

```
printf("\n Цена всех кассет = %f",summ);
printf("\n Общее время = %f",vr);
getch();
return;
}
//————————— Выдача с X по Y ————————————
void SXPOY(int x,int y,LIST L)
{
int i_position=0;
int s_pos=x;
clrscr();
Head();
while(TRUE)
    {
    if (s_pos>=y)
        {
        printf ("L——+———+————+——+—+———\n");
        getch();
        return;
        }
    if(i_position>18)
        {
        i_position=0;
        printf ("L——+———+————+——+—+———\n");
        getch();
        clrscr();
        Head();
        }
    printf ("| %2i | %11s | %14s | % 2.2f | %4i | %3.2f |\n",s_pos,
                L.elements[s_pos].name,L.elements[s_pos].ispolnitel,
                    L.elements[s_pos].time,L.elements[s_pos].kol,
                                        L.elements[s_pos].priese);
    i_position++;
    s_pos++;
    }
}
```

```
//————————— Поиск по альбому ——————————
void POALBOMY(char * alb,LIST L)
{
clrscr();
Head();
for (int i_position=1;i_position<=END(L);i_position++)
    if(strcmp(L.elements[i_position].name,alb)==0)
        printf ("| %2i | %11s | %14s | % 2.2f | %4i | %3.2f |\n",
                                    i_position,L.elements[i_position].name,
            L.elements[i_position].ispolnitel,L.elements[i_position].time,
                L.elements[i_position].kol,L.elements[i_position].priese);
printf ("L———+————+————————+———+———+———\n");
getch();
return;
}
//————————— Поиск по исполнителю ——————————
void POISPOLN(char * isp,LIST L)
{
clrscr();
Head();
for (int i_position=1;i_position<=END(L);i_position++)
    if(strcmp(L.elements[i_position].ispolnitel,isp)==0)
        printf ("| %2i | %11s | %14s | % 2.2f | %4i | %3.2f |\n",
                                    i_position,L.elements[i_position].name,
            L.elements[i_position].ispolnitel,L.elements[i_position].time,
                L.elements[i_position].kol,L.elements[i_position].priese);
printf ("L———+————+————————+———+———+———\n");
getch();
return;
}
//————————— Вывод элементов дешевле заданного ——————————
void MENPR(float pr,LIST L)
{
clrscr();
Head();
for (int i_position=1;i_position<=END(L);i_position++)
    if(L.elements[i_position].priese<=pr)
```

```
        printf ("| %2i | %11s | %14s | % 2.2f | %4i | %3.2f |\n",
                                    i_position,L.elements[i_position].name,
            L.elements[i_position].ispolnitel,L.elements[i_position].time,
                L.elements[i_position].kol,L.elements[i_position].priese);
printf ("L———+————————+————————+———+———+———\n");
getch();
return;
}
//——— Вывод элементов, время которых больше заданного—-
void BOTIM(float tim,LIST L)
{
clrscr();
Head();
for (int i_position=1;i_position<=END(L);i_position++)
    if(L.elements[i_position].time>=tim)
        printf ("| %2i | %11s | %14s | % 2.2f | %4i | %3.2f |\n",
                                    i_position,L.elements[i_position].name,
            L.elements[i_position].ispolnitel,L.elements[i_position].time,
                L.elements[i_position].kol,L.elements[i_position].priese);
printf ("L———+————————+————————+———+———+———\n");
getch();
return;
}
//———————— Функция поиска ———————————-
void POISK_LIST (LIST L)
{
float pr,tim;
int p=0,x,y;
char alb[25];
char isp[25];
while (p!=6)
    {
    clrscr();
    puts ("\n\n\n");
    puts ("\t\t———————————————————————¬");
    puts ("\t\t| 1. Выдать элементы с X по Y позиции |");
    puts ("\t\t| 2. Найти по альбому |");
    puts ("\t\t| 3. Найти по исполнителю |");
```

```
puts ("\t\t| 4. Выдать элементы дешевле заданного |");
puts ("\t\t| 5. Выдать элементы, время которых больше
                                                    заданного |");
puts ("\t\t| 6. Выход |");
puts ("\t\tL————————————————————————");
scanf ("%i",&p);
switch(p)
    {
    case 1:
        {
        printf("Введите X\n");
        scanf("%i",&x);
        printf("Введите Y\n");
        scanf("%i",&y);
        SXPOY(x,y,L);
        break;
        }
    case 2:
        {
        printf("Введите альбом\n");
        scanf("%s",&alb);
        POALBOMY(alb,L);
        break;
        }
    case 3:
        {
        printf("Введите исполнителя\n");
        scanf("%s",&isp);
        POISPOLN(isp,L);
        break;
        }
    case 4:
        {
        printf("Введите цену\n");
        scanf("%f",&pr);
        MENPR(pr,L);
        break;
        }
```

```
        case 5:
            {
            printf ("Введите время: ");
            scanf ("%f",&tim);
            BOTIM(tim,L);
            break;
            }
        }
    }
return;
}

LIST POVOZRAST (LIST L)
{
MUSIC tmp;
for (int i=1;i<=END(L);i++)
    for (int i_position=1;i_position<=END(L);i_position++)
        if(L.elements[i_position].priece<=
                                        L.elements[i_position+1].priese)
            {
            tmp=L.elements[i_position];
            L.elements[i_position]=L.elements[i_position+1];
            L.elements[i_position+1]=tmp;
            }
puts("\n Сортировка выполнена");
getch();
return(L);
}

LIST POUBOV (LIST L)
{
MUSIC tmp;
for (int i=1;i<=END(L);i++)
    for (int i_position=1;i_position<=END(L);i_position++)
        if(L.elements[i_position].time>=L.elements[i_position+1].time)
            {
            tmp=L.elements[i_position];
            L.elements[i_position]=L.elements[i_position+1];
```

```
                L.elements[i_position+1]=tmp;
                }
puts("\n Сортировка выполнена");
getch();
return(L);
}
// ———————— ФУНКЦИЯ ПОИСКА MAX ————————
void MAX_LIST(LIST L)
{
float max=0;
int k;
clrscr();
for (int i_position=1;i_position<=END(L);i_position++)
     if(L.elements[i_position].time>max)
          {
          max=L.elements[i_position].time;
          k=i_position;
          }
printf("Элемент с максимальным временем\n");
Head();
printf ("| %2i | %11s | %14s | % 2.2f | %4i | %3.2f |\n",k,
          L.elements[k].name,L.elements[k].ispolnitel,L.elements[k].time,
                                   L.elements[k].kol,L.elements[k].priese);
printf ("L——+————+————————+———+———+———\n");
getch();
return;
}
// ———————— ОСНОВНАЯ ПРОГРАММА ————————
void main (void)
{
LIST SPISOK;
SPISOK.last=0;
strcpy(SPISOK.elements[0].name,"");
strcpy(SPISOK.elements[0].ispolnitel,"");
SPISOK.elements[0].time=0;
SPISOK.elements[0].kol=0;
SPISOK.elements[0].priese=0;
```

```
char w[20];
int p=0;
textcolor(14);
textbackground(1);
clrscr();
int k=0;
while (k!=16)
    {
    clrscr();
    puts ("\n\n\n");
    puts ("\t\t———————————————————————¬");
    puts ("\t\t| 1. Добавить элемент в список |");
    puts ("\t\t| 2. Удалить первый элемент списка |");
    puts ("\t\t| 3. Определить последнюю позицию списка |");
    puts ("\t\t| 4. Определить, пуст ли список |");
    puts ("\t\t| 5. Выдать значение элемента по позиции списка |");
    puts ("\t\t| 6. Выдать позицию элемента по его значению
                                                    в списке |");
    puts ("\t\t| 7. Сделать список пустым |");
    puts ("\t\t| 8. Вывести элементы списка |");
    puts ("\t\t| 9. Считать список с файла |");
    puts ("\t\t|10. Записать список в файл |");
    puts ("\t\t|11. Подсчет суммы |");
    puts ("\t\t|12. Универсальный поиск |");
    puts ("\t\t|13. Сортировка цены по убыванию |");
    puts ("\t\t|14. Сортировка время по возрастанию |");
    puts ("\t\t|15. Поиск Max элемента |");
    puts ("\t\t|16. Выход из программы |");
    puts ("\t\tL———————————————————————");
    scanf ("%i",&k);
    switch(k)
        {
        case 1:
            {
            SPISOK=NEW_LIST(SPISOK);
            break;
            }
```

```
case 2:
    {
    SPISOK=DELETE(SPISOK);
    break;
    }
case 3:
    {
    printf ("\n Последняя позиция: %i\n", END(SPISOK));
    break;
    }
case 4:
    {
    if (EMPTY(SPISOK))
        printf (" Список пуст\n");
    else
        printf (" В списке есть элементы\n");
    break;
    }
case 5:
    {
    printf ("Введите позицию: ");
    scanf ("%i",p);
        printf ("Значение: %s",RETRIEVE(p,SPISOK));
    break;
    }
case 6:
    {
    printf ("\n Введите значение: ");
    scanf ("%s",&w);
    printf ("Позиция: %i",LOCATE(w,SPISOK));
    break;
    }
case 7:
    {
    SPISOK=MAKENULL(SPISOK);
    break;
    }
```

```
case 8:
    {
    OUTPUT_LIST(SPISOK);
    break;
    }
case 9:
    {
    SPISOK=INPUT_LIST(SPISOK);
    break;
    }
case 10:
    {
    FOUTPUT_LIST(SPISOK);
    break;
    }
case 11:
    {
    SUMM_LIST(SPISOK);
    break;
    }
case 12:
    {
    POISK_LIST(SPISOK);
    break;
    }
case 13:
    {
    SPISOK=POVOZRAST(SPISOK);
    break;
    }
case 14:
    {
    SPISOK=POUBOV(SPISOK);
    }
case 15:
    {
    MAX_LIST(SPISOK);
    }
```

```
        }
    }
clrscr();
char d;
gotoxy(10,10);
printf("Вы хотите сохранить список в файл:\n ");
scanf("%s",&d);
if((d=='y') || (d=='Y') || (d=='н') || (d=='Н'))
    FOUTPUT_LIST(SPISOK);
}
```

