Омский государственный технический университет

О. П. Шафеева, М. С. Дорошенко, А. Е. Симоненко

ПРОГРАММИРОВАНИЕ

Учебное пособие

Учебное текстовое локальное электронное издание

Рекомендовано редакционно-издательским советом
Омского государственного технического университета

Омск
Издательство ОмГТУ
2024

Сведения об издании: 1, 2, 3

ISBN 978-5-8149-3815-2

УДК 004.42(075)05.915(075)
ББК 32.973я7365.291.9-21я73
Ш30

Рецензенты:

Т. В. Вахний, к.ф.-м.н., доцент, доцент кафедры компьютерных технологий и сетей ФГАОУ ВО «Омский государственный университет им. Ф. М. Достоевского»;

О. Н. Лучко, к.п.н., профессор, зав. кафедрой информатики, математики и естественно-научных дисциплин ЧУОО ВО «Омская гуманитарная академия»

Шафеева, О. П. Программирование : учеб. пособие / О. П. Шафеева, М. С. Дорошенко, А. Е. Симоненко ; Ом. гос. техн. ун-т. – Омск : Изд-во ОмГТУ, 2024. – 1 CD-ROM (#,# Мб). – Минимальные систем. требования: процессор с частотой 800 МГц и выше ; 128 Мб RAM и более ; свободное место на жестком диске 300 Мб и более ; Linux / Windows XP и выше ; MacOS X 10.4 и выше ; CD/DVD-ROM-дисковод ; ПО для просмотра pdf-файлов. – Загл. с титул. экрана. – ISBN 978-5-8149-3815-2.

Представлен базовый материал по элементам, правилам и принципам программирования на языке C/C++. Приведены практические примеры алгоритмизации и программирования задач, а также задания для самостоятельного выполнения.

Издание предназначено для студентов очной формы обучения по направлениям 09.03.01 «Информатика и вычислительная техника», 09.03.02 «Информационные системы и технологии», 09.03.03 «Прикладная информатика», 09.03.04 «Программная инженерия».

Учебное текстовое локальное электронное издание

Шафеева Ольга Павловна
Дорошенко Марина Спартаковна
Симоненко Александр Евгеньевич

ПРОГРАММИРОВАНИЕ

Учебное пособие

Издание поставляется на одном CD-ROM-диске

Воспроизведение издания автоматическое –
без установки на жесткий диск компьютера

Для корректной работы с изданием на компьютере должны быть установлены
CD/DVD-ROM-дисковод и программное обеспечение
для просмотра pdf-файлов

Редактор *М. А. Болдырева*
Компьютерная верстка *Л. Ю. Бутаковой*

Для дизайна обложки использованы материалы
из открытых интернет-источников

Иллюстрации для издания предоставлены автором

Сводный темплан 2024 г.
Подписано к использованию 18.07.2024.
Объем #,## Мб. Тираж 10 эл. опт. дисков.

Издательство ОмГТУ. 644050, г. Омск, пр. Мира, 11; т. 8(3812)23-02-12.
Эл. почта: publisher@omgtu.ru, izdomgtu@mail.ru

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

C/C++ – это язык, отличающийся прозрачностью синтаксиса; он хорошо структурирован и удобен для обучения программированию [3, 5, 6]. Ему присущи средства современных языков программирования высокого уровня (структурность, модульность, определяемые типы данных) и средства программирования почти на уровне ассемблера (использование указателей, побитовых операций, операций сдвига).

C/C++ первоначально создавался для решения задач системного программирования. Затем появилось расширение языка в сторону объектно ориентированного программирования и был создан компилятор C/C++ [4, 8, 9].

К достоинствам языка C/C++ относятся быстрая компиляция программ, объединение компилятора с текстовым редактором, предупреждение об ошибках на уровне исходной программы, обширная библиотека подпрограмм и полезные расширения, упрощающие программирование.

В настоящем учебном пособии описаны элементы языка (алфавит, операции, операторы, типы данных), синтаксические правила и основные возможности языка. Практические варианты программирования продемонстрированы на примерах разработки программ для решения конкретных задач, начиная с проектирования схем алгоритмов и заканчивая текстами готовых программ. Рассмотрены особенности обработки массивов данных, реализации программ с подпрограммами, строками, структурами; программ, состоящих из нескольких файлов; особенности работы с динамической памятью.

Учебное пособие подготовлено с учетом опыта чтения лекций и проведения авторами занятий в компьютерных классах ОмГТУ. В первую очередь оно предназначено для студентов университета и является руководством по изучению программирования на языке C/C++.

1. БАЗОВЫЕ СРЕДСТВА ЯЗЫКА C/C++

1.1. Состав языка

Основным элементом любого языка является алфавит [3, 6]. Алфавит C/C++ включает:

– прописные и строчные буквы латинского алфавита (A...Z, a...z);

– прописные и строчные буквы русского алфавита (используются в комментариях и строковых константах);

– арабские цифры (0 1 2 3 4 5 6 7 8 9);

– символ подчеркивания (_);

– знаки пунктуации и специальные символы:

, . ; : ? ‘ ! | / \ ~ * ( ) [ ] { } + – = < > # % " & ^;

– пробельные символы: пробел (SP), символ табуляции (HT) (обычно восемь пробелов), возврат каретки (CR), перевод строки (LF или LH), перевод формата (FF). Эти символы отделяют одни объекты программы (константы, идентификаторы) от других;

– управляющие последовательности символов – это специальные сочетания литер, начинающиеся с обратной косой черты, за которой следует буква или комбинация цифр (табл. 1). Они используются в строковых литералах (в частности, в функциях ввода и вывода).

Если обратная косая черта предшествует символу, не включенному в таблицу, и не является цифрой, то черта игнорируется, а символ рассматривается как литера. Например, **\h** представляет символ **h** в строковой или символьной константе.

Идентификаторы определяют имена переменных, типов, функций и меток, используемых в программе.

Идентификатор состоит из последовательности латинских прописных и строчных букв, цифр и символов подчеркивания. Первым символом идентификатора не может быть цифра.

Компилятор C/C++ рассматривает прописные и строчные буквы как различные символы. Поэтому идентификаторы alfa, ALFA, alFA считаются различными.

Таблица 1

Управляющие последовательности символов

Управляющая последовательность	Наименование	Обозначение кода ASCII	Шестнадцатеричное значение
\a	Звонок	BEL	007
\b	Возврат на шаг (забой)	BS	008
\t	Горизонтальная табуляция	HT	009
\n	Перевод строки	HL LF	00A
\v	Вертикальная табуляция	VT	00B
\f	Перевод формата	FF	00C
\r	Возврат каретки	CR	00D
\"	Кавычки	"	022
\'	Апостроф	'	027
\\	Обратная косая черта	\	05C
\0	Нулевой символ (пусто)	NULL	
\ddd	Восьмеричный код символа		
\xddd	Шестнадцатеричный код символа		

Идентификатор может быть произвольной длины, но значащими для компилятора первой версии языка C (K&R) являются 8 символов, в C++ это ограничение снято.

1.2. Операции

Операции в C/C++ подразделяются на унарные (с одним операндом), бинарные (с двумя операндами) и тернарную (с тремя операндами). Операции, упорядоченные по приоритетам, сведены в табл. 2 [6, 8]. Тернарной является только одна условная операция с приоритетом 13.

Таблица 2

Приоритеты операций

Вес	Знак	Операция	Тип операции	Порядок выполнения
1	( ) [] . ->	вызов функции выделение элемента массива выделение элемента структуры или объединения выделение элемента структуры (или объединения), адресуемой(го) указателем	выражение	слева направо
2	! ~ - ++ -- & * (тип) sizeof	логическое отрицание побитовое отрицание изменение знака (унарный минус) инкремент (увеличение на единицу) декремент (уменьшение на единицу) определение адреса обращение по адресу преобразование типа определение размера в байтах	унарные	справа налево
3	* / %	умножение деление определение остатка от деления по модулю	бинарные арифметические	слева направо
4	+ -	сложение вычитание	бинарные арифмет.	слева направо
5	<< >>	сдвиг влево сдвиг вправо	сдвига	слева направо
6	< <= > >=	меньше чем меньше или равно больше чем больше или равно	отношения	слева направо
7	== !=	равно неравно		слева направо
8	&	побитовая операция И	поразрядная	слева направо
9	^	исключающее ИЛИ	поразрядная	слева направо
10	|	побитовая операция ИЛИ	поразрядная (побитовая)	слева направо

Окончание табл. 2

Вес	Знак	Операция	Тип операции	Порядок выполнения
11	&&	логическая операция И	логическая	слева направо
12	||	логическая операция ИЛИ	логическая	слева направо
13	? :	условная операция	тернарная	справа налево
14	= += -= *= /= %= <<= >>= &= |= ^=	простое присваивание сложение с присваиванием вычитание с присваиванием умножение с присваиванием деление с присваиванием выделение остатка от деления с присваиванием сдвиг двоичного числа влево с присваиванием сдвиг двоичного числа вправо с присваиванием побитовая операция И с присваиванием поразрядная операция ИЛИ с присваиванием исключающее ИЛИ с присваиванием	присваивания бинарные (для двоичных эквивалентов операндов)	справа налево
15	,	операция "запятая" (соединения)	бинарная	слева направо

Проанализируем некоторые особенности выполнения операций.

Операции присваивания делятся на простые, многоступенчатые и составные.

Простая операция присваивания имеет структуру:

<идентификатор> = <выражение>.

Тип правой части преобразуется к типу левой части.

В левой части может стоять идентификатор переменной либо адресное выражение, ссылающееся на переменную, которой присваивается значение.

Многоступенчатое присваивание применяется, когда одно значение присваивается нескольким переменным, например: i=j=k=6 эквивалентно выражению i=(j=(k=6)). Операция выполняется справа налево.

Составные присваивания объединяют арифметические или побитовые операции с присваиванием:

<идентификатор> **<знак операции>=** <выражение>

и выполняются как

<идентификатор> = <идентификатор> знак <выражение>.

Составные операции сведены в табл. 3, где показаны обозначения и в последнем столбце помещены примеры реализации.

Таблица 3

Составные операции

Знак	Операция	Пример	Эквивалент
=	простое присваивание	x = 3; y = 5	
+=	сложение с присваиванием	x += 1	$x = x + 1$
-=	вычитание с присваиванием	x -= y	$x = x - y$
*=	умножение с присваиванием	x *= y	$x = x \cdot y$
/=	деление с присваиванием	x /= 10	$x = x / 10$
%=	выделение остатка от деления с присваиванием	y %=10	$y = y \%\ 10$
++	увеличение на 1 (инкремент)	y++	$y = y + 1$
--	уменьшение на 1 (декремент)	x--	$x = x - 1$
>>=	сдвиг двоичного эквивалента числа X вправо на k разрядов	X =>> k	$X = X >> k$
<<=	сдвиг двоичного эквивалента числа X влево	X <<= k	$X = X << k$
&=	побитовая операция И с присваиванием для двоичных операндов	R &= m	$R = R \& m$
|=	поразрядная операция ИЛИ с присваиванием	R |= m	$R = R \mid m$
^=	исключающее ИЛИ (сложение по модулю 2)	R ^= m	$R = R \wedge m$

Арифметические операции (*, /, %, +, –) выполняются с учетом приоритета слева направо над целыми операндами и операндами плавающего типа. Исключение составляет операция нахождения остатка от деления (%), которая применяется только для целых операндов.

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

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

47/10 = 4; 47/(–10) = –4; –47/10 = –4 (к нулю);

9/4 = 2; 9.0/4 = 2,25; –47/10 = –5 (от нуля).

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

47 % 10 = 7; 47 % (–10) = 7; –47 % 10 = –7; –47 % (–10) = –7.

Операции увеличения и уменьшения. Операция увеличения на единицу **(инкремент «++»)** и операция уменьшения на единицу **(декремент «--»)** относятся к унарным операциям присваивания. Эти операции увеличивают **(«++»)** или уменьшают **(«--»)** значение переменной на единицу. Переменная может быть целого или плавающего типа либо указателем. Различают префиксную и постфиксную формы.

В префиксной форме в выражении **++N (--N) увеличение (уменьшение)** производится до использования переменной N, а в **N++ (N--) увеличение (уменьшение)** выполняется после использования N в выражении (т. е. после обработки основного выражения).

Операции отношения (<, >, <=, >=, ==, !=) сравнивают первый операнд со вторым. Результатом операции является 1 (единица), если проверяемое отношение истинно (не 0); и 0 (нуль), если ложно (нулевое).

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

Обычно операции отношений применяются при формировании условных выражений в условных операциях и операторах**,** а также в операторах цикла**.**

Операция унарный минус имеет приоритет 2 и изменяет знак операнда. Операнд должен быть целой или плавающей величиной. При реализации осуществляются обычные арифметические преобразования, например:

```
int a = 5, s, s1;
    s = -12;          // унарный минус для 12
    s1 = - a;         // унарный минус для переменной a.
```

В машинном представлении отрицательное число записывается в дополнительном коде.

Операция побитового дополнения (побитового отрицания) обозначается « ~ », имеет второй приоритет. Операция вырабатывает побитовое двоичное дополнение каждого разряда до единицы. Операнд должен быть целого типа. Возможны арифметические преобразования результата.

```
Пример: int a = -4,    res;      // двоичный код 1111 ...1100
            res= ~a;     // двоичный код 0000 ...0011     =>     res=3
```

Операции сдвига имеют следующую структуру:

операнд1 << операнд2 или операнд1 >> операнд2.

Производится сдвиг <операнда1> на число битов, указанных в <операнде2>. Оба операнда должны быть целыми величинами. Тип результата определяется типом левого операнда после обычных арифметических преобразований. Например: X << 2 // сдвиг X на 2 двоичных разряда влево.

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

```
Пример: int a = -2, res1, res2;        // 2=0010          -2 = 1111...1110
        res1 = a<<2;                   //   1111...1000          -8
        res2 = a>>1;                   //   1111...1111          -1
```

При реализации операций сдвига не диагностируются ситуации переполнения и потери значимости. Если результат операции сдвига не может быть представлен типом первого операнда после преобразования, то информация теряется.

Логические операции используют операнды целого, плавающих типов и типа указателя. Результатом является значение целого типа int (причем только «0» или «1»).

Операция логического И (x&&y) вырабатывает «1», если оба операнда (x и y) не равны «0», иначе – «0» (если хотя бы один из операндов равен «0»).

Операция логического ИЛИ (x || y) дает «0», если оба операнда (x и y) равны «0»; если x, или y, или оба не нулевые, то результат равен «1».

Операция логического отрицания НЕ (!x) дает «1», если операнд равен «0», и «0» в противном случае.

Например: пусть x = 0, y = 3;

тогда x && y = 0; x || y = 1; !x = 1; !y = 0.

Логические операции не выполняют преобразование типов. Они оценивают каждый операнд с точки зрения равенства нулю.

Операции одинакового приоритета в логических выражениях вычисляются слева направо. Если значения одного операнда достаточно для получения результата, то второй может не вычисляться.

Побитовые (поразрядные) **операции** применяются для операндов любых целых типов. С помощью данных операций поразрядно (побитно) сравниваются соответствующие биты первого и второго операндов.

Операция & (И) сравнивает каждый бит первого операнда с соответствующим битом второго операнда. Бит результата равен «1», если соответствующие биты исходных операндов равны «1», иначе бит результата равен «0».

Операция | (ИЛИ) также сравнивает операнды поразрядно (побитно): если оба сравниваемых бита равны нулю, то соответствующий бит результата равен «0», в противном случае (хотя бы один из сравниваемых битов равен «1») результат равен «1».

Операция ^ (исключающее ИЛИ) выполняется по правилу: бит результата равен «1», если биты исходных операндов различны, и «0», если одинаковы.

Например: для x1 = 1; y1 = 2; x2 = 1; y2 = 3

получатся следующие значения:

x1 & y1 = 0; x2 & y2 = 1;

x1 | y1 = 3; x2 | y2 = 3;

x1 ^ y1 = 3; x2 ^ y2 = 2.

Операция sizeof имеет второй приоритет и определяет размер памяти, соответствующий идентификатору или типу. Ее формат:

sizeof(<идентификатор>) или

sizeof(<выражение>) или

sizeof(<переменная>) или

sizeof(<тип>),

идентификатор не может быть именем функции или относиться к полю битов.

Например: для

```
int c, b;  char s[10];  double a;
        b = sizeof(s) ;     /* b = 10 байт */
        c = sizeof(a);      /* a = 8 байт*/
```

Если в качестве идентификатора указано имя массива, то результатом является размер всего массива, sizeof(<тип>) определяет число байтов для одного элемента типа.

Операция преобразования (приведения) типов используется для явного преобразования одного типа к другому. Ее формат:

(<тип>) <выражение> или (<тип>) <переменная>.

Результатом преобразования является значение нового типа, а тип и значение исходной переменной не меняются.

Например: для int a, a1, b=4; float z;

a = 2.7 + 1.9; // a=4 производится автоматическое преобразование

a1 = (int)2.7 + (int)1.8; // a1=3

z = sqrt((double)b); // z=2, b сохраняется типа int.

Структура **операции «адрес»**:

& <операнд>,

где & – унарная операция, которая возвращает адрес своего операнда.

В качестве операнда должен использоваться идентификатор (имя). Имя функции или массива также может быть операндом, хотя в последнем случае знак адрес (&) является излишним, поскольку имена функций и массивов в языке C/C++ сами являются адресами первого элемента.

Результатом операции «адрес» является указатель на операнд. Тип, адресуемый указателем, определяется типом операнда, например:

1) адреса используются в функции ввода

scanf ("%d, %d", &x, &y); // выполняется ввод значений по адресам;

2) int p=54; // в ячейке с адресом 1024 записано число 54;

printf ("p=%d, адрес = %d \n", p,&p); // выводит: p=54, адрес = 1024.

Конструкции типа &(x+1) или &5 в C/C++ не допускаются.

Указатель – это некоторое символическое представление адреса. В последнем примере символическое представление фактического адреса 1024

есть &p, которое в свою очередь является константой типа «указатель», так как адрес ячейки памяти, отводимой переменной p, в процессе выполнения программы не меняется.

В языке C/C++ имеются переменные типа указатель, их значением является адрес некоторой величины.

Пусть u – имя указателя, тогда оператор u = &p; присваивает адрес p переменной u: u указывает на p, причем u – это переменная, а &p – это константа (адрес).

Операция косвенной адресации (разадресации) в тексте программы осуществляет косвенный доступ к адресуемой величине через указатель. Тип результата определяется типом величины, на которую указывает указатель. Результат не определен, если указатель указывает на несуществующий адрес, например:

```
int b, v, *u;          // Объявляется указатель с именем u, который
                           //указывает на переменную типа int
u = &b;                // u присваивается адрес переменной b
v = *u;                /* станет v = b, т. е. присвоится значение, записанное по адресу переменной u */
```

Тип, адресуемый указателем, определяет количество байтов, извлекаемых из памяти при выполнении операции разадресации.

Операция соединения (запятая) – это двухаргументная операция последовательного вычисления. Она обозначается запятой и имеет структуру

<выражение1> , <выражение2>.

Используется для вычисления двух или более выражений там, где по синтаксису допустимо только одно. Выражения вычисляются последовательно, слева направо, преобразования типов не производится. Результат равен значению и типу второго выражения.

С помощью операций формируются выражения.

1.3. Структура простой программы

Программа на языке C/C++ состоит из директив препроцессора и ряда описаний (переменных, типов, функций), также может сопровождаться комментариями.

Любая программа включает одну или более функций, каждая из которых имеет заголовок. Одна из функций должна иметь имя **main** (главная), с нее начинается выполнение программы. Например, шаблон оформления программы может иметь следующий вид:

```
# директивы
# препроцессора
int main()  /*заголовок главной функции*/  //  int -  целый тип {
      тело главной функции
}
      mun F1 (параметры)
{
      /* тело функции F1 */
}
```

Главная функция обязательна в любой программе. Тело функции состоит из совокупности описаний и операторов (блока), ограниченных фигурными скобками. Каждый оператор заканчивается знаком « ; ».

Директивы препроцессора могут присутствовать в любом месте программы. Директива начинается знаком «#», который должен быть первым в строке. Препроцессор работает до компиляции программы и позволяет включать в текст программы библиотечные или другие файлы, определять макроподстановки. Директивы указывают действия, которые необходимо выполнить до передачи текста программы компилятору.

Практически во всех программах используются:

– директива # define, которая определяет текст с помощью «макроподстановок», например:

#define Pi 3.14159 // выполняет простую подстановку числа вместо идентификатора;

– директива #include подключает необходимую библиотеку: так # include <stdio.h> позволяет включить в программу стандартную библиотеку языка С **<stdio.h>,** в которой имеются функции форматированного ввода и вывода, без которых невозможно ввести данные и увидеть результат на экране дисплея,

include <math.h> и # include <stdlib.h>

подключают файлы, содержащие математические функции, в том числе и функцию генерации псевдослучайных чисел (rand(void)).

В состав библиотеки **<math.h>** входят следующие функции:

double **sin(**double **x)** – sin x**,**

double **cos(**double **x)** – cos x**,**

double **tan(**double **x)** – tg x**,**

double **sinh(**double **x)** – sh x**,**

double **cosh(**double **x)** – ch x**,**

double **tanh(**double **x)** – th x**,**

double **asin(**double **x)** – arcsin x**,**

double **atan(**double **x)** –arctg x**,**

double **acos(**double **x)** – arccos x**,**

double **floor(**double **x)** – округление с уменьшением,

double **exp(**double **x)** – e^x,

double l**og(**double **X)** – lnX,

double **log10(**double **X)** – $\log_{10}X$,

double **sqrt(**double **x)** – $\sqrt{x}$,

double **fabs(**double **x)** – |x|,

double **atan2(x, У)** – arctgY/x,

double **pow(**double **x,** double **y)** – x^y,

double **fmod(x, y)** – остаток от x/y,

double **ceil(**double **x)** – округление с увеличением,

где double – плавающий тип удвоенной точности.

Комментарии начинаются парой символов «/*» и заканчиваются «*/».

Например: /* комментарий (любой текст) */ или /* строка 1

строка 2 */.

Они размещаются везде, где допустимы пробелы. Вложенные комментарии (комментарий в комментарии) не допускаются. В версии С++

для однострочных комментариев может применяться пара символов «//», которые означают, что записанную после «//» часть строки надо воспринимать как комментарий**.**

Ниже представлен пример простой программы для сложения двух чисел x и y:

```
# include <stdio.h>       // подключение библиотеки для функции вывода
#include <locale.h>       // для использования русского шрифта
using namespace std;                  //  определение пространства имен
void main()                           //  главная функция
 /* если главная функция целого типа int main(), то она в конце должна возвращать значение целого типа оператором return */
{
        setlocale(LC_ALL, "RUS");     // для русского шрифта
        int x, y, z;                  // описание переменных целого типа
        x = 6;                        // операции присваивания
        y = 9;
        z = x + y;
        printf("сумма = %d \n", z);   // вызов функции форматированного
                                      // вывода данных из библиотеки <stdio.h>
/* В кавычках (для функции printf) записывается управляющая строка, которая показывает, как выводить данные, после нее указывается список переменных, для которых производится вывод значений. Значение переменной вставляется в шаблон, который начинается со знака «%»: если %d, то выдается целый результат в десятичном виде (z) */
}
```

Если функция ничего не возвращает, то ее имени обычно предшествует ключевое слово void и оператор возврата return в теле функции отсутствует.

1.4. Ввод и вывод данных в стандарте С

Для форматированного ввода и вывода данных применяются функции **scanf** и **printf** из библиотеки <stdio.h>.

1.4.1. Особенности форматированного вывода

Форматированный вывод данных выполняется функцией printf. Ее структура имеет вид:

printf("управляющая строка", параметры***)***;

или **printf** ("список форматов", параметр1, параметр2, ...).

«Управляющая строка» может включать обычные символы, управляющие последовательности (начинаются с символа ‘\ ‘) и шаблоны преобразования. Она заключается в кавычки. Обычные символы копируются в выходной поток (в примере простой программы – это «сумма =» (см. раздел 1.3)). Шаблон начинается знаком «%» и заканчивается символом преобразования, который указывает на тип выводимых данных. Число выводимых параметров должно соответствовать числу шаблонов в «управляющей строке».

Если выводимых параметров несколько, каждая спецификация преобразования соответствует одному из аргументов, которые следуют за строкой форматов; между ними устанавливается соответствие по порядку следования.

В качестве символов преобразования используются буквы:

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

u – для десятичного целого без знака,

f – для числа с плавающей точкой в естественной форме (float),

e (E) – для числа с плавающей точкой в экспоненциальной форме,

g (G) – для вывода наиболее короткой из двух форм e или f записи числа,

c – для отдельного символа,

s – для вывода строки символов,

o – для восьмеричного числа,

x (X) – для вывода шестнадцатеричного числа со строчными буквами (с прописными буквами).

Перед символом преобразования может стоять числовой коэффициент, явно показывающий количество позиций в выводимой строке, например:

printf (" %c = %5d \n", ‘z’, z); выводит z = значение числа из переменной z.

В общем случае шаблон преобразования записывается в виде

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

Квадратные скобки означают, что данное поле может отсутствовать.

В качестве значений поля **[флажок]** могут быть следующие знаки:

«–» (минус) – производит выравнивание по левой границе поля (по умолчанию выравнивание выполняется по правой границе),

«+» (плюс) – указывает на наличие ведущего знака,

« » (пробел) – указывает на наличие ведущего пробела,

«0» (нуль) – предполагает наличие ведущего нуля,

«#» – альтернативная выходная форма.

Поле **[длина]** задается целым положительным числом и определяет количество выводимых символов. Если количество символов в выводимом аргументе меньше данного параметра, то число дополняется пробелами, если больше (или минимальный размер не задан), – параметр игнорируется и число выводится полностью в соответствии с полем [.точность], если оно задано.

Поле **[.точность]** определяет:

а) для чисел с плавающей точкой – количество цифр после десятичной точки, значение округляется в соответствии с заданным количеством цифр. Для формата f, если десятичная точка присутствует, то в числе ей предшествует по меньшей мере одна цифра. По умолчанию точность равна шести. Если точность равна 0, то десятичная точка не выводится;

б) для строк – это максимальное количество выводимых символов, а [длина] – общее поле для вывода. Символы, превышающие точность, не печатаются. По умолчанию выводятся все символы до нулевого (/0).

Например: "%10.5s", "базы данных" /* напечатается: " базы " */

В качестве **модификатора** используются буквы:

h – для вывода данных типа short int,

l – для типов long int (например, ld), double (lg, lf или le),

L – для типа long double (Lf, Lg, Le).

Приведем примеры вывода чисел в соответствии с указанным в функции вывода шаблоном.

```
#include <stdio.h>
    void main()
{                                          // строка вывода:
    printf ("/ %d / \n", 53);              // | / 53 /
    printf ("/ %1d / \n", 53);             // | / 53 /  1 игнорируется
    printf ("/ %5d / \n", 64);             // | /    64 /
    printf ("/ %-5d / \n", 64);            // | / 64     /
    printf ("/ %f / \n", 535.48);          // | / 535.480000 /
    printf ("/ %e / \n", 535.48);          // | / 5.354800e02 /
    printf ("/ %6.2f / \n", 535.48);       // | / 535.48 /
    printf ("/ %5.1f / \n", 535.48);       // | / 535.5 /
    printf ("/ %3.1f / \n", 535.48);       // | / 535.5 / 3 до точки игнорируется
    printf ("/ %4.0f / \n", 535.48);       // | /  535 /
    printf ("/ %10.3e / \n", 535.48);      // | /  5.355e+02 /
}
```

Если в примере для сложения (см. выше) используются три шаблона, функция вывода будет имеет вид:

```
printf("Значения x, y, z соответственно равны %d, %d, %d \n", x, y, z).
```

Если выводятся несколько параметров, то столько же должно быть и шаблонов в управляющей строке.

1.4.2. Форматированный ввод

Для ввода данных используется функция форматированного ввода

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

или

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

где a1, a2 – имена переменных или аргументов; & – признак взятия адреса. Аргументы функции должны являться адресами (указателями на переменные).

В управляющей строке используется свой шаблон со следующей структурой:

[*] [ширина] [модификатор] символ преобразования.

Символ * подавляет присваивание очередного входного поля (например: *d означает, что переменная целого типа читается, но не сохраняется), **[ширина]** – положительное десятичное число, которое задает число символов для чтения из входного потока в указанном символом преобразования типе, назначение остальных полей совпадает с полями функции для вывода.

Отметим особенности применения символов преобразования:

– %f, %e, %g при вводе эквивалентны (они допускают наличие или отсутствие знака, строки цифр с десятичной точкой или без нее и поля показателя степени);

– %c (для символа) необходимо учитывать то, что он читает все символы, в том числе и пробельные.

Приведем пример программы для сложения двух чисел **x** и **y**, вводимых с клавиатуры:

```
# include <stdio.h>      // подключение библиотеки для функции вывода
# include <locale.h>     // для использования русского шрифта
# include <conio.h>                  // для функции getch ();
using namespace std;                 //  определение пространства имен
int main()                           //  главная функция
{
```

```
    setlocale(LC_ALL, "RUS");          // для русского шрифта
    int x, y, z;                       // описание переменных целого типа
    printf("Введи x и y \n");          // элемент диалога
    scanf ("%d %d", &x, &y);           // ввод x и y по адресам как целые
    x = 6;
    y = 9;
    z = x + y;                         // операция присваивания
    printf("сумма = %d \n", z);        // форматированный вывод данных
    return 0;
}
```

Управляющая строка функции scanf указывает, какие данные ожидаются на входе. Если функция встречает в форматной строке %, за которым следует символ преобразования, то она будет пропускать на входе пустые символы, пока не встретится непустой, – с него начнется ввод.

В примере управляющая строка предписывает функции scanf ввести десятичное целое число, которое нужно поместить в переменную x, затем продвинуться к следующему непустому символу (на что указывает пробел после %d) и ввести число, которое присвоится переменной y.

Пробельные символы игнорируются. Обычные символы указывают, что их надо читать из входного потока, но не сохранять. Если символ во входной строке не совпадает с символом в управляющей, то ввод завершается:

```
scanf (" %d, %d ", &x , &y);          // входная строка:        4 , 5
```

Запятая должна присутствовать во входной строке, если она есть в шаблоне для ввода.

Если за % следует символ, не являющийся управляющим, то он сам и все последующие символы до очередного знака % рассматриваются как обычная строка, которая должна совпадать со вводимой.

Сама функция scanf() как результат выдает значение числа успешно опознанных и присвоенных входных элементов, например:

```
int i, n;                    // при вводе зададим любое целое число;
n = scanf("%d", &i);     // n=1 (введено одно число)
```

Можно воспользоваться функцией защищенного ввода **scanf_s()**, которая отличается от scanf() тем, что не вызывает переполнения буфера при чтении данных и применяется для дополнительной безопасности (в Microsoft). Эта функция может содержать дополнительный параметр для задания максимального количества символов (целым числом после списка адресов переменных), записываемых в буфер.

1.4.3. Функции ввода и вывода символов

В библиотеке <stdio.h> предусмотрены также функции для чтения и записи отдельных символов:

getchar() выполняет ввод символа из стандартного входного потока (stdin) – с клавиатуры, например: c = getchar() присваивает переменной "c" входной символ;

putchar(c) выводит содержимое "c" на дисплей (в стандартный поток вывода stdout).

Примеры проектирования простых программ с оператором присваивания

В арифметических выражениях могут использоваться операции (см. табл. 2) и математические функции из библиотек <stdlib.h> и **<**math.h**>** [3, 6].

Пример 1. Спроектировать схему алгоритма (СА), написать программу, проверить ее для следующей задачи. Вычислить площадь прямоугольника по длинам его сторон A и B. Обозначим площадь переменной S, необходимо найти S = A*B.

Программа набирается в окне редактора C++ и компилируется.

Сначала подключаются библиотеки

```
# include <stdio.h>      // для ввода/вывода
# include <conio.h>      // для функций работы с экраном
void main ()             // заголовок главной программы
{
```

```
int A=2, B=3, S;                    // объявление переменных целого типа
S  =A*B ;                           // оператор «выражение»
printf ("\n Площадь прямоугольника = %d\n", S);          // вывод
_getch();   // вызов функции задержки работы программы до нажатия любой клавиши
}
```

В результате выполнения программы на экран выводится сообщение:

Площадь прямоугольника = 6.

Пример 2. Найти площадь равностороннего треугольника. Если стороны треугольника имеют размер A, то его площадь $S = A^2 \sin(\pi/3)/2$. Проектирование начинается с разработки схемы алгоритма (СА) [4, 1]. СА для этой задачи выполнена по ГОСТ 19.701–90 [1] (см. приложение) и приведена на рис. 1. В соответствии с определенными в алгоритме действиями и синтаксисом языка C/C++ разработана и отлажена программа:

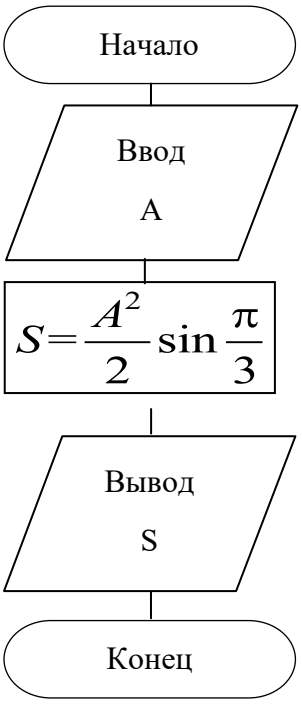


Рис. 1

```
#include <math.h>
#include <stdio.h>
#include <conio.h>
#define  PI  3.1415 // директива постановки
void main()
{
float A, S;   //объявление переменных плавающего типа
printf ( " Введите A");
scanf ("%f", &A);     // ввод
S=A*A*sin(PI/3)/2;
printf (" Результат : S=%6.2f  для A=%f\n", S, A);
_getch();            // из библиотеки <conio.h>
}
```

После выполнения программы на экран выводится строка:

Результат: S = 10.83 для A = 5.000000 (если ввели 5).

Предварительно рассчитанный на калькуляторе результат для A = 5 составил S = 10.8253, что подтверждает верность программы.

Задание 1 для разработки простых программ

Задача 1 (программа1_1)

Вариант	x_1	x_2	a	b	c	Вычислить
1	1	2	1.5	2	–0.7	$w = ae^{-\sqrt{x}}\cos(bx) + c^5$
2	1	2	0.7	–	2.1	$y = \frac{\sin x}{\sqrt{1 + a^2 \sin^2 x}} - c \ln ax$
3	0	2	2	1.2	1	$z = \sqrt{a + b - e^{\sin x} + c}$
4	1	2	4.1	–2.3	–	$w = \frac{x^3 + a^2 \cos 2x}{x + \sqrt{a + b \sin 3x}}$
5	–1	1	0.5	2	1.5	$y = bx^2 e^{a x^2} + a\sqrt{x + 1{,}5}$
6	2	4	0.5	1.3	–	$z = e^{-ax} \frac{x + \sqrt[4]{x + a}}{x + \sqrt{x - b}}$
7	0	2	0.5	1	–	$w = e^{-a^2}\sqrt{x + 1} + e^{bx}\sqrt[3]{x + 1}$
8	–1	1	2.7	1.7	–	$y = \frac{a^2}{(x + 2)e^{-bx}} + \ln(a + bx)$
9	0	π/2	2	0.7	0.5	$z = a^3 \cos(bx \sin^2 x) + \sqrt[3]{c}$
10	1	0	0.5	2.9	1.5	$w = \frac{ax + 2^{-x}\cos(bx)}{bx - e^{-x}\sin(bx) + c}$
11	1	2	1.5	–1.2	–	$y = e^{-ax}\sqrt[3]{ax + b \sin 2x}$
12	1	2	0.5	1.7	2	$z = e^{-bx}\sin(cx + b) - \sqrt{bx + a}$
13	–1	1	0.5	1.2	–	$w = 2^{-x} arctg(x + a) - 3^{-bx}\cos bx$

Вариант	x_1	x_2	a	b	c	Вычислить
14	1	2	0.5	3.1	–	$y = \sqrt{\left\| ax\sin^2 x \right\|} + e^{-2x}(x+b)$
15	1	2	0.5	3.2	–	$z = e^{2x}\ln(a+x) - b^{3x}\lg(b-x)$
16	0	1	1	2	4	$w = \frac{\sqrt{\lvert x-a\rvert - \sqrt[3]{\lvert x\rvert}}}{a+bx^3+cx^2}$
17	0	1	1	3	–	$y = \frac{b+e^{x-1}}{a+x\left\| x-tgx \right\|}$
18	–1	2	1	2	–	$z = \frac{a+\cos(x-b)}{x^4/2+\sin^2 x}$
19	1	2	2	3	1	$w = \left\|\ln\right\|(b-\sqrt{\lvert x\rvert})(x^2 - \frac{c}{a+x/4})$
20	–1	1	2	π/6	0,5	$y = \frac{a\cos(x-b)}{c+\sin^2 x} + 2^x$

Задача 2 (программа1_2)

1. Идет k-я секунда суток. Определить, сколько целых часов (h), целых минут (m) и секунд (s) прошло к этому моменту.