Данные

```
1
Вераокко
В. Леонтьев
0.930000
12
201.000000
2
Восьмое
Е.Осин
1.220000
14
234.000000
3
Герой
Машина_времени
1.330000
9
199.000000
4
Голос_травы
Л.Агутин
0.220000
6
178.000000
```

5
Дай_Бог
А.Малинин
1.440000
16
221.000000
6
Девочка
С.Крылов
1.330000
19
204.000000
7
День_ро-ния
И.Николаев
1.050000
17
215.000000
8
Доля
О.Газманов
1.230000
13
178.000000
9
Дороги
Любэ
1.440000
19
243.000000
10
Душа
Н.Ветлицкая
1.000000
10
194.000000
11

Ещё_не_всё
Л.Вайкуле
1.130000
12
154.000000
12
Жёлтые_боты
Браво
1.150000
13
179.000000
13
Звёзды_ждут
Мираж
1.120000
12
198.000000
14
Извозчик
А.Розенбаум
1.050000
9
145.000000
15
Играй
М.Распутина
1.140000
11
156.000000
16
Как_жаль
Т.Буланова
1.240000
15
214.000000
17
Клетки

Любэ
1.250000
16
235.000000
18
Красные
Кино
1.090000
8
175.000000
19
Летний_дождь
И.Тальков
1.220000
11
198.000000
20
Лодка
Н.Сенчукова
1.100000
14
199.000000
21
Льдинка
Л.Долина
1.030000
10
168.000000
22
Лейтенант
И.Аллегрова
1.050000
13
203.000000
23
Морозов
Т.Овсиенко

1.340000
18
212.000000
24
Мотылёк
С.Ротару
1.240000
17
211.000000
25
Скрипка
В.Меладзе
1.230000
15
190.000000
26
Никто
А.Свиридова
1.430000
16
178.000000
27
Одна
А.Варум
1.200000
14
179.000000
28
Ой_Лёха
А.Апина
1.150000
12
198.000000
29
Осень
А.Варум
1.240000

16
215.000000
30
Осень
Е.Осин
1.190000
15
213.000000
31
Офицеры
О.Газманов
1.310000
18
240.000000
32
Паромщик
А.Пугачёва
1.120000
14
196.000000
33
Петруха
А.Укупник
1.110000
13
189.000000
34
Плачет
Е.Осин
1.170000
15
198.000000
35
Поворот
Машина_времени
1.180000
16

205.000000
36
Помилуй
Любэ
1.230000
17
209.000000
37
Поцелуй
Ника
1.240000
18
195.000000
38
Привет
И.Аллегрова
1.110000
16
167.000000
39
Прости
Л.Долина
1.120000
17
178.000000
40
Прощай
Л.Лещенко
1.150000
15
189.000000
41
Пять_минут
Л.Гурченко
1.210000
17
201.000000

42
Россия
И.Тальков
1.130000
15
176.000000
43
Сама
И.Аллегрова
1.060000
16
187.000000
44
Самолёт
Валерия
1.090000
12
165.000000
45
Свеча
И.Аллегрова
1.210000
16
189.000000
46
Птица
Машина_времени
1.240000
17
196.000000
47
Тайга
Агата_Кристи
1.250000
18
197.000000

48
След
гр.Лицей
1.300000
16
215.000000
49
Снег
Машина_времени
1.170000
14
189.000000
50
Мальчик
Т.Буланова
1.250000
18
203.000000
51
Король
Машина_времени
1.140000
16
194.000000
52
Суженый
И.Аллегрова
1.250000
14
189.000000
53
Сэра
В.Меладзе
1.240000
15
197.000000