2. Определить площадь трапеции с основаниями a, b, высотой h и объем усеченного конуса, если считать a, b площадями оснований.

3. Определить координаты центра тяжести трех материальных точек с массами m_1, m_2, m_3 и координатами (x_1, y_1), (x_2, y_2), (x_3, y_3).

4. Вычислить по заданному радиусу R объем шара и площадь круга, найти соотношение между ними.

5. Вычислить медианы треугольника по заданным сторонам a, b, c.

6. Вычислить площадь поверхности и объем конуса по заданным радиусам и высоте h.

7. Ввести двузначное целое число x < 15. Написать программу перевода его в восьмеричную систему счисления делением на 8 и проверить через форматированный вывод через спецификатор %o.

8. Вычислить, какая идет секунда суток при положении стрелок в h часов, m минут и s секунд, а также угол (в градусах) между положением часовой стрелки в начале суток и ее положением в указанный момент.

9. По заданному радиусу найти объем шара и площадь поверхности.

10. Ввести координаты двух точек (x_1, y_1), (x_2, y_2). Найти расстояние между ними.

Для задания 2 номер варианта определяется остатком от деления индивидуального варианта на число 10.

1.5. Стандартные типы данных

В программе на языке C/C++ все используемые переменные до их применения должны быть объявлены. Они характеризуются типом и классом памяти. Объявление типа данных имеет формат следующего вида:

[<класс памяти>] <тип> <идентификатор_1>

[[= нач.значение1] [, <идентификатор_2>[= нач.значение2], ...];

Класс памяти определяет область действия идентификатора и может в объявлении отсутствовать (квадратными скобками выделены необязательные элементы). Простейшее объявление типа имеет структуру

<тип> <идентификатор>;

например: int i; double d;

При объявлении переменных можно задавать начальные значения в виде

<идентификатор> = <значение>

Класс **auto** (автоматический) используется для явного описания локализованных в блоке переменных. Область действия ограничена той функцией или блоком, в котором она объявлена. Говорят, что такая переменная локализована в блоке: она начинает существовать после обращения к функции и исчезает после выхода из нее. Таким образом не занимает место в памяти, если не используется. Кроме того, поскольку она локализована в неко-

торой функции, значение ее не может быть случайно изменено действиями других функций. Например:

auto int A; auto int X=145; int x=1248;

Если идентификатор класса памяти отсутствует (по умолчанию), подразумевается опущенный в описателе класс **auto**. Если явно не задано значение переменной, то она в классе auto нулем не инициализируется.

Класс **extern** (внешний) используется для объявления глобальных переменных, описанных ниже или в другом файле (т. е. для определения ссылок на внешние переменные.

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

Переменные класса **static (**статический) подобно автоматическим переменным локализуются в блоке или функции, где они описаны, но при выходе из блока значения сохраняют.

Примеры объявлений:

static char x, y; **static** int f = 1; **register** int k;

Классификация основных типов показана на рис. 2.

Типы **char** (символьный), **int** (целый), **short** (короткий целый), **long** (длинный целый) могут быть модифицированы ключевыми словами **signed** (со знаком) и **unsigned** (без знака). При отсутствии модификатора по умолчанию принимается модификатор **signed**.

Названия типов с указанием диапазонов допустимых величин приведены в табл. 4. Стандартные типы могут иметь эквивалентные названия (синонимы).

Имя типа, состоящее из одного модификатора (**signed или unsigned**), эквивалентно имени, содержащему дополнительно ключевое слово **int** (соответственно **signed int** или **unsigned int**). Например:

запись signed K; эквивалентна signed int K;

запись unsigned l; эквивалентна unsigned int l.

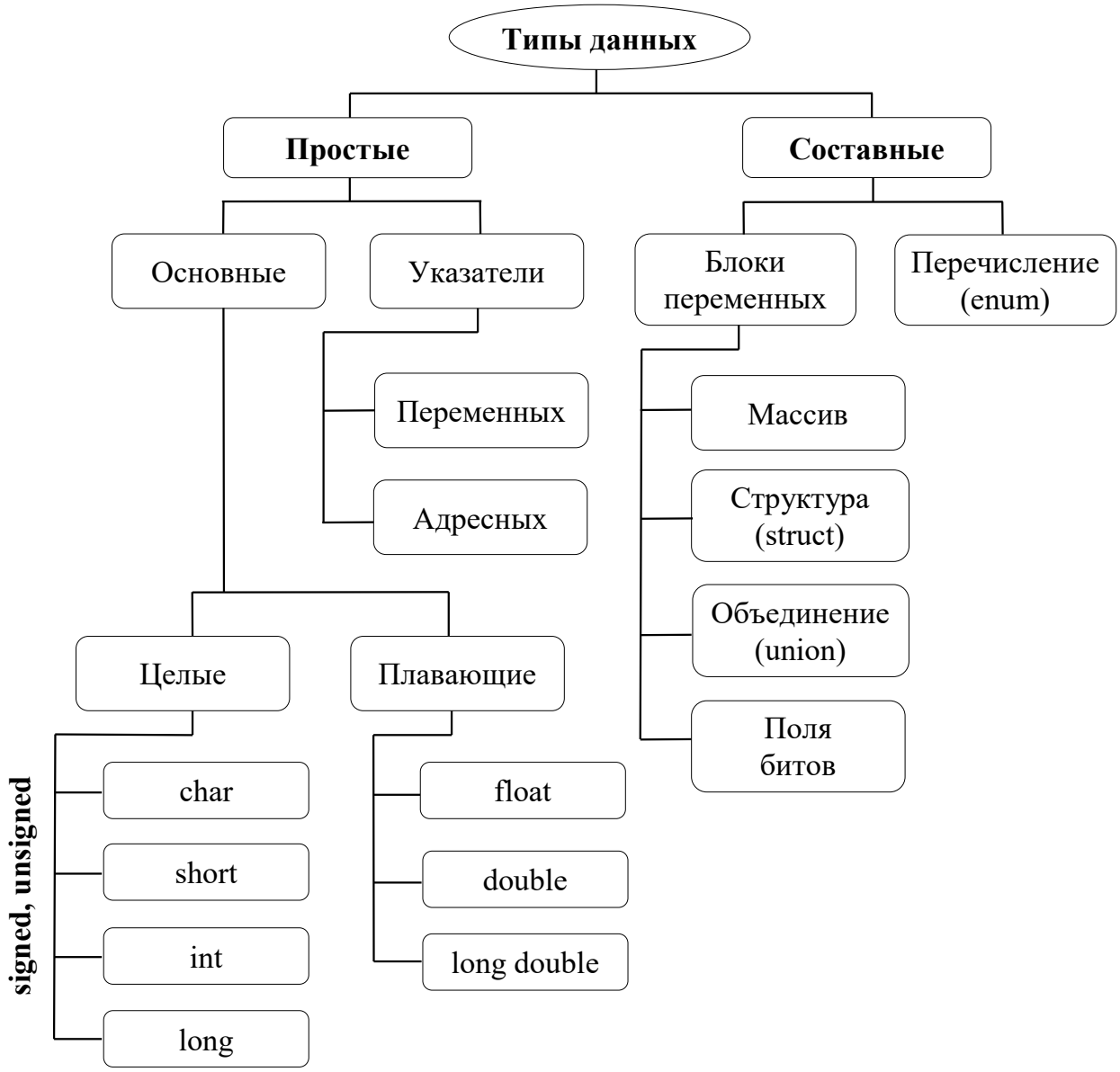


Рис. 2

Таблица 4

Типы данных

№ п/п	Обозначение	Размер (байт)	Диапазон	Тип данных
1	char, signed char	1	–128...127	Символьный со знаком
2	unsigned char	1	0...255	Символьный без знака
3	short, short int, signed short, signed short int	2	–32768...32767	Короткое целое со знаком
4	unsigned short, unsigned short int	2	0...65535	Короткое целое без знака

Окончание табл. 4

№ п/п	Обозначение	Размер (байт)	Диапазон	Тип данных
5	int, signed, signed int	1, 2, 4	Зависит от реализации	Целое
6	unsigned, unsigned int	1, 2, 4	Зависит от реализации	Целое без знака
7	long, signed long, long int	4	–2147483648… 2147483647	Длинное целое со знаком
8	unsigned long	4	0...4294967295	Длинное целое без знака
9	float	4	–3.4e-38...3.14e+38	Вещественное число с плавающей точкой
10	double	8	–1.7e-308...1.7e308	Вещественное число удвоенной точности
11	long double	10	–3.4e-4932... 3.4e4932	Длинное вещественное число удвоенной точности

Значением переменной типа **char** является код из набора кодов ASCII. По умолчанию тип **char** интерпретируется как однобайтовая целая величина в диапазоне –128…127, причем только значения от 0 до 127 имеют символьные эквиваленты. Переменная типа unsigned char запоминает значения в диапазоне 0…255. Поэтому при необходимости использования в качестве значений переменной типа char русских букв надо объявлять переменную типом unsigned char, так как русские буквы имеют коды больше 127.

Представление в памяти и диапазон значений для типов int и unsigned int не определены в стандарте языка. По умолчанию размер int соответствует размеру целого на конкретном компьютере (для16-разрядных – два байта, для 32-разрядных – четыре байта). Таким образом, тип int может быть эквивалентен типам short int или long int в зависимости от реализации.

Диапазон целочисленных значений определен в стандартном файле <limits.h>. Восьмеричные и шестнадцатеричные переменные могут быть типа signed или unsigned в зависимости от их размера.

Значения переменных плавающего типа float занимают четыре байта и состоят из знака, экспоненциальной части и мантиссы (32 бита). Диапазон значений чисел с плавающей точкой определен в файле <float.h>.

Величины типа double занимают восемь байт (один юит отводится под знак, 11 – под порядок и 52 – под мантиссу).

Тип long double семантически эквивалентен типу double.

Для определения числа байтов специфического объекта или требований для хранения типа имеется операция sizeof().

1.6. Операторы C/C++

Все операторы языка Си могут быть условно разделены на категории:

1) условные операторы (if ... else, оператор выбора switch);

2) операторы переходов (goto, break, continue, return);

3) операторы цикла (for, while, do-while);

4) другие операторы (оператор «выражение», пустой, составной).

1.6.1. Оператор «выражение»

Выражение – комбинация знаков операндов и операций. Результатом выражения является отдельное значение.

Каждый операнд в свою очередь может являться другим выражением. Значение выражения зависит от расположения, приоритета операций и наличия круглых скобок. В простейшем случае операндом является константа, переменная, вызов функции, индексное выражение.

В языке C/C++ операции присваивания также являются выражениями (см. табл. 3).

При вычислении выражений тип каждого операнда может быть преобразован к другому типу. Для явного способа используется операция приведения типов. Неявное преобразование производится в соответствии с правилами арифметических преобразований к большему.

Важной особенностью языка является возможность превращения любого выражения в оператор. Для этого достаточно закончить выражение точкой с запятой ("**;**"). Выражение присваивания, законченное ";" становится оператором. Выполнение оператора «выражение» состоит в вычислении значения и присваивании этого значения переменной левой части.

Например: выражение x = 6 определяет операцию «присваивания», а **x = 6**; – оператор языка Си, который присваивает переменной x значение 6.

Таким образом, оператором является любое выражение, заканчивающееся ";". Например:

```
1) i++;    2) i=i+1;          // увеличение переменной на единицу
3) y = cos(x);                // выражение включает операцию
                              // присваивания и вызов функции
4) printf(…);          //оператор "выражение" с вызовом функции вывода.
```

1.6.2. Операторы ветвления и условная операция

1.6.2.1. Условный оператор

Условный оператор в зависимости от исходных или промежуточных данных позволяет выполнить ту или иную последовательность действий.

Структура условного оператора:

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

(если) *(иначе)*

Если <выражение> истинно (не нуль), то выполняется <оператор1>, если ложно (или равно нулю), то – <оператор2>. Например,
для нахождения наименьшего из двух чисел **a** и **b** можно применить следующий условный оператор:

```
if (a>b) min = b;
else min = a; // в переменной min окажется наименьшее значение.
```

Существует сокращенная форма условного оператора, когда часть **[else <оператор2>];** отсутствует. Например: для нахождения наименьшего из трех чисел **a**, **b** и **c** нужны два условных оператора (полный и сокращенный):

```
if (a>b) min = b;
else min = a;
if (c<min) min = c;       // сокращенный оператор if .
```

Допускается использование вложенных операторов **if** в любой части (if или else). Кроме того, в любой части **if** можно разметить несколько операторов, заключив их в фигурные скобки, например:

```
if ( i > j )  i++;
else
     {
     j--;            //  j = j – 1;
     k++;            //  k = k + 1;
     }
```

Пример. Разработать программу для начисления базовой зарплаты согласно следующему правилу: если стаж работы сотрудника менее трех лет, то зарплата равна 100$, при стаже работы от трех до пяти лет – 150$, свыше пяти лет зарплата повышается с каждым годом на 10$, причем при стаже, превышающем 20 лет, она составляет 300$. Для программирования процесса решения этой задачи определим математическую формулировку задачи:

$$ZP = \begin{cases} 100, & \text{если } ST < 3; \\ 150, & \text{если } 3 \le ST \le 5; \\ 150 + (ST - 5) * 10, & \text{если } 5 < ST \le 20; \\ 300, & \text{если } ST > 20, \end{cases}$$

где ZP – зарплата, ST – стаж работы.

Далее построим СА (рис. 3), ей соответствует следующая программа:

```
#include <stdio.h>
#include <conio.h>
void main ()
{
int ST;       // стаж целого типа
float ZP;     // ZP плавающего типа
printf ("\n Введите стаж ");
scanf ("%d",&ST);
if (ST<3) ZP = 100;
else if (ST<5) ZP = 150;
      else if (ST>=20) ZP=300;
            else ZP = 150 + (ST-5)*10;
printf ("\n Зарплата = %10.2f$\n", ZP);
_getch();
}
```

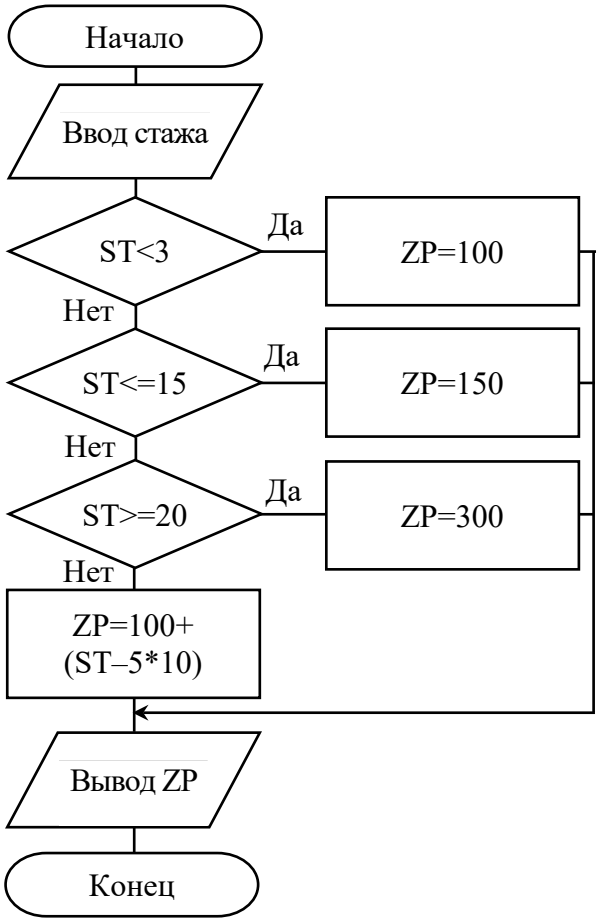


Рис. 3

1.6.2.2. Условная операция

Применение условных операций вместо условных операторов в ряде случаев приводит к получению более короткой программы. Ее формат

<выражение1> ? <выражение2> : <выражение3>

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

Алгоритм выполнения условной операции заключается в следующем: вычисляется <выражение1> и сравнивается с нулем; если его значение не равно 0 (истинно), то вычисляется <выражение2>; если равно 0 (ложно), то вычисляется <выражение3>. Вычисляется лишь одно <выражение>, его значение и станет результатом выполнения операции. Оно может быть

присвоено какой-либо переменной. Например: найти максимальное из двух значений **max = (a>b)? a: b;** // переменной **ma**x присваивается **a** или **b** в зависимости от выполнения условия **a>b**?

Если операнды различаются по типу, то тип результата задается правилами арифметического преобразования (от меньшего к большему).

1.6.2.3. Оператор выбора (оператор-переключатель)

Оператор выбора (или оператор-переключатель) предназначен для организации выбора одного или нескольких вариантов продолжения программы из множества различных. Формат оператора:

```
        switch (<выражение>)
{
                case <константа1> : <список1 операторов>;
                case <константа2> : <список2 операторов>;
                    ...
                [default : <операторы>;]
}
```

где <выражение> – любое допустимое в C/C++ выражение целого или символьного типа.

Оператор сравнивает значение <выражения> с константами во всех вариантах **case** и передает управление оператору, константа для которого соответствует значению <выражения> (это точка входа). Далее все операторы выполняются последовательно, независимо от меток. Каждый вариант **case** может быть помечен целой или символьной константой или константным выражением. Константное выражение не может включать переменные или вызовы функций (оно вычисляется на этапе компиляции). Операторы в строке **default** выполняются в том случае, если ни одна из констант не совпала со значением <выражение>. Вариант **default** не обязателен, а если он есть, то может быть записан не обязательно последним.

Если ни одна константа не совпадает со значением <выражения> и вариант **default** отсутствует, то никаких действий не производится и управление передается на следующий за **switch** оператор.

В операторе **switch** никакие две константы не могут иметь одинаковые значения. Если одни и те же действия необходимо выполнить для различных значений выражения, можно пометить один и тот же оператор несколькими метками, например:

```
int a;
scanf("%d", &a);
switch (a)
{
case 1: func1();
                    case 2: func2();
                     case 0:
                    case 4: func3();
                    default:  printf("bye\n");
              }
```

При **a = 2** выполнение начинается с оператора с меткой **case2:** и будут выполнены функции **func2**, **func3** и **printf**. Для значений **case0** и **case4** выполняются одни и те же действия: **func3**, **printf**.

Допускается использование вложенных операторов **switch**, тогда в вариантах case внешнего и внутреннего switch могут использоваться одинаковые константы.

Для обхода вычисления каких-либо функций или выхода из ветви **case** обычно используется оператор **break**, который выполняет и выход из оператора **switch**.

В теле оператора **switch** могут присутствовать объявления и инициализация переменных. Программист должен следить за точками входа в тело этого оператора, чтобы объявления не были пропущены.

Пример программирования многовыходных разветвлений. Напечатать в зависимости от числа углов название фигуры (треугольник, четырехугольник, пятиугольник, шестиугольник, многоугольник).

Вариант реализации задачи с применением оператора выбора представлен СА (рис. 4) и следующей программой:

```
      #include <stdio.h>
      #include <conio.h>
void main ()
{
int T; // T – число углов
printf ("Введите число углов ");
scanf ("%d", &T);
switch (T)
  {
     case 1:
     case 2 :  printf ("Это не фигура\n ");
                                        break;
     case 3: printf ("С %d углами –
              треугольник\n ", T);    break;
     case 4: printf ("С %d  углами –
          четырехугольник\n ", T); break;
     case 5 : printf ("С %d  углами –
          пятиугольник\n ", T );      break;
     case 6:  printf ("С %d  углами –
          шестиугольник\n ", T );   break;
     default : printf ("С %d  углами –
                    многоугольник\n ", T );
          }
_getch();
}
```

Рис. 4

Задание 2 для программирования задач с ветвлениями

Задача 1 (программа2_1)

Вычислить значение функции в зависимости от интервала, в который попадает вводимый с клавиатуры аргумент, с проверкой принадлежности аргумента допустимому интервалу.

1. Для $t \in [0, 3]$,
где $a = -0.5$, $b = 2$

$$z = \begin{cases} a\,t^2 \ln t & \text{при } 1 \le t \le 2, \\ 1 & \text{при } t < 1, \\ e^{a\,t} \cos b\,t & \text{при } t > 2, \end{cases}$$

2. Для $x \in [0, 4]$,
где $a = 2.3$

$$f = \begin{cases} \sqrt[5]{x + a} & \text{при } x > 2, \\ x & \text{при } 0.3 < x \le 2, \\ \cos(x - a) & \text{при } x \le 0.3, \end{cases}$$

3. Для $x \in [0, 7]$,
где $a = -2.7$, $b = -0.27$

$$z = \begin{cases} (a + b)/(e^x + \cos x) & \text{при } 0 \le x < 2.3, \\ (a + b)/(x + 1) & \text{при } 2.3 \le x < 5, \\ e^x + \sin x & \text{при } 7 \ge x \ge 5, \end{cases}$$

4. Для $i \in [7, 12]$,
где $a = 2.2$, $b = 0.3$

$$y = \begin{cases} a\,i^4 + b\,i & \text{при } i < 10, \\ \operatorname{tg}(i + 0.5) & \text{при } i = 10, \\ e^{2i} + \sqrt{a^2 + i^2} & \text{при } i > 10, \end{cases}$$

5. Для $x \in [0.9, 5]$,
где $a = 1.5$

$$y = \begin{cases} \pi\,x^2 - 7/x^2 & \text{при } x < 1.3, \\ ax^3 + 7\sqrt{x} & \text{при } 1.3 \le x < 3, \\ \lg(x + 7\sqrt{x}) & \text{при } x \ge 3, \end{cases}$$

6. Для $t \in [-1, 4]$,
где $a = 2.1$, $b = 0.37$

$$z = \begin{cases} \sqrt{at^2 + b\sin t + 1} & \text{при } t < 0.1, \\ at + b & \text{при } 0.1 \le t < 2, \\ \sqrt{at^2 + b\cos t + 1} & \text{при } t \ge 2, \end{cases}$$

7. Для $x \in [0, 6]$,
где $a = 1.5$

$$y = \begin{cases} a\,e^{\sin x} + 2.5 & \text{при } x < 0.3, \\ e^{\cos x} + a & \text{при } 0.3 \le x < 4, \\ \sin x & \text{при } x \ge 4, \end{cases}$$

8. Для $x \in [1, 6]$,
где $a = 1.8$,
$b = -0.5$, $c = 3.5$

$$y = \begin{cases} (\sin x)/(a + e^{x}) & \text{при } x \geq 4, \\ a/x + b\,x^{2} - c & \text{при } x \leq 1.2, \\ (a + bx)/\sqrt{x+1} & \text{при } 1.2 < x < 4, \end{cases}$$

9. Для $t \in [1, 5]$,
где $a = 2.5$

$$z = \begin{cases} t\sqrt[3]{t-a} & \text{при } t > a, \\ t \sin a\,t & \text{при } t = a, \\ e^{-at} \cos a\,t & \text{при } t < a, \end{cases}$$

10. Для $x \in [0, 4]$,
где $a = 1$, $b = 3$

$$y = \begin{cases} e^{-bx} \sin b\,x & \text{при } x < 2.3, \\ \cos bx & \text{при } 2.3 \leq x < 3, \\ e^{-ax} \cos b\,x & \text{при } x \geq 3, \end{cases}$$

11. Для $t \in [0.5, 8]$,
где $a = 1.3$, $b = 6.5$

$$z = \begin{cases} a\,t^{2} - b\,\sqrt{t+1} & \text{при } t < a, \\ a - b & \text{при } a \leq t \leq b, \\ a\,t^{2/3} - \sqrt[3]{t+a} & \text{при } t > b, \end{cases}$$

12. Для $x \in [0, 2]$,
где $b = -2.9$

$$y = \begin{cases} |e^{-2x} \sin bx| & \text{при } x > 1, \\ \cos bx & \text{при } x = 1, \\ e^{-x} \cos bx & \text{при } x < 1, \end{cases}$$

13. Для $x \in [0.5, 2]$
где $a = -0.8$

$$z = \begin{cases} \sin(\cos a\,x) & \text{при } x > 1, \\ \operatorname{tg} ax & \text{при } x = 1, \\ a^{2}\,x & \text{при } x < 1, \end{cases}$$

14. Для $x \in [1, 2]$,
где $b = 1.3$.

$$y = \begin{cases} \ln bx - 1/(bx + 1) & \text{при } x < 1.3, \\ bx + 1 & \text{при } 1.3 < x < 1.7, \\ \ln bx + 1/(bx + 1) & \text{при } x > 1.7, \end{cases}$$

15. Для $x \in [-1, 1]$, где $a = 2.5$, $b = -0.9$.

$$z = \begin{cases} ax^2 + bx^{2/3} & \text{при } x < 0.1, \\ a\,x^2 & \text{при } x = 0.1, \\ b\,x^{2/3} & \text{при } x > 0.1. \end{cases}$$

16. Ввести координаты точки (x, y). Напечатать, в каком квадранте или на какой оси координат находится эта точка.

17. Ввести радиусы R_1, R_2 и высоту. Вычислить объем усеченного конуса: $h(S_1 + \sqrt{S_1 S_2} + S_2)/3$, где S – площадь оснований. Если $R_1 = R_2$ – объем и площадь цилиндра, если $R_1 = 0$ или $R_2 = 0$ – объем ($h\pi r^2$) и площадь $S = \pi r(\sqrt{r^2 + h^2})$ поверхности конуса.

18. Ввести с клавиатуры цифру. Определить, какой системе счисления она может принадлежать.

19. Ввести число. Определить, делится ли оно нацело на два, три или пять.

20. Ввести a, b, h. Если $h = 0$, вычислить площадь прямоугольника; при $a = b$ найти площадь квадрата; в противном случае подсчитать площадь трапеции.

Задача 2 (программа2_2)

1. Определить остаток от деления на восемь введенного числа x и написать восьмеричную цифру прописью.

2. По цифре, введенной с клавиатуры, напечатать название этой цифры.

3. С клавиатуры ввести число k (1–30). Определить, какому дню недели оно соответствует, если первое число – понедельник.

4. Ввести число и номер месяца. Напечатать дату прописью.

5. Идет k-я секунда суток. Вычислить, сколько прошло часов и полных минут к этому моменту, при этом согласовать со значением слова (час, часа, часов, минута, минуты, минут).

6. В зависимости от номера (N) типа фигуры организовать ввод необходимых данных и вычислить при $N = 1$ – площадь круга, $N = 2$ – объем

шара ($4/3\pi R^3$), $N = 3$ – объем цилиндра, $N = 4$ – площадь поверхности сферы $4\pi r^2$.

7. Ввести число N ($0 \le N \le 15$). Определить и напечатать шестнадцатеричную цифру, ему соответствующую.

8. Для целого числа K (1– 99) напечатать фразу «Мне K лет», при определенных значениях K слово «лет» заменить словом «год» или «года».

9. В зависимости от номера (N) типа фигуры организовать ввод необходимых данных и вычислить при $N = 1$ – площадь прямоугольника, при $N = 2$ – площадь параллелограмма, при $N = 3$ – площадь трапеции $1/2(a + b)h$. В последнем случае напечатать, является ли трапеция параллелограммом или ромбом.

10. Перевести число x ($0 \le x \le 31$) в шестнадцатеричную систему счисления.

11. Напечатать прописью остаток от деления любого целого числа на пять.

12. По введенному номеру напечатать нужный цвет в радуге.

13. Спроектируйте программу, которая показывает, что если сумма цифр двузначного числа кратна трем, то и само число делится на три без остатка.

14. Разработайте программу, которая доказывает, что квадрат двузначного числа $k5$ (последняя цифра – 5) равен $k * (k + 1) * 100 + 25$ (т. е. может быть получен умножением старшей цифры k на следующую по порядку и припиской «25»). Например, 35 * 11 = 3 * 4 * 100 + 25.

Для задания 2 номер варианта определяется остатком от деления индивидуального варианта на число 14.

Задача 3 (программа2_3)

Программу2_1 реализовать через условную операцию (? :), описанную в разделе 1.6.2.2.

1.6.3. Оператор цикла с параметром

Оператор цикла с параметром имеет следующую структуру

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

где <выражение1> – описывает инициализацию цикла и используется для установки начального(ых) значения(й) переменной(ых), управляющей(х) циклом; <выражение2> – определяет условие, при котором оператор цикла необходимо выполнять; <выражение3> – вычисляется после каждой итерации и изменяет переменную цикла на указанное в этом выражении значение.

Проверка условия всегда выполняется в начале цикла. Цикл не выполнится ни разу, если условное выражение сразу ложно, например:

при n = 10; for (i=1; i<n; i=i+2) <оператор>;

Допускается использовать несколько переменных, управляющих циклом. Пример программы для записи чисел в прямом и обратном порядке:

```
#include <stdio.h>                          // результаты:
void main()                                 // i = 0, j = 4
  {
      int i, j;                             // i = 1,  j = 3
      for (i=0, j=4; i<5; i++,  j--)        // i = 2,  j = 2
      printf ("i= %d, j= %d \n", i, j);     // i = 3,  j = 1
}                                           // i=4,  j=0
```

В операторе цикла <выражение1>, <выражение2>, <выражение3> могут отсутствовать одновременно либо одно из них, может отсутствовать сам оператор, выполняемый в цикле:

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

Последняя форма может применяться для организации задержки выполнения программы.

Примеры разработки программ со счетным циклом

Пример 1. Вычислить сумму элементов s = 1 +1/4 +1/9 +1/16 + ... На основе анализа правила изменения параметра для знаменателя определим алгоритм решения задачи (рис. 5) и напишем программу:

```
#include <stdio.h>
#include <conio.h>
void main ()
{
 float s = 0, r;
 int i, N;
printf ("\n Введите N ");
 scanf ("%d", &N);
 for ( i=1; i<=N; i++ )
  {
    r = 1.0/ (i*i);
    s += r; // s = s + r
  }
printf ("Сумма = %6.2f\n ", s);_getch();
}
```

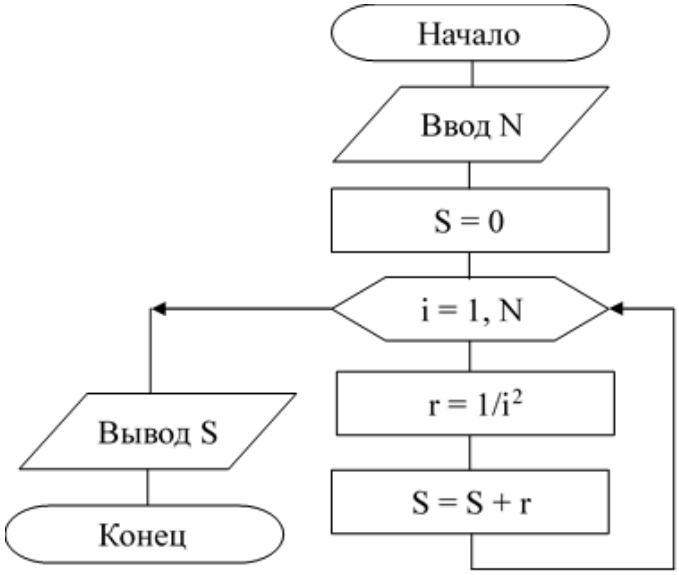


Рис. 5

Пример 2. Вычислить значения двух функций F1(x) = tg(x) и F2(x) = sin(x) в **n** точках, равномерно распределенных на интервале a ≤ x ≤ b, где a = –π/4, b = π. Для реализации первого варианта данной задачи разработана СА (рис. 6) и следующая программа:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#  define b 3.1415
   void main ()
{
```

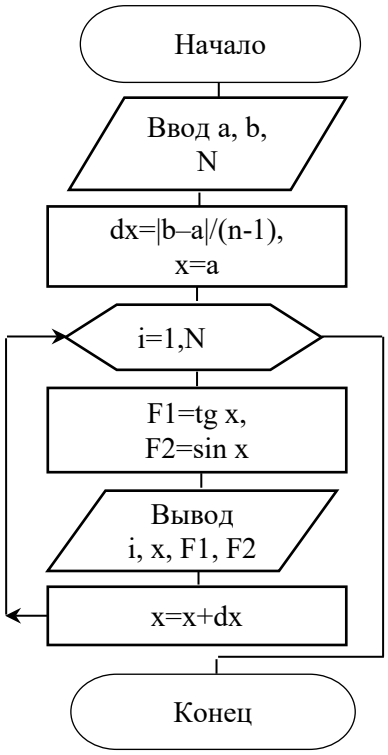


Рис. 6

```
float F1, F2, x, dx, a = - b/4; // dx – шаг изменения int i, n;    // i – переменная цикла (счетчик)
printf ("Введите число точек \n");
scanf ("%d", & n);
dx = fabs(b-a)/(n-1);
x = a;
printf (" _____ \n");
printf ("| I  |      X |     F1 |     F2 | \n");
printf ("|---|--------|--------|--------|\n");
for ( i=1; i<=n; ++i )                    // оператор цикла со счетчиком
 {
    F2 = sin(x);
    F1 = tan(x);                          // вычисление F1 = tg x
    printf (" |%3d| %8.3f |%8.4f| %8.4f|\n", i, x, F1, F2);
    x = x + dx;                           // x += dx
  }
printf (" _____ \n");
_getch();
}
```

Если точки не нумеровать, можно в цикле применить параметр **x** и шаг изменения **dx**. Тогда во втором варианте программы оператор цикла имеет вид: **for (x = a; x < b + dx/2; x+ = dx**) { . . . }

Задание 3 для программирования задач с ветвлениями

Задача 1 (программа3_1)

Для заданных с клавиатуры значений переменных x и n вычислить:

1. $X = 1 + 1/2 + 1/3 + ... + 1/10$.
2. $Z = 2 \cdot 4 \cdot 6 \cdot 8 \cdot ... \cdot 20$.
3. $Y = -x + 4x - 9x + ... - 81x$.
4. $Y = x + x/3 + x/5 + ... + x/17$.
5. $Y = n! = 1 \cdot 2 \cdot 3 \cdot ... \cdot n$.
6. $Y = 1 - 3 + 3^2 - 3^3 + . . . + 3^{10}$.

7. $Y = \sum_{i=5}^{20} x^2/(2i-1)$.

8. $Z = \prod_{1=2}^{10} (x+i)/i$.

9. $Y = \sum_{i=1}^{15} x^2/i$, где Σ – сумма всех элементов; П – произведение всех элементов.

10. $Y = 1 + x/2 + x^2/4 + x^3/6 + ... + x^i/2\text{i} + ... + x^9/18$.

11. $Y = 1 + x^2/1! + x^4/2! + x^6/3! + \ldots + x^{20}/10! = 1 + \sum_{i=1}^{10} x^{2i}/i!$.

12. $Y = 1 - x + x^3/3! - x^5/5! + ... + (-1)^n x^{2n-1}/(2n-1)! + ... + x^{11}/11!$.

13. $e = 1 + 1/1! + 1/2! + ... + 1/n! + ...$ (сравнить результат со значением функции EXP(1), определенной в C/C++).

14. $\pi = 4(1 - 1/3 + 1/5 - 1/7 + \ldots + (-1)^n/(2n+1) + \ldots)$ (результат сравнить с определенным в математике числом π).

15. $Y = x - x^3/3 + x^5/5 - ... + (-1)^n x^{2n+1}/(2n+1) + ...$ ($|x| < 1$). Сравнить результат со значением y = arctg x.

16. $Z = x - x^2/2 + x^3/3 - ... + (-1)^{n-1} x^n/n + ...$ ($|x| < 1$). Сравнить результат со значением $z = LN(1 + x)$.

17. $S = x - x^3/3! + x^5/5! - ... + (-1)^n x^{2n+1}/(2n+1)! + ...$.

18. Вычислить суммы положительных и отрицательных значений функции $z = \cos(nx + a)\ \sin(nx - a)$, где $n = 1, 2, ..., 5$, a и x – переменные плавающего типа.

19. Вычислить сумму четных и сумму нечетных чисел натурального ряда от единицы до числа N.

20. Найти сумму факториала $M = \sum_{i=1}^{n} i\ !$.

Задача 2 (программа3_2)

Вычислить значения двух функций в n точках, равномерно распределенных в диапазоне $a \le x \le b$. Результаты оформить в виде таблицы.

Вариант	a	b	n	$F1(x)$	$F2(x)$
1	0	2π	20	$\sin x \cdot \cos x$	$\sin x + \cos x - 1$
2	1	2	18	$1 + 2^{x+5}$	$(x-1)^3$
3	–1	5	15	$4e^{-\lvert x\rvert} - 1$	$\cos x$
4	–2	5	14	$\lvert x + 10\rvert^5$	$e^{-(x+5)}$
5	0	π	16	$2\sin 2x + 1$	$(x+5)^3 (1 + \sin^2 x)$
6	–π	π	20	$2 - \cos x$	$\sqrt{x+4}$
7	–1	3	20	$2^{-x}/100$	$20/(1 + x^2)$
8	-4	4	12	$x^3 e^{2x}$	$e^x \sin x$
9	1	3	15	$\sqrt{e^x - 1}$	$x \ln^2 x$
10	1	4	20	$1/(1 + \sqrt{x})$	$2^x / (1 - 4^x)$
11	0	2π	20	$5 - 3\cos x$	$\sqrt{1 + \sin^2 x}$
12	–π	π	18	$\lvert\sin x\rvert + \lvert\cos x\rvert$	$\lvert\sin x\rvert - \lvert\cos x\rvert$
13	0	π	16	$e^{-x} + \cos 2x$	e^{-2x}
14	1	2	12	$e^{-x} \lg\sqrt{x+1}$	$x + \sin x$
15	2	4	10	$x \cos x/2$	$\sqrt[3]{x} + \sqrt{2}\ e^{-x}$
16	2	4	16	$2^x \lg x - 3^x \lg x$	$\operatorname{ctg} x$
17	0	5	20	$3^{-x} / 50$	$x\, e^{-x} + \ln x$
18	1	2	18	$e^{2x}\sqrt[3]{x} - \sin x$	$10/(2 + x^2)$
19	3	4	20	$2^x \operatorname{arctg} x - 1$	e^{axx}
20	1	3	20	$\sqrt[5]{x+1}^{\,5}$	$e^{2x} \lg x - 3^{3x}$

Задача 3 (программа3_3)

Модифицировать задачу 2, организовав цикл по параметру x с шагом изменения dx (без нумерации точек).

1.6.4. Оператор цикла с предусловием

Оператор цикла с предусловием предполагает проверку условия повторения тела цикла до него (перед телом цикла) и имеет следующую структуру:

while (<выражение>) <оператор>;

<оператор> может быть либо простым, либо пустым, либо составным. Если <выражение> истинно, то <оператор> выполняется до тех пор, пока <выражение> не станет ложным. Если <выражение> ложно, управление передается оператору, следующему за циклом.

Пример: найти сумму цифр целого числа N. Алгоритм выполнения задачи основан на определении очередной цифры путем вычисления остатка от деления числа, а затем полученных частных до тех пор, пока в частном не получится нуль. Для этого реализована программа в следующем виде:

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

Пример разработки программы с вложенными циклами (циклы с предусловием)

Для трех значений a = 0.1, a = 0.2, a = 0.3 протабулировать функцию y = a·tg(x/4) при изменении аргумента x на интервале [0.5, 0.9] с шагом, равным a. На рис. 7 приведена спроектированная СА и для выделенных в алгоритме действий разработана программа:

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
void main ()
{
float a, x, y;
printf ("-----------------------------\n");
printf ("     a|         x|          y |\n");
printf ("-----------------------------\n");
a = 0.1;
while (a<=0.31)
{
        printf ("%5.2f \n", a);
        x = 0.5;
        while (x<=0.91)
                {
                    Y =  a*tan(x/4);
                    printf ("     %12.2f| %10.2f\n", x, y);
                    x = x + a;                              // x += a;
                }
        printf("-----------------------------\n");
        a = a+0.1;                                          // a+=0.1;
}
_getch();
}
```