Задача 3.4. Пример на двусвязный список

Работа с двусвязным списком описана выше

```
// Реализация списка в виде двусвязного списка
// Реализовано А. Г. Минаком
# include <stdio.h>
# include <alloc.h>
# include <stdlib.h>
# include <conio.h>
# include <string.h>
# include <graphics.h>
# define DLLIST struct dllist
# define TRUE 1
# define FALSE 0
FILE *bazi;
int n=0;
FILE *f1;
DLLIST
    {
    int tag;
    DLLIST * previous;
    DLLIST * next;
    int nomer;
    char fam[25];
    char adres[25];
    int tel;
    int god;
    int grupa;
    };
DLLIST * LIST;
void lin (int y,int y1)
{
line(50,y,56,y);
line(56,y,56,y1);
line(56,y1,50,y1);
line(50,y1,50,y);
}
```

```
void grap()
{
int gdriver=DETECT,gmode,errorcode;
initgraph(&gdriver,&gmode,"");
errorcode=graphresult();
if (errorcode != grOk)
     {
     printf("Graphics error: %s\n", grapherrormsg(errorcode));
     printf("Press any key to halt");
     getch();
     exit(1);
     }
}
// -------
void menu()
{
textcolor(0);
clrscr();
cleardevice();
puts ("\n Создать новый список");
puts ("Удалить элемент списка по номеру мед. карты");
puts (" Определить, пуст ли список");
puts (" Выдать значение элемента по ключу");
puts (" Выдать ключ по значению элемента в списке");
puts (" Сделать список пустым");
puts (" Вывести элементы списка");
puts (" Сохранить файл");
puts (" Открыть файл");
puts (" Очистить базу данных");
puts (" Добавить новые записи");
puts (" Сортировка записи");
settextstyle(0,0,2);
textcolor(15);
outtextxy (30,300,"Для выхода из программы нажмитеESC");
}
```

```
void HEAD()
{
textcolor(15);
puts("\n ——T————————-T————————-T————————T———T———-¬");
puts("| Номер | Фамилия | Домашний | Год рождения | Телефон |
                                                    Группа- |");
puts("| мед. | Имя | адрес | | | крови |");
puts("| карты | Отчество | | | | |");
puts(" +——+————————-+————————-+————————+———+———-+");
}
// -------- Функция определения - пуст ли список (EMPTY) --------
int EMPTY(DLLIST *L)
{
DLLIST *cur=L;
if (!cur)
    return (TRUE);
else
    return (FALSE);
}
// --------- Функция вставки элемента списка (INSERT) ----------
void INSERT (DLLIST *t , DLLIST ** L)
{
DLLIST *cur=*L;
DLLIST *pr=0;
DLLIST *now;
now=(DLLIST*)malloc(sizeof(DLLIST));
*now=*t;
while (cur!=0)
    {
    pr=cur;
    cur=cur->next;
    }
//now->previous=cur;
//pr=cur;
//cur=cur->next;
now->next=cur;
if (pr==0)
    *L=now;
```

```
else
    {
    pr->next=now;
    cur->previous=now;
    now->previous=pr;
    now->next=cur;
    }
}
// ---- Функция удаления элемента списка по заданному ключу
                                                    (DELETE) ----
void DELETE (int tag_d, DLLIST ** L)
{
DLLIST *cur=*L;
DLLIST *pr=0;
while ((cur)&&(cur->nomer!=tag_d))
    {
    pr=cur;
    cur=cur->next;
    }
if (!cur)
    {
    printf ("\n Элемент списка не удается удалить\n");
    return;
    }
if (!pr)
    *L=cur->next;
else
    {
    pr->next=cur->next;
    cur->next->previous=pr;
    }
free(cur);
}
// ------ Функция поиска ключа по элементу списка (LOCATE) ------
int LOCATE (char * value, DLLIST * L)
{
DLLIST *cur=L;
```

```
if (!cur)
    {
    printf ("\n Список пуст \n");
    return(0);
    }
while ((cur)&&(strcmp(cur->fam,value)!=0))
    cur=cur->next;
if (!cur)
    {
    printf ("\n Элемент не найден \n");
    return(0);
    }
else
    return(cur->tag);
}
// ---- Функция поиска элемента списка по ключу (RETRIEVE) ----
char * RETRIEVE (int tag_s, DLLIST * L)
{
DLLIST *cur=L;
if (!cur)
    return (" Список пуст ");
while ((cur)&&(cur->tag!=tag_s))
    cur=cur->next;
if (!cur)
    return (" Элемент не найден ");
else
    return(cur->fam);
}
// ------ Функция, делающая список пустым (MAKENULL) ------
void MAKENULL (DLLIST **L)
{
DLLIST * cur=*L;
DLLIST * pr=0;
while (cur)
    {
    pr=cur;
    cur=cur->next;
```

```
    free(pr);
    }
*L=0;
}
// — Функция вывода элементов списка (OUTPUT_LIST) ——
void OUTPUT_LIST(DLLIST * L)
{
clrscr();
cleardevice();
HEAD();
DLLIST * cur=L;
int p=0;
int str=1;
if (EMPTY(L))
    //printf ("\n Список пуст");
    ;
else
    while (cur)
        {
        n++;
        if (p==20)
            {
            puts(" L—+———-+———+———+———+———");
            printf("\n Страница №%i",str);
            printf("\n Нажмите клавишу ESC или ENTER
                              для перехода на следующую страницу");
            str++;
            p=0;
            getch();
            cleardevice();
            closegraph();
            gotoxy(1,1);
            grap();
            HEAD();
            }
        p++;
        printf(" |%5i|%17s|%15s|%14i|%10i|%9i|",cur->nomer,
                    cur->fam, cur->adres,cur->god,cur->tel,cur->grupa);
```

```
            cur=cur->next;
            puts(" ");
            }
            puts(" L—+———————+———————+———————+———+———————");
            printf(" Страница №%i",str);
            if (str>1)
                {
                printf("\n Нажмите клавишу ESC или ENTER
                                    для перехода на следующую страницу");
                }
}
// -----
void podchet(DLLIST *L)
{
int st=0,plo=0,kolplo=0;
DLLIST *cur1=L;
while(cur1)
    {
    n++;
    kolplo++;
    plo+=cur1->nomer;
    st+=cur1->grupa;
    cur1=cur1->next;
    }
}
// ---------- Функция ввод в файл ----------
void FOUTPUT_LIST (DLLIST *L)
{
DLLIST *cur=L;
clrscr();
cleardevice();
int kluch=0;
char ima[30],a1[30];
printf("\n Введите имя файла\n");
scanf("%s",&ima);
printf("\n Правильно написано имя?(Y/N)");
```

```
char q=getch();
if (((q=='n')||(q=='N'))||((q=='т')||(q=='T')))
    return;
bazi=fopen("fbazi.txt","r");
while (!feof(bazi))
    {
    fgets(a1,30,bazi);
    if (strcmp(ima,a1)==0)
        kluch=1;
    }
fclose(bazi);
if (kluch==0)
    {
    bazi=fopen("fbazi.txt","a");
    fprintf(bazi,"\n%s",ima);
    }
fclose(bazi);
f1=fopen(ima,"w");
while (cur)
    {
    fprintf (f1,"\n%i\n%s\n%s\n%i\n%i\n%i",cur->nomer,
                    cur->fam,cur->adres,cur->god,cur->tel,cur->grupa);
    cur=cur->next;
    }
fclose (f1);
}
// —— Функция удаления предпоследнего символа ————————
char *dl(char *q)
{
int n=strlen(q);
q[n-1]=q[n];
return(q);
}
// —— Функция чтения из файла ————————————
void INPUT_LIST ()
{
MAKENULL(&LIST);
```

```
puts("");
DLLIST now;
char a[80];
char filename[25];
printf ("\n Введите имя файла для чтения списка: ");
scanf ("%s",&filename);
f1=fopen(filename,"r");
fgets(a,80,f1);
while (!feof(f1))
    {
    fgets(a,80,f1);
    now.nomer=atoi(a);
    fgets(a,80,f1);
    strcpy(now.fam,dl(a));
    fgets(a,80,f1);
    strcpy(now.adres,dl(a));
    fgets(a,80,f1);
    now.god=atoi(a);
    fgets(a,80,f1);
    now.tel=atoi(a);
    fgets(a,80,f1);
    now.grupa=atoi(a);
    INSERT(&now,&LIST);
    }
fclose (f1);
return;
}
//————————— Сортировка записи ————————————-
void sort(DLLIST *L)
{
DLLIST *cur=L;
//DLLIST *tmp=L;
int tmp;
char qw[80];
/*
tmp=(DLLIST *)malloc(sizeof(DLLIST));
*/
```

```
DLLIST *cur1=L;
podchet(LIST);
clrscr();
cleardevice();
for (int i=0;i<n;i++)
    {
    while (cur1)
        {
        cur1=cur1->next;
        if (cur->nomer>cur1->nomer)
            {
            tmp=cur->nomer;
            cur->nomer=cur1->nomer;
            cur1->nomer=tmp;
            tmp=cur->grupa;
            cur->grupa=cur1->grupa;
            cur1->grupa=tmp;
            strcpy (qw,cur->fam);
            strcpy(cur->fam,cur1->fam);
            strcpy(cur1->fam,qw);
            tmp=cur->tel;
            cur->tel=cur1->tel;
            cur1->tel=tmp;
            strcpy(qw,cur->adres);
            strcpy(cur->adres,cur1->adres);
            strcpy(cur1->adres,qw);
            tmp=cur->god;
            cur->god=cur1->god;
            cur1->god=tmp;
            }
        cur=cur->next;
        }
    cur=L;
    cur1=L;
    }
return;
}
```

```
// ———————— ОСНОВНАЯ ПРОГРАММА ————————
void main (void)
{
grap();
bazi=fopen("fbazi.txt","a");
fclose(bazi);
DLLIST a;
textbackground(1);
int kl=0;
LIST->next=NULL;
LIST->previous=NULL;
int gh=0;
int tag,tag_ins,tag_del,tag_serch;
char adr[25],value[25];
int ploch,etaj;
int kolkom,stoim;
int y=getmaxy()/29,y1=getmaxy()/29+getmaxy()/29,p=0;
char element[25];
cleardevice();
int k=1;
while (1)
    {
    menu();
    lin(y,y1);
    while (1)
        {
        if (kbhit())
            {
            cleardevice();
            menu();
            int ch=getch();
            switch(ch)
                {
                case 27:
                    exit(1);
                case 72:
                    if (k>1)
```

```
                    {
                    k--;
                    y-=getmaxy()/29;
                    y1-=getmaxy()/29;
                    lin(y,y1);
                    }
                else
                    lin(y,y1);
                break;
            case 80:
                if (k<12)
                    {
                    k++;
                    y+=getmaxy()/29;
                    y1+=getmaxy()/29;
                    lin(y,y1);
                    }
                else
                    lin(y,y1);
                break;
            case 13:
                //goto m1;
                kl=1;
                break;
            }
        }
    if (kl==1)
        {
        kl=0;
        break;
        }
    }
switch(k)
    {
    case 1:
        {
        p=0;
```

```
cleardevice();
closegraph();
gotoxy(1,1);
cprintf("Для создания нового списка нажмите
                                любую клавишу, иначе ESC");
int q=getch();
if (q!=27)
    MAKENULL(&LIST);
else
    //goto m3;
    {
    k=1;
    gh=1;
    }
if (gh==1)
    {
    grap();
    gh=0;
    break;
    }
HEAD();
for (int i=0;i<16;i++)
    puts(" | | | | | | |");
puts(" L–+———+———+———+———+———");
while (1)
    {
    gotoxy(1,1);
    cprintf(" ");
    if (p==16)
        {
        grap();
        clrscr();
        cleardevice();
        closegraph();
        puts(" ");
        printf("\n Нажмите клавишу ESC
                для перехода на следующую страницу");
        gotoxy(1,1);
```

```
                cprintf(" ");
                HEAD();
                for (int i=0;i<16;i++)
                    puts(" | | | | | | |");
                puts (" L—+——+——+——+–+—");
                p=0;
                }
            gotoxy (5,7+p);
            scanf ("%i",&a.nomer);
            gotoxy (12,7+p);
            scanf ("%s",&a.fam);
            gotoxy (29,7+p);
            scanf ("%s",&a.adres);
            gotoxy (45,7+p);
            scanf ("%i",&a.god);
            gotoxy (59,7+p);
            scanf ("%i",&a.tel);
            gotoxy (73,7+p);
            scanf ("%i",&a.grupa);
            p++;
            INSERT(&a,&LIST);
            gotoxy(1,1);
            cprintf("Нажмите ESC для выхода из программы");
            int h=getch();
            if (h==27)
                {
                grap();
                //goto m3;
                k=1;
                break;
                }
            }
        break;
        }
    case 2:
        {
        printf ("\n Введите номер мед. карты: ");
```

```
        scanf ("%i",&tag_del);
        DELETE(tag_del,&LIST);
        break;
        }
    case 3:
        {
        int nil=EMPTY(LIST);
        if (nil)
            printf ("\n Список пуст \n");
        else
        printf ("\n В списке имеются элементы \n");
        getch();
        break;
        }
    case 4:
        {
        printf ("\n Ключ: ");
        scanf ("%i",&tag_serch);
        char * serch=RETRIEVE(tag_serch,LIST);
        printf ("\n %s \n",serch);
        getch();
        break;
        }
    case 5:
        {
        printf ("\n Значение: ");
        scanf ("%s",&value);
        int tag_sr=LOCATE(value,LIST);
        printf ("\n Ключ :%i \n",tag_sr);
        getch();
        break;
        }
    case 6:
        {
        MAKENULL(&LIST);
        break;
        }
```

```
case 7:
    {
    OUTPUT_LIST(LIST);
    getch();
    break;
    }
case 8:
    {
    FOUTPUT_LIST(LIST);
    break;
    }
case 9:
    {
    clrscr();
    cleardevice();
    char a2[80];
    bazi=fopen("fbazi.txt","r");
    while (!feof(bazi))
        {
        fgets(a2,80,bazi);
        printf("%s",a2);
        }
    fclose(bazi);
    getch();
    INPUT_LIST();
    break;
    }
case 10:
    {
    clrscr();
    cleardevice();
    bazi=fopen("fbazi.txt","w");
    fclose(bazi);
    bazi=fopen("fbazi.txt","a");
    fprintf(bazi," ");
    fclose(bazi);
    printf("\n База обновлена");
```

```
        getch();
        break;
        }
    case 11:
        {
        p=0;
        cleardevice();
        closegraph();
        HEAD();
        for (int i=0;i<16;i++)
            puts(" | | | | | | |");
        puts(" L–+———+———+———+———+———");
        while (1)
            {
            gotoxy(1,1);
            cprintf(" ");
            if (p==16)
                {
                grap();
                clrscr();
                cleardevice();
                closegraph();
                puts(" ");
                printf("\n Нажмите клавишу ESC
                        для перехода на следующую страницу");
                gotoxy(1,1);
                cprintf(" ");
                HEAD();
                for (int i=0;i<16;i++)
                    puts(" | | | | | | |");
                puts(" L–+—+———+———+—+———");
                p=0;
                }
            gotoxy (5,7+p);
            scanf ("%i",&a.nomer);
            gotoxy (12,7+p);
            scanf ("%s",&a.fam);
```

```
                gotoxy (29,7+p);
                scanf ("%s",&a.adres);
                gotoxy (45,7+p);
                scanf ("%i",&a.god);
                gotoxy (59,7+p);
                scanf ("%i",&a.tel);
                gotoxy (73,7+p);
                scanf ("%i",&a.grupa);
                p++;
                INSERT(&a,&LIST);
                gotoxy(1,1);
                cprintf("Нажмите ESC для выхода из программы");
                int h=getch();
                if (h==27)
                    {
                    grap();
                 //goto m3;
                    k=11;
                    break;
                    }
                }
            }
        case 12:
            {
            sort(LIST);
            break;
            }
        }
    }
closegraph();
}
```

Задания для самостоятельного решения

1. Найти все вершины заданного графа, недостижимые от заданной его вершины.

2. Найти самый длинный простой путь в графе.

3. Известно, что заданный граф — не дерево. Проверить, можно ли удалить из него вершину (вместе с инцидентными ей ребрами), чтобы в результате получилось дерево.

4. Из заданного графа удалить все вершины, от которых недостижима заданная.

4. ФАЙЛЫ

Задача 4.1. Работа с файлами

```
//Вывод файла на экран
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#define EOF -1
void main()
{
char *s;
int i=0;
char ch = 0,chn = 0;
clrscr();
/*Степанов Владимир 904 a1 */
FILE *filE;
if((filE = fopen("C:\\User\\NINA\\Andr\\filers.cpp","rt"))== NULL)
    {
    cout << "Cannot open file" << "\n";
    }
else cout << "File opened" << "\n" << '\n';
fseek(filE, 0, SEEK_SET);
// char st[256];
do
    {
    ch = fgetc(filE);
    cout << ch;
    // if((ch!=EOF)&&(ch=='\n'))i++;
     }while(ch!=EOF);
```

```
fclose(filE);
// cout << "\n" << "\n" <<"Count of string " << (i+1);
getche();
}
```

Задания для самостоятельного решения

1. Дан символьный файл *F*. Получить копию файла в файле *G*.