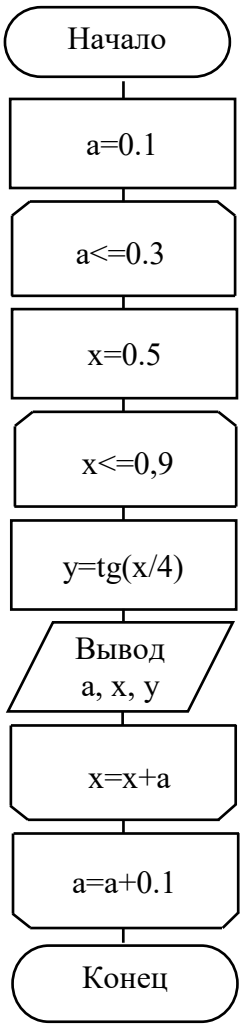


Рис. 7

Второй вариант программы можно реализовать через другие циклы.

Задание 4 для программирования задач с вложенным циклом

Задача 1 (программа4_1)

Начертите схему алгоритма, напишите и отладьте программу для табуляции следующих функций:

1. $S = a\,e^{-x}\sin ax + \sqrt{ay}$ при $-1 \le x \le 1$ с шагом 0.2,
   $a = 0.75$ $1 \le y \le 5$ с шагом 1.5.

2. $Z = \sqrt[5]{a \cdot x \cdot y^2} + 1.3\sin(x - a)$ при $2 \le x \le 5$ с шагом 0.5,
   $a = 1.9$ $-1 \le y \le 1$ с шагом 0.5.

3. $S = \sqrt{t+1}\,e^{2ty}\cos(t - a)$ при $1 \le t \le 2$ с шагом 0.2,
   $a = -2.1$ $2 \le y \le 3$ с шагом 0.3.

4. $Z = bx\sqrt{t+b}\,(tx + 2.1)$ при $1 \le x \le 2$ с шагом 0.2,
   $b = 3.5$ $0 \le t \le 1$ с шагом 0.2.

5. $$Z = \begin{cases} x^2 e^{-x^2/2}, & \text{если } 0 \le x \le 2, \\ e^{x^2/a} - 1, & \text{если } 2 < x \le 3.6, \end{cases}$$ $a = 0.50;\ 0.75$, шаг $dx = a/2$.

6. $$Y = \begin{cases} \sin ax - \sqrt{\lg(a^2 + x^2)}, & \text{если } 0.1 \le x \le 0.4, \\ a\cos^2(ax), & \text{если } 0.4 < x \le 1.2, \\ 2 - \sin ax, & \text{если } 1.2 < x \le 1.6, \end{cases}$$ $a = 1.0;\ 1.5$, шаг $dx = a/5$.

7. $$Z = \begin{cases} \ln(a + x^2/\sqrt{a}), & \text{если } 0 \le x \le 2, \\ 2e^{2x}, & \text{если } 2 < x \le 3.6, \end{cases}$$ $a = 1.0;\ 1.3;\ 1.6$, шаг $dx = a/4$.

8. $$Z = \begin{cases} a(e^{x+2a} + e^{-(x-3a)}, & \text{если } 0.1 \le x < 0.5, \\ \sin x, & \text{если } x = 0.5, \\ a + a\cos(x + 3a), & \text{если } 0.5 < x \le 1.5, \end{cases}$$ $a = 2;\ 2.1$, шаг $dx = a/10$.

9. $Z = \begin{cases} -ae^{x-3a}, & \text{если } 0 < x \le 3, \\ -a(1 + \ln(x + 3a)), & \text{если } 3 < x \le 4, \end{cases}$ | $a = 1; 1.5,$ шаг $dx = a/2.$

10. $Y = \begin{cases} -(x + 3a)^2 - 2a, & \text{если } -1 \le x < 0, \\ a\cos(x + 3a) - 3a, & \text{если } 0 < x < 1, \\ a\,e^x, & \text{если } x = 1, \end{cases}$ | $a = 0.7; 1,$ $dx = (a + 0.2)/10.$

11. $Z = \begin{cases} a(x - a)^{3/2}, & \text{если } 1 \le x \le 2.5, \\ a/2(e^{x/a}+e^{-x/a}), & \text{если } 2.5 < x \le 4, \end{cases}$ | $a = 0.5; 1.0,$ шаг $dx = a/2.$

12. $Y = \begin{cases} x^{a\cdot x^2+1}, & \text{если } 0.5 \le x < 1.5, \\ a\cos x, & \text{если } x = 1.5, \\ (a\,x^2 + 1)^x, & \text{если } 1.5 < x \le 3, \end{cases}$ | $a = 0.1; 0.2; 0.3,$ шаг $dx = 2a.$

13. $Z = \begin{cases} (\sin^2 x + a)^2\, e^{a \sin x}, & \text{если } 0.1 \le x \le 0.5, \\ \operatorname{tg}(x/4), & \text{если } 0.5 < x \le 0.9, \end{cases}$ | $a = 0.1; 0.2; 0.3,$ шаг $dx = a.$

14. $Y = \begin{cases} \operatorname{tg}(a^2 + \sin ax), & \text{если } 0 \le x \le 1, \\ a\sin(a - \cos ax), & \text{если } 1 < x \le 2, \\ \lg x, & \text{если } 2 < x < 3, \end{cases}$ | $a = 0.5; 0.75; 1,$ шаг $dx = a/4.$

15. $Z = \begin{cases} e^{\cos x} - a\sin^2(ax), & \text{если } 0.5 \le x \le 1.5 \\ a\,x^2 - \cos ax, & \text{если } 1.5 < x <= 2 \end{cases}$ | $a = 0.1; 0.7; 1.3,$ шаг $dx = a/4.$

16. $Y = \begin{cases} 1/x, & \text{если } 0.1 \le x \le 0.4 \\ \ln(x^2 + ax), & \text{если } 0.4 < x \le 1.2 \\ x^2, & \text{если } 1.2 < x < 1.6 \end{cases}$ | $a = 1.0; 1.5,$ шаг $dx = a/5.$

17. Дана непустая последовательность различных натуральных чисел, за которой следует нуль (признак конца последовательности). Определить порядковый номер наименьшего из них и его значение.

18. Найти первый отрицательный элемент последовательности: cos (ctg (*n*)), где *n* = 1,2,3,... . Напечатать его номер и значение. Распечатать всю последовательность.

19. Вычислить наибольший общий делитель (*k*) натуральных чисел *d* и *f*. Оператор FOR не использовать.

20. Дан ряд неотрицательных вещественных чисел. Определить, сколько из них больше своих «соседей»: предыдущего и последующего чисел. Признаком окончания ряда чисел считать появление отрицательного числа.

Задача 2 (программа4_2)

Модифицировать (изменить) программу 3_2 для вычисления функций F1(x) и F2(x) с применением вместо счетного цикла оператора цикла с предусловием. Выполнить ее и сравнить результаты с полученными в предыдущей работе.

Задача 3 (программа4_3)

Изменить программу4_1: вместо цикла с предусловием использовать цикл с параметром.

1.6.5. Оператор цикла с постусловием

Оператор цикла с постусловием применяется в тех случаях, когда тело цикла должно выполниться хотя бы один раз, условие повторения многократно выполняющихся действий записывается после тела цикла и имеет следующий формат:

do { <операторы> } while(<выражение>);

Сначала выполняются <операторы>, затем проверяется истинность <выражения>: если оно истинно (!=0), то <операторы> вновь выполняются и вычисляется значение <выражения>. Если <выражение> ложно, то управление передается следующему за **do-while** оператору.

Данный оператор применяется часто в итерационных алгоритмах. **В итерационных алгоритмах** повторение операторов тела цикла заканчивается при выполнении условия, связанного с проверкой монотонно изменяющейся величины.

Примеры разработки программы с циклом с постусловием

Пример 1. Вычислить 15 значений функций y1(x) = tg(x) и y2(x) = ctg(x) при a ≤ x ≤ b, a = 0.6*Pi, b = 0.7*Pi.

Для вычислений разработана СА (рис. 8) и следующая программа:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#define  PI  3.1413   // директива постановки константы
void main()
{
int n=15;
float a=0.6* PI, b=0.7* PI;
float x, y1, y2, dx;
dx = fabs((a-b)/(n-1));
x = a;
do
{
Y1= tan(x);
y2 = 1/y1;
printf ("x= %6.4f y1= %7.4f y2= %7.4f\n", x, y1,y2);
x = x + dx;
}
while (x<b +dx/2);
_getch();
}
```

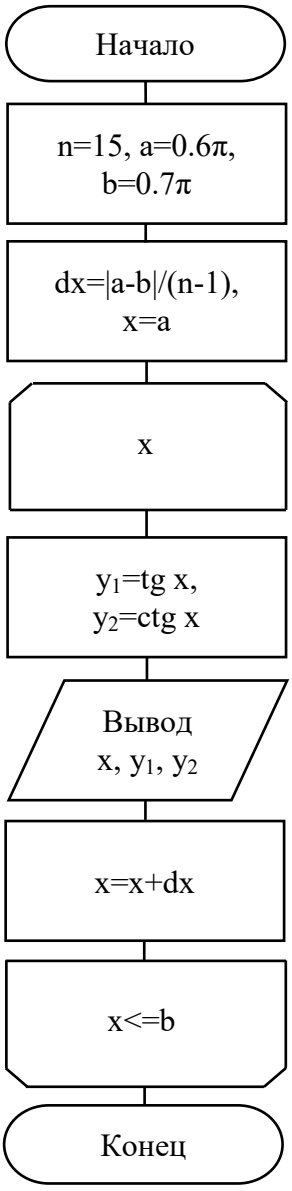


Рис. 8

Пример 2. Вычислить сумму элементов s = 1+1/4+1/9+1/16+... с точностью до E = 0.00001. Для решения данной задачи разработаем итерационный алгоритм, схема которого приведена на рис. 9. В соответствии с алгоритмом подготовлена и отлажена нижеприведенная программа.