2. Сведения об автомобиле состоят из его марки, номера и фамилии владельца. Дан файл *F*, содержащий сведения о нескольких автомобилях. Найти:

а) фамилии владельцев и номера автомобилей данной марки;

б) количество автомобилей каждой марки.

3. Дан файл *F*, содержащий различные даты. Каждая дата — это число, месяц и год. Найти:

а) год с наименьшим номером;

б) все весенние даты;

в) самую позднюю дату.

5. ЭЛЕМЕНТЫ КОМПЬЮТЕРНОЙ ГРАФИКИ

Задача 5.1. Графика в двумерном пространстве

Все типы преобразования двумерного пространства.

```
//Вращение, сжатие, растяжение, треугольника

#include <stdlib.h>
#include <conio.h>
#include <iostream.h>
#include <graphics.h>
#include <process.h>
#include <math.h>
#include <stdio.h>
#include <dos.h>
#define pi 3.1415926535897932385
float theta=0.0;
float thinc=6.2831;
```

```
float Tx,Ty,Sx,Sy;
float R3x[3],aa[3][3],p[3][3],p1[3][3];
float Px[10],Py[10],PLx[3];
int R,x,y,t ;
int tekx,teky;
void PLOT( int x1,int y1,int N);
void PLOSK(int n);
void AROT(float theta);
void arot(float theta);
void Graph(float x[100],float y[100]);
void X(float a[3][3],float x[3]);
void Vector(int n,int m);
void Init(void);
void AROTG(void);
void A_ROTS(void);
void V_ector(int n);
void V_ecto_r(int n);
void OSI(void);

void OSI(void)
{
setcolor(3);
PLOT(320,240,-3);
PLOT(0,220, 3); outtextxy(330,10,"y");
PLOT(0,-220,2); outtextxy(330,440,"-y");
PLOT(-220,0, 3); outtextxy(40,240,"-x");
PLOT(220,0,2); outtextxy(600,240,"x");
setcolor(14);
}
void V_ecto_r(int n)
{
int i,l;
A_ROTS();
for(i=0;i<n;i++)
    {
    PLx[0]=Px[i];PLx[1]=Py[i];PLx[2]=1;
    X(aa ,PLx);
```

```
    Px[i]=R3x[0];Py[i]=R3x[1];
    }
}
void A_ROTS(void)
{ //матрица сжатия
int i,j;
for(i=0;i<3;i++)
for(j=0;j<3;j++)
aa[i][j]=0.0;
aa[0][0]=Sx;
aa[1][1]=Sy;
aa[2][2]=1.0;
}
void V_ector(int n)
{int i,l;
AROTG();
for(i=0;i<n;i++)
    {
    PLx[0]=Px[i];PLx[1]=Py[i];PLx[2]=1;
    X(aa ,PLx);
    Px[i]=R3x[0];Py[i]=R3x[1];
    }
}
void AROTG(void)
{ //матрица переноса
int i,j;
for(i=0;i<3;i++)
for(j=0;j<3;j++)
    {
    aa[i][j]=0.0;
    if(i==j) aa[i][j]=1.0;
    }
aa[0][2]=-Tx;
aa[1][2]=-Ty;
aa[2][2]=1.0;
}
```

```
void Init(void)
{
Px[0]=100;Py[0]=0;
Px[1]=0;Py[1]=100;
Px[2]=-100;Py[2]=0;
}
void Vector(int n,int m)
{int i,l;
if(m==0)
AROT(theta);
else arot(theta);
for(i=0;i<n;i++)
    {
    PLx[0]=Px[i];PLx[1]=Py[i];PLx[2]=1;
    X(aa,PLx);
    Px[i]=R3x[0];Py[i]=R3x[1];
    }
}
void X(float a[3][3],float x[3])
{
float ab;int i,j;//матрица поворота 3x3 умножается на вектор
for (i=0;i<3;i++)
    {
    ab=0.0;
    for (j=0;j<3;j++)
        {
        ab=ab+a[i][j]*x[j];
        }
    R3x[i]=ab;
    }
}
void AROT(float theta)
{ //матрица поворота возле x
int i,j;
float c,s;
for(i=0;i<3;i++)
for(j=0;j<3;j++)
```

```
aa[i][j]=0.0;
c=cos(theta); //возле x
s=sin(theta);
aa[0][0]=c;
aa[1][1]=c;
aa[0][1]=s;
aa[1][0]=-s;
aa[2][2]=1.0;
}
void arot(float theta)
{ //матрица поворота возле y
int i,j;
float c,s;
for(i=0;i<3;i++)
for(j=0;j<3;j++)
aa[i][j]=0.0;
c=cos(theta);
s=sin(theta);
aa[0][0]=c;
aa[0][2]=-s;
aa[1][1]=1.0;aa[0][2]=s;
aa[2][2]=c;
aa[2][2]=1.0;
}
void PLOSK(int n)
{ //плоская фигура
int i;
//setcolor(n%15);
PLOT(320,240,-3);
PLOT(Px[0],Py[0],3);
for(i=0;i<n;i++)
PLOT(Px[i],Py[i],2);
settextstyle(0,0,2);
PLOT(Px[0],Py[0],2);
}
void PLOT( int x1,int y1,int N)
{
int xt,yt;
```

```
switch(N)
    {
    case -3: tekx=320;teky=240; break;
    case 3: x=320+x1;y=240-y1;
        tekx=x; teky=y; break;
    case 2: xt=320+x1; yt=240-y1;
        line(tekx,teky,xt,yt); tekx=xt;teky=yt; break;
    default: setbkcolor(4); getch(); break;
    }
}

main()
{
int gdriver = DETECT, gmode, errorcode;
int midx, midy,i,j;
/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, " ");
/* read result of initialization */
errorcode = graphresult();
if (errorcode != gr0k) /* an error occurred */
    {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1); /* terminate with an error code */
    }
OSI();
Init();
PLOSK(3);
theta=0.0;
thinc=0.01;
for(i=0;i<70;i++)
    {
    theta=theta+thinc;
    Vector(3,0);
    delay(100);
    cleardevice();
```

```
    OSI();
    PLOSK(3);
    }
theta=0.0;
thinc=0.01;
for(i=0;i<50;i++)
    {
    theta=theta+thinc;
    Vector(3,1);
    delay(100);
    cleardevice();
    OSI();
    PLOSK(3);
    }
Tx=0.0;
Ty=0.0;
for(i=0;i<50;i++)
    {
    Tx=Tx+0.1;Ty=Ty-0.1;
    V_ector(3);
    delay(100);
    cleardevice();
    OSI();
    PLOSK(3);
    } // Init();
Sx=1;
Sy=1;
for(i=0;i<70;i++)
    {
    Sx=Sx-0.002;Sy=Sy-0.002;
    V_ecto_r(3);
    delay(100);
    cleardevice();
    OSI();
    PLOSK(3);
    }
}
```

Задача 5.2

Растяжение и сжатие трехмерного пространства, вращение трехмерного объекта, удаление невидимых линий.

```
//Рисуется октаэдр, его можно вращать, используя клавиши 1..9
#include <IOSTREAM.H>
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <complex.h>
#include <math.h>
#include <dos.h>
float map=50;
float x[9],y[9],z[9];
float xc[9],yc[9];
int Gd,Gm,i;
float v,psi,fi,a11,a12,a13,a21,a22,a23,a31,a32,a33;
int launch[9][4];
float vertical,horizontal;
char ch;
float cc,butco,erco;
long msx,msy;
int lb,rb,tb;
float moverx,movery;
float gx(float a,float b,float c);
float gy(float a,float b,float c);
void normalize(void);
void launcher(void);
void subcounter(int x,int y,int z,float &x1,float &y1,float &z1,
                                                    float fi,float psi);
float normal(int a,int b,int c,float x[9],float y[9],float z[9],float &v);

float gx(float a,float b,float c)
{
float x;
x=320+a*100+c*sqrt(0.5)*100;
```

```
return x;
}
float gy(float a,float b,float c)
{
float y;
y=250+b*100+c*sqrt(0.5)*100;
return y;
}
void normalize(void)
{
for(i=1;i<9;i++)
     {
     xc[i]=gx(x[i],y[i],z[i]);
     yc[i]=gy(x[i],y[i],z[i]);
     }
}
void launcher(void)
{
moverx=0.05;
movery=0.05;
launch[1][1]=1; launch[1][2]=1; launch[1][3]=1;
launch[2][1]=1; launch[2][2]=0-1; launch[2][3]=1;
launch[3][1]=1; launch[3][2]=0-1; launch[3][3]=0-1;
launch[4][1]=1; launch[4][2]=1; launch[4][3]=0-1;
launch[5][1]=0-1; launch[5][2]=1; launch[5][3]=1;
launch[6][1]=0-1; launch[6][2]=0-1; launch[6][3]=1;
launch[7][1]=0-1; launch[7][2]=0-1; launch[7][3]=0-1;
launch[8][1]=0-1; launch[8][2]=1; launch[8][3]=0-1;
}
void subcounter(int x,int y,int z,float &x1,float &y1,float &z1,
                                                    float fi,float psi)
{
a11=cos(psi); a12=sin(fi)*sin(psi);
a13=0; a21=0; a22=cos(fi); a23=0;
a31=sin(psi); a32=0-cos(psi)*sin(fi);
a33=0;
x1=a11*x+a21*y+a31*z;
```

```
y1=a12*x+a22*y+a32*z;
z1=a13*x+a23*y+a33*z;
}
float normal(int a,int b,int c,float x[9],float y[9],float z[9],float &v)
{
float a1,a2,b1,b2,c1,c2,n,m;
a1=x[a]-x[b]; b1=y[a]-y[b]; c1=z[a]-z[b];
a2=x[a]-x[c]; b2=y[a]-y[c]; c2=z[a]-z[c];
n=b1*c2-c1*b2; m=c1*a2-a1*c2;
v=a1*b2-a2*b1;
if ((x[a]*n+y[a]*m+z[a]*v)<0){v=0-v;}
return v;
}
void drawcub(void)
{
int cnt;
float v;
for(cnt=1;cnt<9;cnt++){
subcounter(launch[cnt][1],launch[cnt][2],launch[cnt][3],x[cnt],
                                                    y[cnt],z[cnt],fi,psi);}
normalize();
v=normal(8,5,4,x,y,z,v);
if (v<=0)
    {
    line(xc[5],yc[5],xc[8],yc[8]);
    line(xc[8],yc[8],xc[4],yc[4]);
    line(xc[1],yc[1],xc[4],yc[4]);
    line(xc[1],yc[1],xc[5],yc[5]);
    };
v=normal(2,1,3,x,y,z,v);
if (v>=0)
    {
    line(xc[1],yc[1],xc[2],yc[2]);
    line(xc[2],yc[2],xc[3],yc[3]);
    line(xc[3],yc[3],xc[4],yc[4]);
    line(xc[1],yc[1],xc[4],yc[4]);
    }
v=normal(1,2,5,x,y,z,v);
```

```
if (v>=0)
    {
    line(xc[1],yc[1],xc[5],yc[5]);
    line(xc[5],yc[5],xc[6],yc[6]);
    line(xc[6],yc[6],xc[2],yc[2]);
    line(xc[1],yc[1],xc[2],yc[2]);
    }
v=normal(4,3,8,x,y,z,v);
if (v<=0)
    {
    line(xc[3],yc[3],xc[4],yc[4]);
    line(xc[8],yc[8],xc[4],yc[4]);
    line(xc[8],yc[8],xc[7],yc[7]);
    line(xc[7],yc[7],xc[3],yc[3]);
    }
v=normal(8,5,7,x,y,z,v);
if (v>=0)
    {
    line(xc[5],yc[5],xc[8],yc[8]);
    line(xc[8],yc[8],xc[7],yc[7]);
    line(xc[7],yc[7],xc[6],yc[6]);
    line(xc[6],yc[6],xc[5],yc[5]);
    }
v=normal(3,2,7,x,y,z,v);
if (v<=0)
    {
    line(xc[2],yc[2],xc[3],yc[3]);
    line(xc[3],yc[3],xc[7],yc[7]);
    line(xc[7],yc[7],xc[6],yc[6]);
    line(xc[6],yc[6],xc[2],yc[2]);
    }
}
void main(void)
{
int x,y;
int gdriver = DETECT, gmode, errorcode;
int midx, midy;
initgraph(&gdriver, &gmode, "");
```

```
horizontal=1; vertical=1;
launcher();movery=0;moverx=0;
for(;;)
    {
    if (ch=='\x0'){ ch=getch();}
    if (ch=='\x1B'){ break; }
    if (ch=='8'){movery = -0.05; moverx=0;}
    if (ch=='4'){moverx = 0.05; movery=0;}
    if (ch=='2'){movery = 0.05; moverx=0;}
    if (ch=='6'){moverx = -0.05; movery=0;}
    if (ch=='7'){movery = -0.05; moverx=-0.05;}
    if (ch=='9'){moverx = 0.05; movery=0.05;}
    vertical=vertical+movery;
    horizontal=horizontal+moverx;
    psi = horizontal;
    fi = vertical;
    setcolor(14);
    drawcub();
    ch=getch();
    setcolor(0);
    drawcub();
    }
getch();
closegraph();
}
```