```
#include <stdio.h>
#include <conio.h>
void main ()
{
float E=0.00001;
float s, r;          // s – сумма, r – слагаемое
int i;               // счетчик слагаемых
s = 0;
i = 1;
do
{
       r = 1.0/(i*i);       // вычисление слагаемых
       s = s + r;       //  s+ = r;
       i++ ;            //  i = i +1;
}
while (r>E);         // сравнение слагаемых с E
printf ("Сумма=%9.5f\n",s);  // вывод суммы
_getch();
}
```

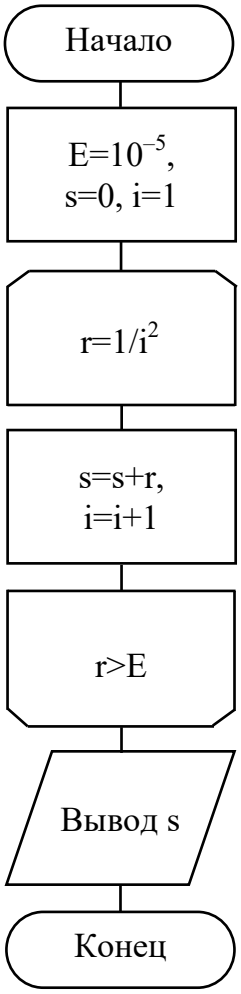


Рис. 9

Задание 5 для программирования задач с циклами с предусловием и итерационными

Задача 1 (программа5_1)

Модифицировать программу4_2 для вычисления функций *F*1(*x*) и *F*2(*x*) с применением оператора цикла с постусловием. Выполнить ее и сравнить результаты с полученными ранее.

Задача 2 (программа5_2)

Спроектировать схему алгоритма, написать и отладить программу для одной из следующих задач.

1. Вычислить приближенное значение $z = \text{arctg}\, x$ и сравнить с $Z = x - x^3/3 + x^5/5 - \ldots + (-1)^n x^{2n+1}/(2n+1) + \ldots$ ($|x| < 1$), прекращая вычисления, когда очередной член по абсолютной величине будет меньше eps = 0.00001.

2. Вычислить $y = x - x^2/2 + x^3/3 - \ldots + (-1)^{n-1} x^n/n + \ldots$ с точностью eps = 0.00001, где $|x| < 1$. Сравнить результат с вычисленным через стандартную функцию значением $y = \text{LN}(1 + x)$.

3. Вычислить при $y = 1 + x/1! + x^2/2! + \ldots + x^n/n! + \ldots$ с точностью eps = 0.00001 и сравнить результат с вычисленным через стандартную функцию значением $y = \text{EXP}(x)$.

4. Вычислить $Y = x - x^3/3! + x^5/5! - \ldots + (-1)^n x^{2n+1}/(2n+1)! + \ldots$ с точностью eps = 0.00001 и сравнить с $y = \sin x$.

5. Вычислить $y = 1 - x^2/2! + x^4/4! - \ldots + (-1)^n x^{2n}/(2n)! + \ldots$ с точностью eps = 0.0001 и сравнить результат с вычисленным через стандартную функцию значением $y = \cos(x)$.

6. Найти произведение цифр заданного натурального числа.

7. Определить число, получаемое выписыванием в обратном порядке цифр заданного натурального числа.

8. Определить номер первого из чисел $\sin x$, $\sin(\sin x)$, $\sin(\sin(\sin x))$, ... , меньшего по модулю 10^{-3}.

9. Дана непустая последовательность различных целых чисел, за которой следует нуль. Определить порядковый номер и величину наибольшего среди отрицательных чисел этой последовательности.

10. Вычислять периметры и площади прямоугольных треугольников по длинам катетов, пока один из заданных катетов не окажется нулевым.

11. Дана непустая последовательность положительных целых чисел, за которой следует отрицательное число (это признак конца последовательности). Вычислить среднее геометрическое этих чисел.

12. Дана непустая последовательность ненулевых целых чисел, за которой следует нуль. Определить, сколько раз в этой последовательности меняется знак.

13. Числа Фибоначчи (f_n) определяются формулами:

$f_0 = f_1 = 1$; $f_n = f_{n-1} + f_{n-2}$ при $n = 2, 3, \ldots$.

Вычислить сумму всех чисел Фибоначчи, которые не превосходят 1000.

14. Дана непустая последовательность положительных вещественных чисел x_1, x_2, x_3,..., за которыми следует отрицательное число. Вычислить величину $x_1 + 2x_2 + \ldots + (N-1)\,x_{N-1} + N x_N$, где N заранее не известно.

15. Вычислить длины окружностей, площади кругов и объемы шаров для ряда заданных радиусов. Признаком окончания счета является нулевое значение радиуса.

16. Определить, есть ли одинаковые цифры в заданном числе.

17. Определить, является ли заданное натуральное число палиндромом, т. е. таким, десятичная запись которого читается одинаково слева направо и справа налево.

18. Вычислить наименьшее общее кратное натуральных чисел a и b.

19. Дано число L. Определить первый отрицательный член последовательности x_1, x_2, x_3,..., где $x_1 = L$, $x_i = \mathrm{tg}(x_{i-1})$.

20. Определить, является ли заданное натуральное число совершенным, т. е. равным сумме всех своих (положительных) делителей, кроме самого этого числа (например, совершенное число 6 = 1 + 2 + 3).

Задача 3 (программа5_3)

Модифицировать программу4_1 с использованием оператора цикла с постусловием do … while и сравнить с полученными результатами в задании 4.

1.6.6. Другие операторы

1.6.6.1. Составной оператор

Составной оператор объединяет записанные в фигурных скобках операторы в единый блок, синтаксически эквивалентен одному оператору и применяется там, где согласно правилам языка допустим один оператор. Он имеет следующий формат:

{
[объявления;] // описания переменных и задание начальных значений
оператор1 ;[оператор2;] ...
}

Все объявления, включенные в блок, должны быть в начале. Действие составного оператора заключается в последовательном выполнении действий (операторов) в фигурных скобках. Основное назначение – группировать операторы в блок.

1.6.6.2. Оператор передачи управления goto

Оператор безусловной передачи управления имеет следующую структуру:

goto <метка>; ...

и по тексту программы должен быть оператор с такой меткой:

<метка> : <оператор>

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

Современная технология программирования основана на программировании без **goto**, применение этого оператора делает программу запутанной и сложной в отладке. Однако в некоторых случаях без оператора **goto** не обойтись. Обычно он используется, если необходимо выйти из вложенных управляющих структур (из двух или более циклов).

1.6.6.3. Пустой оператор

Пустой оператор состоит из точки с запятой и никаких действий не выполняет. Обычно используется в операторах if, for, while, do-while, когда тело оператора отсутствует, хотя по синтаксису оператор необходим, а также если требуется пометить фигурную скобку меткой (синтаксис языка C/C++ требует, чтобы после метки обязательно следовал оператор, фигурная скобка же оператором не является). Например: с помощью оператора **for** и пустого оператора может быть организована **задержка** выполнения программы: **for** (i=0; i<100000; i++) ;
применяется для того, чтобы некоторое время не закрывалось окно результатов.

1.6.6.4. Оператор выхода break; (разрыв)

Оператор выхода break; (разрыв) обеспечивает прекращение выполнения самого внутреннего из объемлющих его операторов цикла for, do-while, while или оператора switch. После выполнения оператора break управление передается оператору, следующему за прерванным.

Одно из назначений оператора – закончить выполнение цикла при присваивании некоторой переменной определенного значения. Пример см. в разд. 1.6.2.3.

1.6.6.5. Оператор возврата return

Завершает выполнение текущей функции и возвращает управление в вызывающую функцию в точку, непосредственно следующую за вызовом. Ее формат: **return[<выражение>] ;**

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

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

2. ОБРАБОТКА МАССИВОВ ДАННЫХ

Массив – упорядоченная группа однотипных данных, обозначенная одним именем.

Массивы относятся к блокам переменных наряду со структурами и объединениями.

Массивы состоят из компонентов одинакового типа, а структуры и объединения включают в общем случае различные типы компонентов.

Компоненты массива называются элементами, а компоненты структуры или объединения − полями. Поскольку компонентами блоков могут быть не только простые переменные, но также структуры и объединения, возможно описание блоков данных большой сложности.

2.1. ОДНОМЕРНЫЕ МАССИВЫ ОДНОТИПНЫХ ДАННЫХ

При описании одномерного массива задается тип его элементов, имя и указывается число элементов (размер) в квадратных скобках:

<класс> <тип> <идентификатор>[конст. выраж1];

Индексирование элементов в языке C/C++ начинается с нуля.

При обращении к элементу массива указывается номер элемента или константное выражение, определяющее элемент:

<идентификатор>[конст.выраж1][конст.выраж2]... ,

например, vec[0], vec[2] //line[i+j-1], matr[i][j]

Константное выражение при объявлении массива может быть опущено:

1) если массив объявлен как параметр в функциях;

2) если при объявлении массив инициализируется, например:

char string[] = "hello" //содержит 6 символов (это описание эквивалентно char string[]={'h','e','l','l','o','\0'});

3) если массив объявлен как ссылка на массив, явно определенный в другом файле (extern).

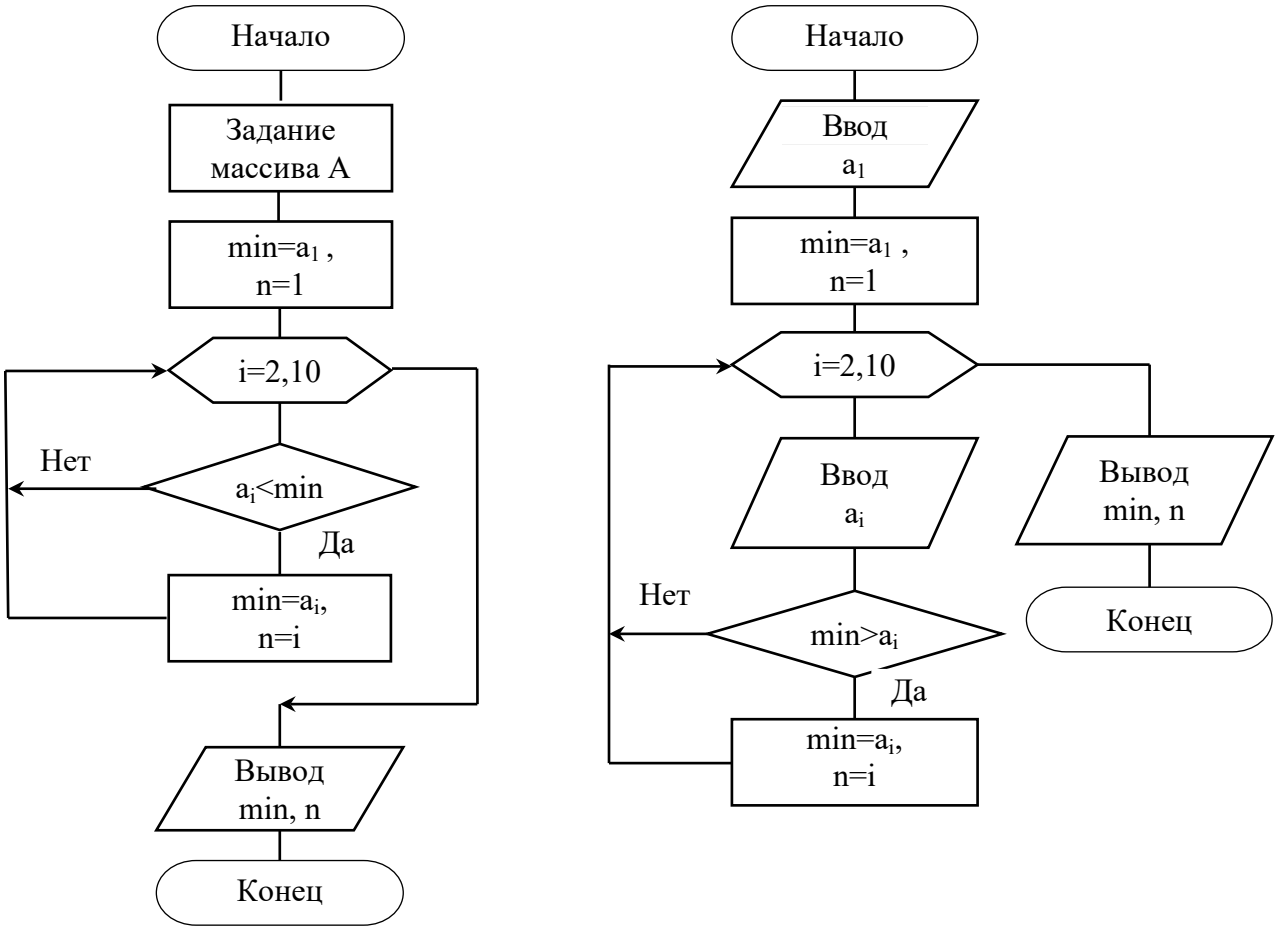
Начальные значения массивам могут быть присвоены при компиляции, если они объявлены классом памяти static или являются внешними.

При инициализации список элементов массива должен быть заключен в фигурные скобки. Если размер опущен, то он определяется по числу начальных значений:

```
int a[]={1,4,3,65};          // всего 4 значения: a[0]...a[3].
```

Пример программирования алгоритмов обработки одномерных массивов

Найти значение и номер наименьшего элемента в одномерном массиве. При задании элементов исходного массива константой спроектированная СА решения задачи имеет вид, представленный на рис.10, *а*. В соответствии с данной схемой разработана нижеприведенная программа.



а *б*

Рис. 10

```
#include <conio.h>
#include <stdio.h>
void main ()
{
        int a[10] = {5,2,7,1,8,3,4,5,6,6};
        int i, min, n;
min = a[0];
n = 0;
for (i=1; i<10; i++)
{
        if (a[i] < min)
    {   min = a[i];
        n = i+1;
    }
}
printf ("минимальный элемент %d-й равен %d\n ", n, min);
_getch();
}
```

Если элементы массива вводятся с клавиатуры, то СА имеет вид, представленный на рис. 10, *б*, и текст программы изменится:

```
#include <stdio.h>
#include <conio.h>
void main ()
{
 int a[10];
 int i, min, n;
 printf("Введите элементы массива: ");
 scanf ("%d", &a[0]);
 min = a[0];
 n = 1;
 for (i=1; i<10; i++)
```

```
   {
      scanf ("%d", &a[i]);
      if (a[i] < min)
      {      min = a[i];
             n = i +1;
      }
   }
printf ("минимальный элемент %d  равен n= %d\n ", n, min);
getch();
}
```

Задание 6 для программирования задач с одномерными массивами

Задача 1

Начертить схему алгоритма, написать и отладить программу для одной из следующих задач. В программе 6_1 исходные массивы задать типизированными константами. В программе 6_2 элементы исходных массивов ввести через функцию ввода.

1. Дан массив из N чисел ($N < 12$). Вычислить среднее геометрическое значение.

2. Дан массив из N чисел ($10 < N < 15$). Найти максимальное значение.

3. Дано 20 чисел. Найти их среднее арифметические значение.

4. Дано 12 вещественных чисел. Найти порядковый номер того из них, которое наиболее близко к какому-нибудь заданному целому числу X.

5. Дана последовательность из 15 целых чисел. Определить количество отрицательных чисел в ней и максимальное число подряд следующих отрицательных чисел.

6. Дано 15 целых чисел. Найти наибольшее из них. Определить, сколько из чисел принимает наибольшее значение.

7. Дано целое $n > 1$ и вещественные числа x_1, x_2, ..., x_n. Вычислить математическое ожидание и дисперсию по формулам:

$$M = \sum_{i=1}^{n} x_i / n, \qquad D = \sqrt{\sum_{i=1}^{n} (x_i - M)^2 / (N-1)}.$$

8. Дан массив из N чисел ($8 < N < 12$). Вычислить сумму элементов с нечетными индексами и их среднее арифметические значение.

9. Даны два одномерных массива A и B. Вычислить элементы массива C по правилу: если a_i и b_i различны, то c_i присвоить их сумму, при одинаковых a_i, b_i в c_i переписать соответствующий элемент массива A.

10. Дано 10 вещественных чисел. Вычислить разность между максимальным и минимальным из них.

11. Дано 10 вещественных чисел. Определить, образуют ли они возрастающую последовательность.

12. Дан массив X из n чисел ($6 < n < 10$). Вычислить:

$y = x_1 - x_2 + x_3 - ... - x_{n-1} + x_n$.

13. Дано 18 чисел. Определить количество элементов, отличных от последнего числа.

14. Дано 12 чисел. Напечатать сначала все отрицательные, а затем все остальные.

15. Сформировать одномерный массив из 15 простых чисел.

16. Дано восемь натуральных чисел. Найти их наибольший общий делитель.

17. Дана последовательность натуральных чисел. Вычислить сумму тех из них, порядковые номера (индексы) которых – простые числа.

18. Дан массив из 20 натуральных чисел. Вычислить сумму тех из них, порядковые номера которых – числа Фибоначчи, определяемые формулами $f_0 = f_1 = 1; f_n = f_{n-1} + f_{n-2}$ при $n = 1, 2, 3, ...$.

19. Дан массив X из n чисел. Вычислить значение функции

$y = x_n(x_n + x_{n-1})(x_n + x_{n-1} + x_{n-2})...(x_n + ... + x_1)$.

20. Дано 24 целых числа. Распечатать их в обратном порядке по шесть чисел в строке.

Задача 2 (программа6_3)

Модифицировать программу 3_2 для функций $F1(x)$ и $F2(x)$ таким образом, чтобы результаты были сформированы в виде трех одномерных массивов. Выполнить ее и сравнить результаты с полученными в заданиях 3 и 4.

2.2. Многомерные массивы данных

Массив можно представить последовательностью данных, а можно как таблицу, данные которой распределены по строкам и столбцам. В двумерном массиве (матрице) одна размерность соответствует строкам, а вторая столбцам. При объявлении массива каждая размерность представляет собой дополнительный индекс.

Описание многомерного массива:

<класс><тип><идентификатор>[конст. выраж1][конст. выраж2]…,
<идентификатор>[конст. выраж1][конст. выраж2]…;

квадратные скобки здесь обязательны, константное выражение в квадратных скобках задает количество элементов в массиве (размерность) по каждому измерению. Число измерений определяется числом квадратных скобок.

Если компоненты блока (массива), которому присваиваются начальные значения, также представляют собой блок (массив), то его элементы в свою очередь рекомендуется заключать в фигурные скобки подсписка:

```
mass[2][3]={{1,2,3},{4,5,6}};          // 1 2 3 – 1-я строка
                                       // 4 5 6 – 2-я строка
```

эквивалентная запись: mass[2][3]={1,2,3,4,5,6};

Записи ... ARR[3][2]={{1,2},{3}}

эквивалентна ARR[3][2]={1,2,3},

но не эквивалентна CARR[3][2]={{1},{2},{3}}

Пример разработки программы с двумерными массивами

Пример. Вычислить суммы элементов в столбцах двумерного числового массива A[3][4] размером 3×4 элемента.

Текст программы для СА (рис. 11):