Задача 5.3

```
//Синхронное вращение трех октаэдров клавишами 1..9
#include <IOSTREAM.H>
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <complex.h>
#include <math.h>
#include <dos.h>
```

```
float map=50;
float x[9],y[9],z[9];
float xc[9],yc[9];
int Gd,Gm,i;
float v,psi,fi,a11,a12,a13,a21,a22,a23,a31,a32,a33;
int launch[9][4];
float vertical,horizontal;
char ch;
float cc,butco,erco;
long msx,msy;
int lb,rb,tb;
float moverx,movery;
float gx(float a,float b,float c);
float gy(float a,float b,float c);
void normalize(void);
void launcher(void);
void subcounter(int x,int y,int z,float &x1,float &y1,float &z1,
                                                    float fi,float psi);
float normal(int a,int b,int c,float x[9],float y[9],float z[9],float &v);

float gx(float a,float b,float c)
{
float x;
x=320+a*50+c*sqrt(0.5)*50+b;
return x;
}
float gy(float a,float b,float c)
{
float y;
y=250+b*50+c*sqrt(0.5)*50+a;
return y;
}
void normalize(void)
{
for(i=1;i<9;i++)
    {
    xc[i]=gx(x[i],y[i],z[i]);
```

```
    yc[i]=gy(x[i],y[i],z[i]);
    }
}
void launcher(void)
{
moverx=0.05;
movery=0.05;
launch[1][1]=1; launch[1][2]=1; launch[1][3]=1;
launch[2][1]=1; launch[2][2]=0-1; launch[2][3]=1;
launch[3][1]=1; launch[3][2]=0-1; launch[3][3]=0-1;
launch[4][1]=1; launch[4][2]=1; launch[4][3]=0-1;
launch[5][1]=0-1; launch[5][2]=1; launch[5][3]=1;
launch[6][1]=0-1; launch[6][2]=0-1; launch[6][3]=1;
launch[7][1]=0-1; launch[7][2]=0-1; launch[7][3]=0-1;
launch[8][1]=0-1; launch[8][2]=1; launch[8][3]=0-1;
}
void subcounter(int x,int y,int z,float &x1,float &y1,float &z1,
                                                    float fi,float psi)
{
a11=cos(psi); a12=sin(fi)*sin(psi);
a13=0; a21=0; a22=cos(fi); a23=0;
a31=sin(psi); a32=0-cos(psi)*sin(fi);a33=0;
x1=a11*x+a21*y+a31*z;
y1=a12*x+a22*y+a32*z;
z1=a13*x+a23*y+a33*z;
}
float normal(int a,int b,int c,float x[9],float y[9],float z[9],float &v)
{
float a1,a2,b1,b2,c1,c2,n,m;
a1=x[a]-x[b]; b1=y[a]-y[b]; c1=z[a]-z[b];
a2=x[a]-x[c]; b2=y[a]-y[c]; c2=z[a]-z[c];
n=b1*c2-c1*b2; m=c1*a2-a1*c2;
v=a1*b2-a2*b1;
if ((x[a]*n+y[a]*m+z[a]*v)<0){v=0-v;}
return v;
}
```

```
void drawcub(int L,int L1,int L2,int L3,int L4,int L5,int L6)
{
int cnt;
float v;
int poly[8],pol[8];
for(cnt=1;cnt<9;cnt++){
    subcounter(launch[cnt][1],launch[cnt][2],launch[cnt][3],x[cnt],
                                                    y[cnt],z[cnt],fi,psi);}
normalize();
v=normal(8,5,4,x,y,z,v);
if (v<=0)
    {
    line(xc[5],yc[5],xc[8],yc[8]);
    line(xc[8],yc[8],xc[4],yc[4]);
    line(xc[1],yc[1],xc[4],yc[4]);
    line(xc[1],yc[1],xc[5],yc[5]);
    poly[0] = xc[5];
    poly[1] =yc[5];
    poly[2] = xc[8];
    poly[3] = yc[8];
    poly[4] = xc[4];
    poly[5] = yc[4];
    poly[6] = xc[1];
    poly[7] = yc[1];
        setfillstyle(3, L);
        fillpoly(4, poly);
    poly[0] = xc[5]+25;
    poly[1] =yc[5]+25;
    circle(xc[5]+25,yc[5]+25,15);circle(xc[5]+25,yc[5]+25,5);
    circle(xc[5]+25,yc[5]+25,3);
    poly[2] = xc[8]+25;
    poly[3] = yc[8]+25;
    circle(xc[8]+25,yc[8]+25,15);circle(xc[8]+25,yc[8]+25,5);
    circle(xc[8]+25,yc[8]+25,3);
    poly[4] = xc[4]+25;
    poly[5] = yc[4]+25;
    circle(xc[4]+25,yc[4]+25,15);circle(xc[4]+25,yc[4]+25,5);
    circle(xc[4]+25,yc[4]+25,3);
```

```
    poly[6] = xc[1]+25;
    poly[7] = yc[1]+25;
    circle(xc[1]+25,yc[1]+25,15);circle(xc[1]+25,yc[1]+25,5);
    circle(xc[1]+25,yc[1]+25,3);
        setfillstyle(1, L);
        fillpoly(4, poly);
    pol[0] = xc[5]+150;
    pol[1] =yc[5]+150;
    pol[2] = xc[8]+150;
    pol[3] = yc[8]+150;
    pol[4] = xc[4]+150;
    pol[5] = yc[4]+150;
    pol[6] = xc[1]+150;
    pol[7] = yc[1]+150;
        setfillstyle(1, L);
        fillpoly(4, pol);
    pol[0] = xc[5]-150;
    pol[1] =yc[5]-150;
    pol[2] = xc[8]-150;
    pol[3] = yc[8]-150;
    pol[4] = xc[4]-150;
    pol[5] = yc[4]-150;
    pol[6] = xc[1]-150;
    pol[7] = yc[1]-150;
        setfillstyle(1, L);
        fillpoly(4, pol);
    };
v=normal(2,1,3,x,y,z,v);
if (v>=0)
    {
    line(xc[1],yc[1],xc[2],yc[2]);
    line(xc[2],yc[2],xc[3],yc[3]);
    line(xc[3],yc[3],xc[4],yc[4]);
    line(xc[1],yc[1],xc[4],yc[4]);
    poly[0] = xc[1];
    poly[1] =yc[1];
    poly[2] = xc[2];
    poly[3] = yc[2];
```

```
    poly[4] = xc[3];
    poly[5] = yc[3];
    poly[6] = xc[4];
    poly[7] = yc[4];
        setfillstyle(5,L1);
        fillpoly(4, poly);
    pol[0] = xc[1]+150;
    pol[1] =yc[1]+150;
    pol[2] = xc[2]+150;
    pol[3] = yc[2]+150;
    pol[4] = xc[3]+150;
    pol[5] = yc[3]+150;
    pol[6] = xc[4]+150;
    pol[7] = yc[4]+150;
        setfillstyle(1,L1);
        fillpoly(4, pol);
    pol[0] = xc[1]-150;
    pol[1] =yc[1]-150;
    pol[2] = xc[2]-150;
    pol[3] = yc[2]-150;
    pol[4] = xc[3]-150;
    pol[5] = yc[3]-150;
    pol[6] = xc[4]-150;
    pol[7] = yc[4]-150;
        setfillstyle(1,L1);
        fillpoly(4, pol);
    }
v=normal(1,2,5,x,y,z,v);
if (v>=0)
    {
    line(xc[1],yc[1],xc[5],yc[5]);
    line(xc[5],yc[5],xc[6],yc[6]);
    line(xc[6],yc[6],xc[2],yc[2]);
    line(xc[1],yc[1],xc[2],yc[2]);
    poly[0] = xc[1];
    poly[1] =yc[1];
    poly[2] = xc[5];
    poly[3] = yc[5];
```

```
    poly[4] = xc[6];
    poly[5] = yc[6];
    poly[6] = xc[2];
    poly[7] = yc[2];
        setfillstyle(2, L2);
        fillpoly(4, poly);
    pol[0] = xc[1]+150;
    pol[1] =yc[1]+150;
    pol[2] = xc[5]+150;
    pol[3] = yc[5]+150;
    pol[4] = xc[6]+150;
    pol[5] = yc[6]+150;
    pol[6] = xc[2]+150;
    pol[7] = yc[2]+150;
        setfillstyle(1, L2);
        fillpoly(4, pol);
    pol[0] = xc[1]-150;
    pol[1] =yc[1]-150;
    pol[2] = xc[5]-150;
    pol[3] = yc[5]-150;
    pol[4] = xc[6]-150;
    pol[5] = yc[6]-150;
    pol[6] = xc[2]-150;
    pol[7] = yc[2]-150;
        setfillstyle(1, L2);
        fillpoly(4, pol);
    }
v=normal(4,3,8,x,y,z,v);
if (v<=0)
    {
    line(xc[3],yc[3],xc[4],yc[4]);
    line(xc[8],yc[8],xc[4],yc[4]);
    line(xc[8],yc[8],xc[7],yc[7]);
    line(xc[7],yc[7],xc[3],yc[3]);
    poly[0] = xc[3];
    poly[1] =yc[3];
    poly[2] = xc[4];
```

```
    poly[3] = yc[4];
    poly[4] = xc[8];
    poly[5] = yc[8];
    poly[6] = xc[7];
    poly[7] = yc[7];
        setfillstyle(4, L3);
        fillpoly(4, poly);
    pol[0] = xc[3]+150;
    pol[1] =yc[3]+150;
    pol[2] = xc[4]+150;
    pol[3] = yc[4]+150;
    pol[4] = xc[8]+150;
    pol[5] = yc[8]+150;
    pol[6] = xc[7]+150;
    pol[7] = yc[7]+150;
        setfillstyle(1, L3);
        fillpoly(4, pol);
    pol[0] = xc[3]-150;
    pol[1] =yc[3]-150;
    pol[2] = xc[4]-150;
    pol[3] = yc[4]-150;
    pol[4] = xc[8]-150;
    pol[5] = yc[8]-150;
    pol[6] = xc[7]-150;
    pol[7] = yc[7]-150;
        setfillstyle(1, L3);
        fillpoly(4, pol);
    }
v=normal(8,5,7,x,y,z,v);
if (v>=0)
    {
    line(xc[5],yc[5],xc[8],yc[8]);
    line(xc[8],yc[8],xc[7],yc[7]);
    line(xc[7],yc[7],xc[6],yc[6]);
    line(xc[6],yc[6],xc[5],yc[5]);
    poly[0] = xc[5];
    poly[1] =yc[5];
```

```
    poly[2] = xc[8];
    poly[3] = yc[8];
    poly[4] = xc[7];
    poly[5] = yc[7];
    poly[6] = xc[6];
    poly[7] = yc[6];
        setfillstyle(5, L4);
        fillpoly(4, poly);
    pol[0] = xc[5]+150;
    pol[1] =yc[5]+150;
    pol[2] = xc[8]+150;
    pol[3] = yc[8]+150;
    pol[4] = xc[7]+150;
    pol[5] = yc[7]+150;
    pol[6] = xc[6]+150;
    pol[7] = yc[6]+150;
        setfillstyle(1, L4);
        fillpoly(4, pol);
    pol[0] = xc[5]-150;
    pol[1] =yc[5]-150;
    pol[2] = xc[8]-150;
    pol[3] = yc[8]-150;
    pol[4] = xc[7]-150;
    pol[5] = yc[7]-150;
    pol[6] = xc[6]-150;
    pol[7] = yc[6]-150;
        setfillstyle(1, L4);
        fillpoly(4, pol);
    }
v=normal(3,2,7,x,y,z,v);
if (v<=0)
    {
    line(xc[2],yc[2],xc[3],yc[3]);
    line(xc[3],yc[3],xc[7],yc[7]);
    line(xc[7],yc[7],xc[6],yc[6]);
    line(xc[6],yc[6],xc[2],yc[2]);
    poly[0] = xc[2];
    poly[1] =yc[2];
```

```
        poly[2] = xc[3];
        poly[3] = yc[3];
        poly[4] = xc[7];
        poly[5] = yc[7];
        poly[6] = xc[6];
        poly[7] = yc[6];
            setfillstyle(6, L5);
            fillpoly(4,poly);
        pol[0] = xc[2]+150;
        pol[1] =yc[2]+150;
        pol[2] = xc[3]+150;
        pol[3] = yc[3]+150;
        pol[4] = xc[7]+150;
        pol[5] = yc[7]+150;
        pol[6] = xc[6]+150;
        pol[7] = yc[6]+150;
            setfillstyle(1, L5);
            fillpoly(4, pol);
        pol[0] = xc[2]-150;
        pol[1] =yc[2]-150;
        pol[2] = xc[3]-150;
        pol[3] = yc[3]-150;
        pol[4] = xc[7]-150;
        pol[5] = yc[7]-150;
        pol[6] = xc[6]-150;
        pol[7] = yc[6]-150;
            setfillstyle(1, L5);
            fillpoly(4, pol);
        }
}
void main(void)
{
int x,y;
int gdriver = DETECT, gmode, errorcode;
int midx, midy;
initgraph(&gdriver, &gmode, " ");
horizontal=1; vertical=1;
```

```
launcher();movery=0;moverx=0;
setbkcolor(8);
for(;;)
    {
    if (ch=='\x0'){ ch=getch();}
    if (ch=='\x1B'){ break; }
    if (ch=='8'){movery = -0.05; moverx=0;}
    if (ch=='4'){moverx = 0.05; movery=0;}
    if (ch=='2'){movery = 0.05; moverx=0;}
    if (ch=='6'){moverx = -0.05; movery=0;}
    if (ch=='7'){movery = -0.05; moverx=-0.05;}
    if (ch=='9'){moverx = 0.05; movery=0.05;}
    vertical=vertical+movery;
    horizontal=horizontal+moverx;
    psi = horizontal;
    fi = vertical;
    setcolor(14);
    drawcub(1,2,3,4,5,12,7);
    ch=getch();
    setcolor(8);
    drawcub(8,8,8,8,8,8,8);
    }
getch();
closegraph();
}
```

Задания для самостоятельного решения

1. Написать программу, вычерчивающую тетраэдр.
2. Написать программу, вычерчивающую кубооктаэдр.
3. Написать программу, вычерчивающую икосаэдр.
4. Написать общий алгоритм удаления невидимых линий для ортогональной проекции платоновых тел и реализовать программу.
5. Написать алгоритм нанесения текстуры (характеристик поверхностей) на грани платоновых тел.
6. Написать и реализовать алгоритм преобразования трехмерного пространства, в которое помещены платоновы тела.

ЛИТЕРАТУРА

Болски М. И. Язык программирования Си: Справочник / Пер. с англ. — М.: Радио и связь, 1988.

Anderson B. Type Syntax in Language C: an object lesson in syntactic innovation. — SIGPLAN Notices, 1980. V. 15, N. 3 (March).

AT&T UNIX (Release 5.0). UNIX System User's Manual (Release 5.0). — AT&T Bell Laboratories, Murray Hill. 1982.

Bourne S. R. The UNIX System. — Addison-Wesley Publishing Co. 1982.

Stroustrup B. Data Abstraction in C. — 1984.

Учебное издание

Серия «Школа – Колледж – Университет»

Костюкова Нина Ивановна

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ СИ

Редактор *Л. А. Федотова*
Обложка *С. Л. Ярославцев*
Технический редактор *В. Н. Морошкин*
Корректор *П. В. Макаревич*
Компьютерная верстка *Т. В. Велигжанина*

Изд. лиц. ИД № 00313 от 22.10.99

Гигиенический сертификат
№ 54.НК.05.953.П.000146.12.02 от 02.11.2002

Подписано в печать 27.02.03. Формат 60 90/16. Бумага газетная. Гарнитура Таймс.
Печать офсетная. Усл. печ. л. 6,9. Уч.-изд. л. 9,9. Тираж 2000 экз. Заказ № 4008.

Сибирское университетское издательство
630058, г. Новосибирск, ул. Русская, 39

ФГУИПП «Советская Сибирь»
630048, г. Новосибирск, ул. Немировича-Данченко, 104

Уважаемые учащиеся и преподаватели информатики!

Данные методические пособия используются при обучении в Заочной школе информатики и программирования Высшего колледжа информатики ВКИ НГУ.

Обучение в Заочной школе (ЗШ) предполагает самостоятельное выполнение заданий по выбранному курсу. Школа работает круглогодично, и каждый слушатель может сам выбрать необходимый ему темп выполнения заданий.

Обучение учащихся в Заочной школе может осуществляться индивидуально или в форме «коллективного ученика». При индивидуальном обучении выполненные задания проверяют преподаватели колледжа. При обучении по форме «коллективный ученик» работы проверяются преподавателем школы, а преподаватели ВКИ НГУ консультируют и направляют его деятельность. Работа по форме «коллективный ученик» возможна только при наличии у преподавателя сертификата ВКИ НГУ.

Если Вы не имеете нашего сертификата, то Вы можете либо приехать в Колледж и пройти курс обучения очно, либо обучиться **заочно**.

Получив наш сертификат, Вы можете сами формировать группу учащихся для обучения в ЗШ, осуществлять контроль, проверку и аттестацию выполненных заданий. Итоги аттестации высылаются в Колледж для заключительной оценки и выдачи удостоверений слушателям ЗШ.

Условия приема в ЗШ ВКИ НГУ общие для преподавателей и учащихся:

1. Обучение в ЗШ платное.
2. После подтверждения оплаты Вам высылаются методические материалы по выбранному курсу.
3. Выполненные задания Вы отправляете по почте или по e-mail.
4. Если задание выполнено правильно, Вас аттестуют и высылают сертификат ВКИ НГУ.

Результаты обучения в Заочной школе учитываются при поступлении учащихся в ВКИ НГУ.

Справки по тел: (383-2) 33-79-44
e-mail: kulakova@ci.nsu.ru
сайт: www.ci.nsu.ru