```
#include <stdio.h>
#include <conio.h>
    void main ()
{
int a [3][4] = {{1,2,3,4},{1,2,3,4},{1,2,3,4}};
int s[4];      // массив сумм
int i, j;   clrscr();
for ( j=0; j<4; ++j)
{
    s[j] = 0;
    for ( i=0; i<3; ++i )
            s[j] = s[j] + a[i][j];
    printf (" s[%d] = %d  ",  j, s[j]);
}
_getch();
}
```

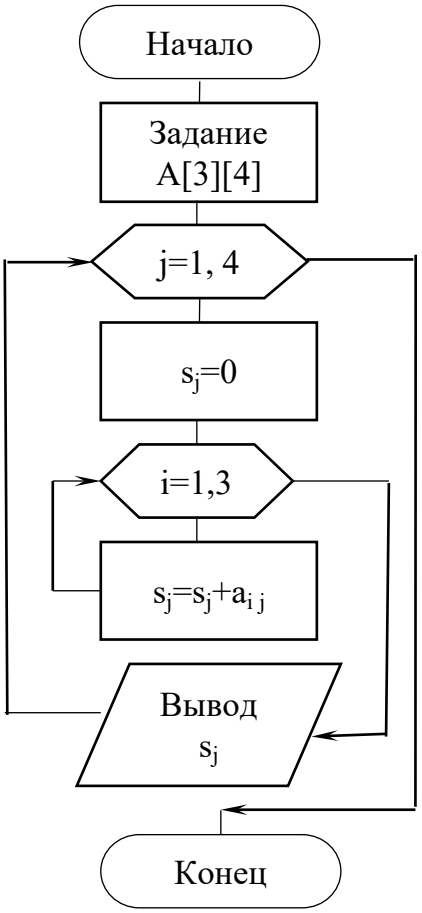


Рис. 11

Задание 7 для программирования задач с двумерными массивами

Задача 1 (программа7_1)

1. Даны матрицы A[4][4], B[4][4] вещественных чисел. Вычислить матрицу C поэлементным сложением соответствующих элементов матриц A и B.

2. Дана матрица B[5][5] вещественных чисел. Найти для нее транспонированную матрицу.

3. Даны матрица A[4][4] вещественных чисел и константа K. Вычислить матрицу C = A*K.

4. Сформировать массив по правилу

$$\begin{vmatrix} 1 & 0 & 0 & \dots & 0 \\ 2 & 1 & 0 & \dots & 0 \\ 3 & 2 & 1 & \dots & 0 \\ & & \dots & & \\ 10 & 9 & 8 & \dots & 1 \end{vmatrix}$$

5. Даны натуральное *N* и элементы квадратной вещественной матрицы A пятого порядка. Вычислить *N*-ю степень каждого элемента этой матрицы

($a_{ij}^1 = a_{ij}$, $a_{ij}^2 = a_{ij} * a_{ij}$, $a_{ij}^3 = a_{ij}^2 * a_{ij}$ и т.д., где i, j = 1,2,...,5).

6. Сформировать массив по правилу

$$\begin{vmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 2 & \dots & 0 \\ & & \dots & & \\ 0 & 0 & 0 & \dots & 9 \end{vmatrix}$$

7. Сформировать массив последовательностью натуральных чисел:

$$\begin{vmatrix} 1 & 2 & \dots & 10 \\ 11 & 12 & \dots & 20 \\ 21 & 22 & \dots & 30 \\ & & \dots & \\ 91 & 91 & \dots & 100 \end{vmatrix}$$

8. Сформировать двумерный массив

$$\begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 1 & 2 & 3 & 4 \\ 4 & 5 & 1 & 2 & 3 \\ 3 & 4 & 5 & 1 & 2 \\ 2 & 3 & 4 & 5 & 1 \end{vmatrix}$$

9. Дана матрица A[N][N] целых чисел. Сформировать вектор *B* из максимальных элементов каждой строки.

10. Дана матрица A[6][6] целых чисел и переменная *X*. Элементы нечетных строк матрицы A заменить на *X*.

11. Дана матрица B[5][5]. Получить массив C удалением (либо обнулением) *n*-й строки и *k*-го столбца из матрицы B.

12. Определить, является ли заданная квадратная матрица девятого порядка магическим квадратом, т. е. такой, в которой суммы элементов во всех строках и столбцах одинаковы.

13. Определить, является ли заданная целая квадратная матрица шестого порядка симметричной (относительно главной диагонали).

14. Дана вещественная матрица A[7][4]. Переставляя ее строки и столбцы, добиться того, чтобы наибольший элемент оказался в верхнем левом углу.

15. Дана вещественная матрица размером 4*8. Упорядочить ее строки по возрастанию сумм их элементов.

16. Для заданной целой матрицы размером 6*8 элементов напечатать индексы всех ее седловых точек. Элемент матрицы называется седловой точкой, если он является наименьшим в своей строке и одновременно наибольшим в своем столбце или, наоборот, является наибольшим в своей строке и наименьшим в своем столбце.

17. Дана матрица A[6][2] вещественных чисел. Рассматривая A[i][1] и A[i][2] как координаты точек на плоскости, определить радиус наименьшего круга (с центром в начале координат), внутрь которого попадают все эти точки.

18. Дан массив F[N][M] из целых малых величин, определить количество «особых» элементов в нем. Элемент считается «особым», если он больше суммы остальных элементов своего столбца.

19. Дана матрица D[5][6] из целых чисел. Упорядочить элементы строк в нем по убыванию.

20. Дан массив *M* координат точек на плоскости. Найти наибольшее расстояние между этими точками.

Задача 2 (программа7_2)

Сформировать таблицу Пифагора.

Задача 3 (программа7_3)

Модифицировать программу6_3 таким образом, чтобы значения *X*, *F*1 и *F*2 были представлены двумерным массивом, состоящим из трех строк.

3. ПРОЕКТИРОВАНИЕ СЛОЖНЫХ ПРОГРАММ

3.1. Подпрограммы

3.1.1. Общие правила работы с функциями

Подпрограммы – относительно самостоятельные фрагменты программы, оформленные определенным образом и снабженные заголовком. С помощью подпрограмм любая программа может быть разбита на ряд формально независимых друг от друга частей. В C/C++ подпрограммы оформляются в виде функций.

Подпрограммы применяют при модульном проектировании программ (восходящим и нисходящим методами разработки программ) или когда необходимо некоторую последовательность действий выполнить на разных участках программы неоднократно, возможно для разных данных (например, для трех массивов вычислить среднее арифметическое значение).

Любые идентификаторы переменных, констант, типов, описанные внутри подпрограммы, локализуются только в ней и называются локальными для данного блока (подпрограммы). Такой блок (подпрограмма) называется областью действия этих локальных имен. Локальные имена не являются формальными параметрами.

Переменные, описанные до всех функций, называются глобальными. Область действия глобальных переменных – все функции, описанные ниже.

Все имена в пределах подпрограммы должны быть уникальными и не могут совпадать с именем самой функции.

Рекомендуется описывать имена в том блоке, где они используются, если это возможно. Если *переменная*, используемая в функции, должна сохранять свое значение до следующего вызова функции, то она описывается классом памяти **static**.

Любая программа на C/C++ состоит из одной или нескольких функций. Каждая функция должна иметь имя, используемое для вызова функции.

Имя одной из функций – main обязательно должно присутствовать в каждой программе (оно зарезервировано). Функция main не обязательно должна быть первой, хотя с нее всегда начинается выполнение программы.

Схема передачи параметров из основной программы в вызываемую функцию (подпрограмму): при обращении к подпрограмме фактический параметр заменяет формальный и выполнение функции происходит для фактического параметра. Оператор return возвращает значение, указанное в нем, в главную программу.

С использованием подпрограмм в языке C/C++ связаны три понятия:

1) [объявление функций;]

2) определение функций;

3) вызов функций.

В программе объявление функции может отсутствовать (поэтому в квадратных скобках), но в этом случае определение функции должно предшествовать вызову.

Определение функции присутствует в любом случае. В нем указывается идентификатор (имя) функции, типы и список ее формальных параметров, описания и операторы, которые определяют действия самой функции.

В определении задается тип значения, возвращаемого функцией (тип возврата) и может быть задан класс памяти.

Структура заголовка определения функции:

[класс памяти] тип [*] <имя>([<список формальных параметров>]),

где <список формальных параметров> включает последовательность описаний (объявлений) параметров, разделенных запятыми, которые требуется передать в функцию при ее вызове; [класс памяти] – необязательный параметр, он задает область видимости функции (если он не задан, то по умолчанию считается внешним). Формальные параметры – это переменные, которые будут принимать конкретные значения, передаваемые функции во время вызова.

Определение функции содержит кроме заголовка тело функции, представляющее собой последовательность операторов и описаний в фигурных скобках.

Элемент <списка формальных параметров> имеет формат:

[register] <тип> <имя параметра>, например: int Func(int par1, int par2).

[register] – необязательный параметр, определяющий класс памяти формального параметра. Спецификатор <тип> задает тип параметра. Он может быть опущен, если параметр имеет тип int.

Если список формальных параметров заканчивается запятой с многоточием:

(<параметр1>,<параметр2>,<параметр3>, ...) , то

предполагается использование *функции с переменным числом аргументов*.

Если функции не передаются аргументы, то в скобках вместо списка формальных параметров указывается ключевое слово void.

Параметр <тип> задает тип возвращаемого функцией значения. Это может быть любой основной тип (целый, плавающий) либо тип структуры или объединения; <имя> – имя функции, может быть задано со звездочкой <*имя>. Это означает, что функция возвращает указатель (переменная или выражение, определяющее адрес, по которому помещено значение). Функции не могут возвращать массив или функцию, но могут возвращать указатель на любой тип.

Тип возвращаемого значения, задаваемый в определении функции, должен соответствовать типу в объявлениях этой функции, сделанных где-либо в программе.

Если функция заканчивается оператором *return*<*выражение*>в теле функции, то функция возвращает значение. Если *return* отсутствует или в нем не содержится выражения, то функция ничего не возвращает. В этом случае тип функции должен быть помечен ключевым словом void, означающим отсутствие возвращаемого значения.

Если функция объявлена со спецификацией типа возвращаемого значения, а фактически значение не возвращается, то поведение программы после возврата управления из такой функции может быть непредсказуемым.

Объявление функции (прототип) необходимо, когда вызов функции предшествует ее определению и задает ее имя, тип возвращаемого значения и список типов передаваемых параметров:

[класс] тип имя ([список типов параметров]);

Порядок и типы формальных параметров в определении функции и во всех ее объявлениях (если они имеются) должны совпадать. Типы фактических параметров в вызовах функции должны быть совместимы с типами соответствующих формальных параметров. Формальный параметр может быть любого основного типа, структурой, объединением, перечислением, указателем или массивом.

Если необходимо, компилятор выполняет арифметические преобразования для каждого формального параметра и каждого фактического аргумента независимо.

Пример определения функции:

```
float SUMMA(int A, char B, char C)
{return A+B+C;}
```

Описание параметров может находиться после списка параметров (классическое описание), но до тела функции, например:

```
float SUMMA(A, B, C)
int A, char B, C;
{return A+B+C;}
```

Тело функции – это составной оператор, который содержит операторы, определяющие действия функции. Он может включать объявления переменных, используемых во вложенных операторах (помимо объявленных формальных параметров).

Все переменные, объявленные в теле функции, имеют класс auto, если они не объявлены иначе и являются локальными.

Фактические параметры функции передаются только по значению и рассматриваются как локальные переменные, место для которых распределяется при вызове функции, они инициализируются значениями переданных аргументов и теряются при выходе из функции. (Поэтому изменить значение фактического параметра в вызывающей функции нельзя).

Вызов функций производится указанием имени или имени со списком фактических параметров. Фактические параметры не указываются, если в функции используются глобальные переменные (помимо локальных) и в заголовке функции (подпрограммы) отсутствует список формальных параметров.

Если заголовок функции содержит формальные параметры, то при обращении к функции должен обязательно присутствовать список фактических переменных (через запятую). Количество, тип и порядок перечисления фактических параметров должны обязательно соответствовать количеству, типу и порядку перечисления формальных параметров в определении подпрограммы.

Формат вызова функции следующий:

<имя_функции>(<список фактических аргументов>)

или

<выражение>(<список выражений>);

где <выражение> вычисляется и вызывается из него или вычисляется как адрес функции;

<список выражений> – список фактических аргументов, каждый из которых может вычисляться. Список при вызове может быть пустым, если формальные параметры в заголовке функции отсутствуют.

Фактические аргументы могут быть любой величиной основного типа, структурой, перечислением, объединением или указателем.

Указатель на функцию может быть передан в качестве параметра функции.

Применяется следующая схема вызова функции:

1) вычисляются выражения в списке выражений. Если известен прототип функции, то типы аргументов сравниваются с соответствующими типами формальных параметров. Если они не совпадают, производится *преобразование типов* либо выдается диагностическое сообщение. Если в прототипе вместо списка формальных параметров задано ключевое слово (void), то параметры в функцию не передаются;

2) происходит замена формальных параметров на фактические;

3) передается управление на первый оператор функции, и далее выполняются необходимые по алгоритму последующие операторы тела функции;

4) выполнение оператора return в теле функции возвращает управление и, возможно, значение в вызывающую функцию. Если оператор return не задан, то управление в вызывающую функцию передается после последнего выполняемого в ее теле, при этом возвращаемое значение не определено.

Злоупотребление глобальными переменными делает программу сложной в отладке. В связи с этим рекомендуется там, где это возможно, передавать результаты через параметры и указатели.

Чем меньше используется глобальных переменных, тем меньше возможность получения не предвиденных программистом побочных эффектов в программе.

В стандарте языка С при наличии одноименных переменных в основной программе и подпрограмме локальные переменные закрывают глобальные. В С++ можно глобальную переменную сделать видимой, используя операцию разрешения видимости ::A.

3.1.2. Возврат из функции одного значения

Механизм возврата из функции одного значения реализуется оператором

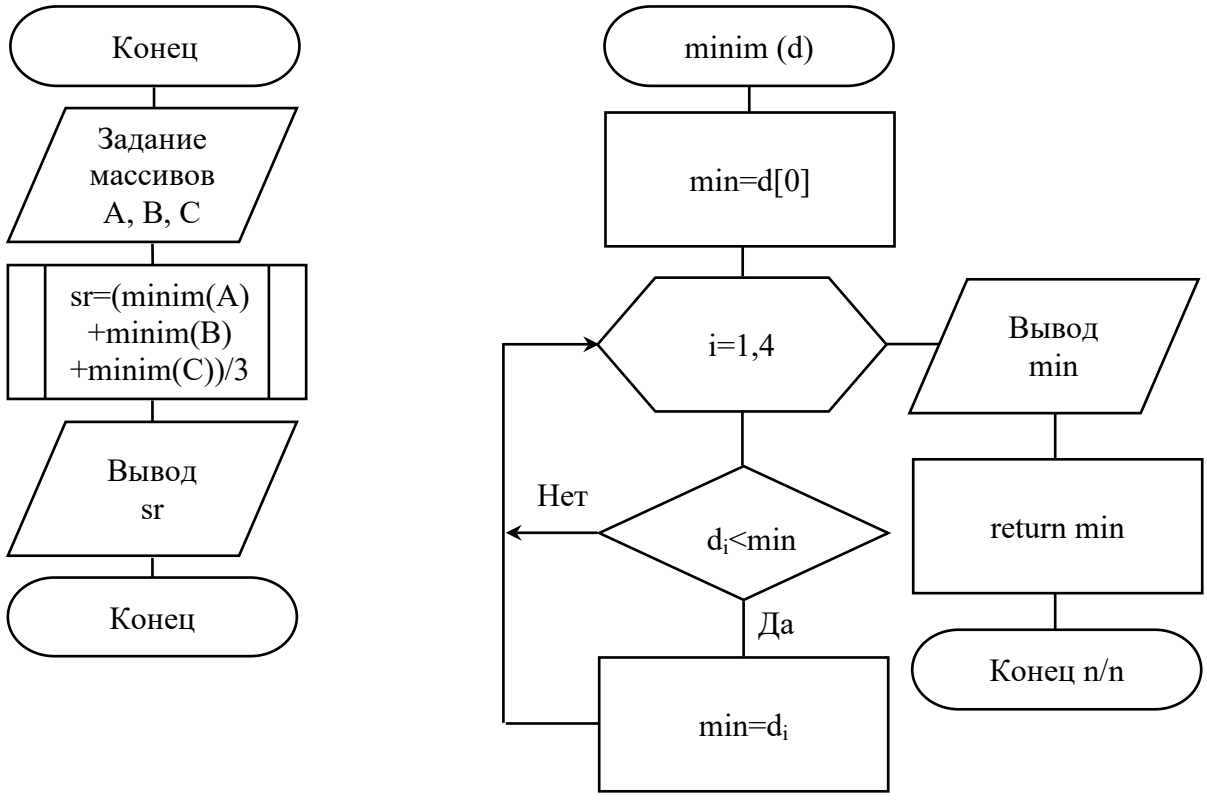
return [выражение];

При этом функция возвращает значение: <выражение> вычисляется, преобразуется, если необходимо, к типу возвращаемого значения, указанного в определении (объявлении) функции, и управление передается в точку вызова.

Пример возврата из функции одного значения

Даны три массива A[5], B[5], C[5]. Найти среднее арифметическое значение наименьших элементов в каждом из массивов.

До написания программы разрабатывается схема основного алгоритма (рис.12, *а*) и СА для нахождения наименьшего элемента в любом (формальном) массиве (рис.12, *б*).



а *б*

Рис. 12

Отлаженная программа решения задачи имеет следующий вид:

```
#include <stdio.h>
#include <conio.h>
// функция вычисления минимального элемента в массиве
int minim (int d[5])
{
int i, min;
min = d[0];
for (i=1; i<5; ++i)
        if ( d[i]<min )
                        min = d[i];
printf ("минимальный элемент равен %d\n", min);
return min;
}
void main ()                    // главная функция
{
     int a[] = {5,2,7,1,8};
     int b[] = {3,4,5,6,6};
     int c[] = {3,8,5,2,6};
           int n;
     float sr;
   sr = (minim(a) + minim(b) + minim(c)) / 3;
     printf(" sr= %5.2f ", sr);
     _getch();
}
```

Задание 8 для программирования задач с возвратом из функции одного значения

Написать и отладить программы для задач 1 и 2 из заданий 1 (программа1_1) и 6 (программа6_2 для двух массивов) с применением подпрограммы-функции с параметрами. Вычисление функции (или обработку массива) выполнить в подпрограмме, а ввод исходных данных и вывод результатов – в основной программе. Предусмотреть, по крайней мере, два обращения к функции с различными фактическими параметрами (двумя массивами).

Для отладки программ использовать средства среды C/C++: пошаговое исполнение программы (трассировку) или исполнение по контрольным точкам с проверкой значений переменных в окне отладчика Watch.

Задача 3 (программа8_3)

1. По заданным вещественным массивам A[1..6], B[1..6] и C[1..6] вычислить

$$Y = \begin{cases} (\max B)/\max A + (\max C)/\max(B+C) & \text{при } \min A < \max B, \\ \max(B+C) + \max C & \text{– в противном случае.} \end{cases}$$

2. Даны две квадратные вещественные матрицы шестого порядка. Напечатать квадрат той из них, в которой наименьший след (сумма диагональных элементов), считая, что такая матрица одна.

3. Определить координаты центра тяжести трех материальных точек с массами m_1, m_2, m_3 и координатами (x_1,y_1), (x_2,y_2), (x_3,y_3) по формулам:

$$x_c = (m_1x_1 + m_2x_2 + m_3x_3)/(m_1 + m_2 + m_3),$$

$$y_c = (m_1y_1 + m_2y_2 + m_3y_3)/(m_1 + m_2 + m_3).$$

Вычисление координаты оформить функцией с параметрами.

4. Вычислить все медианы для каждого из трех треугольников по заданным в массивах A, B, C сторонам: $m_a = 0.5\sqrt{2b^2 + 2c^2 - a^2}$,

$m_b = 0.5\sqrt{2a^2 + 2c^2 - b^2}$, $m_c = 0.5\sqrt{2a^2 + 2b^2 - c^2}$. Вычисление одной медианы оформить функцией.

5. Даны три одномерных массива вещественных чисел A[6], B[8] и C[1..7]. Найти общую сумму положительных элементов в массивах. Нахождение суммы элементов в массиве оформить функцией.

6. Даны два двумерных массива целых чисел с размерами (4*5) элементов. Подсчитать количество отрицательных элементов в каждом из них.

7. Даны два одномерных массива целых чисел A[8] и B[8]. Найти сумму их максимальных элементов. Для нахождения максимального элемента в массиве использовать функцию.

8. Даны два двумерных массива целых чисел с размерами (5×5) элементов каждый. Подсчитать произведение элементов главных диагоналей в каждом из них.

9. Даны три одномерных массива вещественных чисел A[6], B[8] и C[7]. Найти среднее геометрическое значение положительных элементов для каждого.

10. Даны две матрицы целых чисел M[3][2], K[3][3]. Найти среднее арифметическое значение для каждой из них.

11. Даны три одномерных массива целых чисел A[6], B[8] и C[7]. Подсчитать количество неотрицательных элементов в каждом.

12. Даны две матрицы целых чисел S[3][2], K[3][2]. В каждой матрице имеется по два одинаковых числа, распечатать их значения.

13. Даны два одномерных массива целых чисел A[7] и B[8]. Вычислить значение Z = (min A[i] + min B[j]) / (min A[i] – min B[j]).

14. По заданным целым массивам X[8] и Y[8-15] вычислить

$$Z = \begin{cases} \sum\limits_{i=0}^{7} x_i^2 \text{ при } \sum\limits_{i=0}^{7} x_i \; y_{i+8} > 0, \\ \sum\limits_{i=8}^{15} y_i^2 \qquad \text{– в противном случае.} \end{cases}$$

15. Дана матрица целых чисел D[6][5]. Найти наименьшую из сумм неотрицательных элементов строк матрицы. Для вычисления суммы использовать подпрограмму (функцию).

16. Дана матрица целых чисел E[3][5]. Используя функцию, найти среднее геометрическое значение для каждого столбца матрицы.

17. Дана матрица целых чисел F[4][5]. Найти наименьшие значения элементов в каждой из строк матрицы с помощью функции.

18. Даны две квадратные вещественные матрицы шестого порядка. Напечатать квадрат той из них, в которой наименьший след (сумма диагональных элементов), считая, что такая матрица одна.

19. Сформировать двумерный массив

$$\begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 4 & 9 & 16 & 25 \\ 1 & 8 & 27 & 64 & 125 \\ & & \dots & & \end{vmatrix}$$

Найти правило и оформить функцией вычисление любой строки.

20. Даны две матрицы целых чисел V[2][3], W[3][2]. Найти суммы элементов строк и столбцов в этих матрицах.

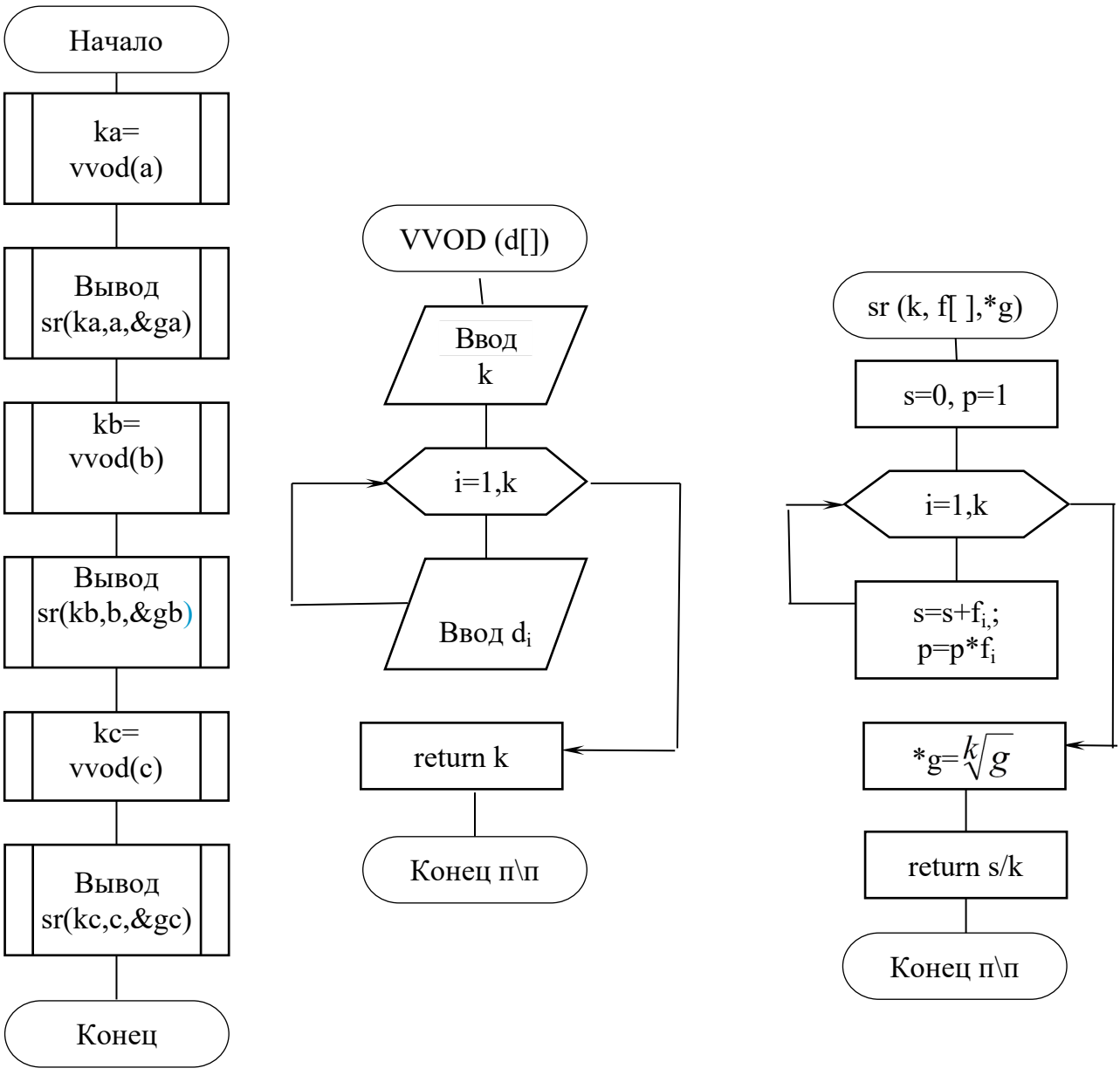
3.1.3. Возврат из функции нескольких значений

Для возврата значений из функции можно применить несколько способов. Первый способ заключается в использовании глобальных переменных. Второй способ позволяет возвратить одно значение с помощью оператора return, а другие записать в ячейки памяти с указанными адресами. В последнем случае в функцию необходимо передать адреса ячеек, куда надо поместить значения. Это выполняется с помощью указателей. Третий способ предполагает применение структур, содержащих элементы (поля) разных типов.

Пример возврата из функции двух значений

Ввести три массива A[3], B[4], C[5] (с помощью подпрограммы). Вычислить средние арифметические и геометрические значения в каждом из них.

Для решения задачи спроектированы СА (рис. 13): первая – для основного алгоритма (*а*), вторая – для ввода элементов массива (*б*), третья – для определения среднего арифметического и среднего геометрического значений в массиве (*в*).



а *б* *в*

Рис. 13

В соответствии с алгоритмом для данной задачи реализована программа:

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
int vvod(int d[])
{
    int i, k;
    printf("Vvedite chislo elementov ");
    scanf("%d",&k);
    printf("Vvedite massiv is %d elementov \n", k);
    for ( i=0; i<k; ++i )
        scanf("%d", &d[i]);
    return k;
}
float sr(int k, int f[], double *g)
{
    int i;
    long int s, p ;
    s = 0;   p = 1;
    for ( i = 0; i<k; ++i )
    {
        s += f[i];              //  s = s + f[i];
        p *= f[i];                      //  p =  p * f[i];
    }
    *g = pow((double)p, 1.0/k);
    return ((float)s)/k;
}
void main ()
{
    int ka, kb, kc;  double ga, gb, gc;
    int a[5], b[5], c[5];
```

```
    ka=vvod(a);
        printf("srA = %5.2f \t ", sr(ka,a,&ga));
        printf(" srGA = %5.2f \n",  ga);
    kb=vvod(b);
        printf("srA = %5.2f \t ",sr(kb,b, &gb));
        printf("srGB = %5.2f  \n", gb);
    kc=vvod(c);
        printf("srA = %5.2f \t ",sr(kc,c, &gc));
        printf("srGC = %5.2f \n", gc);
    _getch();
}
```

Задание 9 для программирования задач с возвратом из функции нескольких значений

Задача 1 (программа9_1)

Написать и отладить программу для задачи из задания 7 (программа7_1) с выделением алгоритма обработки или формирования массива. Организовать вызов процедуры с параметрами-массивами для двух наборов исходных данных. Обработку массив выполнить в подпрограмме, а ввод исходных данных и вывод результатов – в главной функции. Для ускорения отладки массив можно задать константой.

Задача 2 (программа9_2)

Для возврата второго и последующих значений из функции применять указатели.

1. Даны массивы A[6], B[6], C[6], вычислить суммы положительных и отрицательных элементов. Получить A*B, B*C, A*C. Вычисление произведения массивов оформить подпрограммой.

2. Даны массивы A[5], B[6]. Сформировать новые массивы путем сдвига элементов в массивах на два разряда вправо, освободившиеся слева

элементы обнулить. Сдвиг элементов в массиве оформить подпрограммой, из подпрограммы вернуть выпавшие справа разряды.

3. Даны два одномерных массива A[6] и C[6]. Получить A^2, C^2, A*C. Подсчитать число четных и число нечетных чисел в полученных массивах. Перемножение массивов и подсчет выполнить в подпрограмме.

4. Даны матрицы целых чисел S[3][2], K[3][2], в каждой из которых имеется по два одинаковых числа. Найти и напечатать их значения и индексы.

5. Вычислить значение функции $Z = x_1 + e^{x_2}$, где x_1, x_2 – корни уравнения $A_i x^2 + B_i x + C_i = 0$, где $i = 1, 2, \ldots, N$. Коэффициенты уравнения заданы в массивах A[N], B[N], C[N]. Для вычисления корней использовать подпрограмму.

6. Составить подпрограмму для перемножения двух квадратных матриц, с помощью которой вычислить вторую, третью и четвертую степени матрицы M[5, 5]. Для каждой матрицы вычислить сумму и среднее арифметическое значение элементов.

7. Даны массивы A[6], B[6], C[6]. Преобразовать их таким образом, чтобы каждому элементу массива присваивалось значение соседнего с ним справа. Последнему элементу присвоить значение первого. Напечатать исходные, результирующие массивы и отдельно последние два элемента результирующего.

8. По заданным вещественным массивам A[6], B[6] и C[6] вычислить $(\min A_i) / \max A_i + (\max C_i) / \min(C_i) + \max(B + C)_i / \min(B + C)_i$.

9. Даны массивы A[6], B[8]. Выбрать из них положительные элементы и записать соответственно в массивы A[k] и B[k], где $k < 6$, $n < 8$; из отрицательных элементов сформировать массивы A2[6-k], B2 [8-n]. Напечатать суммы и произведения элементов для каждого.

10. Даны массивы A[6], B[6], C[6]. Переставить элементы в них таким образом, чтобы слева подряд были записаны отрицательные, а справа положительные, подсчитать количество положительных элементов и число отрицательных в каждом массиве.

11. Даны две целые квадратные матрицы четного порядка. Элементы массивов с четными номерами строки и столбца заменить нулем (стереть). Напечатать исходные и полученные массивы, количество четных и число нечетных чисел в каждом.

12. Даны одномерные массивы A[6], B[8], C[10]. Записать их в виде матриц AA[3][2], BB[2][4], CC[5][2], найти наименьшие элементы в каждой и напечатать их номера.

13. Даны две целые квадратные матрицы шестого порядка. Распечатать элементы главных диагоналей для каждой из матриц и вычислить суммы элементов отдельно главной и побочной диагоналей.

14. По заданным 10 элементам вещественных массивов A, B и C вычислить

$$Z = \begin{cases} (\min(b_i)) / \max(a_i) + (\max(c_i)) / \min(b + c)_i & \text{при } \min A_i < \min B_i \\ \max(b + c)_i + \min(c_i) & \text{при } \min A_i > \min B_i \end{cases}$$

15. Даны матрицы целых чисел V[2][3], W[3][2]. Сформировать из них одномерные массивы VV и WW, записывая элементы построчно. Напечатать индексы исходных матриц для максимальных значений.

16. Дана матрица чисел H[5][6]. Переставить элементы в строках матрицы таким образом, чтобы они были расположены в порядке возрастания их модулей. Подсчитать, сколько в исходных матрицах положительных и отрицательных чисел.

17. Дана матрица чисел G[2][6]. Переставить элементы в матрице так, чтобы элементы каждого столбца матрицы были смещены циклически вправо. Перестановку элементов в столбце реализовать подпрограммой, напечатать два «выпавших» элемента.

18. Даны массивы A[6], B[6], C[6]. Упорядочить элементы в них в порядке убывания их модулей, напечатать наименьшее и наибольшее значения.

19. Даны две матрицы целых чисел V[2][3], W[2][3]. Найти общие суммы элементов строк, из функции вернуть две суммы.

20. Даны две целые квадратные матрицы шестого порядка. Вычислить суммы элементов выше главной диагонали и ниже нее. Определить, можно ли отражением относительно главной диагонали преобразовать одну в другую.

3.1.4. Рекурсивные вызовы функций

Рекурсия – это способ организации вычислительного процесса, при котором функция в ходе выполнения своих операторов обращается сама к себе.

Пусть задано целое положительное число. Вывести на экран дисплея цифры в обратном порядке. Выделим в рекурсивную подпрограмму определение остатка от деления *M* на 10 для определения очередной цифры и рекурсивный вызов себя с параметром, равным частному от деления числа (первоначально – исходного, зачет полученного на предыдущем шаге после деления) на 10. Фрагмент программы показан ниже:

```
int N;
void  REVERS(int M)
{
printf(«%d», M % 10);
if (M / 10)   REVERS(M /10);          //  M/10 != 0
}
    void main()
       {
       scanf(«%d», N);
       REVERS(N);
       }
```

В ходе выполнения программы функция рекурсивно обращается сама к себе и выводит на экран при каждом обращении очередную цифру до тех пор, пока частное не станет равно нулю (M/10 = 0).

Рекурсия может быть прямой или косвенной. В первом случае функция содержит оператор вызова этой же функции. Во втором случае один модуль (например, A) вызывается из другого модуля (например, B), а B – из A.

Поскольку по правилам языка каждый идентификатор перед использованием должен быть определен, то необходимо выполнить опережающее описание подпрограммы B. Для этого объявляется заголовок функции B (прототип). Теперь из функции **A** можно обращаться к функции **B**.

Например:

```
void  B(int );
void A(int  J);
{
...
B(J);
}
void B(int I);
{
    ...
    A(I);
}
```

3.1.5. Вызовы функции с переменным числом аргументов

При вызове функции с переменным числом аргументов просто задается нужное количество аргументов.

В объявлении же (прототипе) и определениях такой функции список типов или формальных параметров заканчивается многоточием (, ...):

в определении [класс] <тип> [*] <имя_функции>(пар1,пар2,...);

в прототипе (объявлении) ... <имя_функции>(тип1,тип2,...);

При вызове функции все фактические аргументы размещаются в стеке и их значения присваиваются соответствующим формальным параметрам, указанным в определении. Затем сравниваются типы перечисленных формальных и фактических параметров.

Примерами функций с переменным числом аргументов являются функции ввода/вывода из библиотеки <stdio.h>:

printf (печать), fprintf (в файл), sprintf (в строку символов) для форматированного вывода любого вида данных,

fscanf, sscanf – для форматированного ввода.

Прототипы функций форматированного вывода:

int printf (const char *<строка форматов>, пар1,пар2,...); – функция форматированного вывода в стандартный поток stdout (экран дисплея) или в выходной поток, const – ключевое слово спецификатора для задания объектов, которые не могут подвергаться модификации.

Аналогично определяется функция чтения:

scanf (const char * format[, адрес], ...) – функция форматированного чтения из входного потока stdin (с клавиатуры);

int fprintf (FILE *stream, const char *format [,параметр], ...) – функция форматированной записи в файл согласно формату;

int fscanf (FILE *stream, const char *format [,адрес], ...) – функция чтения из файла.

Функция чтения из строки символов имеет вид:

int sscanf(const char *s, const char *format, ...);

sprintf(const char *s, const char *format,...) – функция записи в строку.

3.2. Работа с файлами

Под файлом обычно подразумевается именованная область памяти на внешнем носителе.

Прежде чем читать информацию из файла или записывать данные в него, нужно его открыть. Для этого в библиотеке **<stdio.h>** имеется специальная функция

FILE *fopen(char *fname, char *mode);

где ***fname** – имя файла; ***mode** – режим ([1, 2, 3, 5]. Функция возвращает указатель (ссылку) на файл, который должен быть предварительно описан. Например, объявим указатели на переменные файлового типа:

```
FILE *uin, *uout;          // два указателя на переменные файлового типа;
uin = fopen("name1", "r");     // открыть файл "name1" для чтения
                               и далее идентифицировать как uin
uout = fopen("name2", "w");    // открыть для записи и связать с иден-
                               тификатором uout.
```

По умолчанию fopen открывает файлы, используя кодировку ANSI, но также поддерживает файловые потоки Юникода. Поэтому, чтобы открыть новый или существующий файл Юникода, необходимо передать ccs-флаг, указывающий нужную кодировку, например:

fopen("newfile.txt", "w+, ccs=UNICODE");

либо при создании файла сохранить его с указанием нужной кодировки.

Если производится открытие несуществующего файла, то он создается. Для открытия файла с именем test рекомендуется следующий прием, который позволяет определить ошибку открытия файла [6, 8]:

```
# include <stdio.h>          // работа с файлами и константа NULL
# include <stdlib.h>         // для функции exit()
void main()
{
```

```
    FILE *fp;
    if (( fp=fopen("test","w")) == NULL)
        {
            puts("Не могу открыть файл\n");  // печать строки
            exit(1);
        }
    puts("Файл открыт\n");
}
```

Если компилятор выдаёт сообщение, что функция fopen является небезопасной, так как в современных IDE данная функция Microsoft считается уязвимой к атакам переполнения буфера, то можно либо отключить предупреждения перед включением заголовочных файлов с помощью строки #define _CRT_SECURE_NO_WARNINGS, либо использовать более безопасную версию функции fopen_s, которая выполняет дополнительную проверку ошибок:

errno_t fopen_s(FILE fp, char *fname, char *mode);** возвращает код ошибки, а ссылка на файл внутри функции.

Режимы открытия файлов и функции для работы с файлами

При открытии файлов применяются следующие режимы:

«r» – открыть для **чтения**,

«w» – создать для **записи**,

«a» – открыть для **добавления** в существующий файл,

«rb» – открыть **двоичный** файл для **чтения**,

«wb» – создать **двоичный** файл для **записи**,

«ab» – открыть **двоичный** файл для **добавления**,

«r+» – открыть для **чтения и записи**,

«w+» – **создать** для чтения и записи,

«a+» – открыть файл для добавления или создать для чтения и записи,

«r+b» – открыть **двоичный** файл для **чтения** и **запис**и,

«w+b» – создать **двоичный** файл для **записи и чтения**,

«a+b» – открыть **двоичный** файл для **добавления** или создать для **чтения** и **записи**,

«rt» – открыть **текстовый** файл для **чтения**,

«wt» – создать **текстовый** файл для **записи**,

«at» – открыть **текстовый** файл для **добавления**,

«r+t» – открыть **текстовый** файл для **чтения** и **записи**,

«w+t» – создать **текстовый** файл для **записи и чтения**,

«a+t» – открыть **текстовый** файл для **добавления** или создать для **чтения** и **записи**.

В библиотеке <stdio.h> определены также следующие функции работы с файлами (fp – указатель на файл, возвращаемый функцией):

int fclose(FILE *fp) – **закрытие** файла. Возвращает нуль, если операция выполнена успешно, и иное значение в противном случае. Функция является рекомендуемой, поскольку файлы при нормальном завершении закрываются автоматически;

int puts(int ch, FILE *fp) – **записать** символ типа **int** в поток. Возвращается записанный символ, если операция была успешной. Если произошла ошибка, возвращается EOF;

int gets(FILE *fp) – **чтение** символа типа **int** из потока (или **gets_s**). Возвращает признак конца файла EOF, если достигнут конец файла или произошла ошибка при чтении файла;

int feof(FILE *fp) – возвращает значение «истинно», если достигнут конец файла, и нуль в противном случае; например:

while (!feof(fp)) { ch = getc(fp); } // вводятся символы, пока не достигнут конец файла;

int ferror(FILE *fp) – возвращает значение нуль, если обнаружена ошибка (рекомендуется для обнаружения ошибок чтения и записи после каждой операции с файлами);

int fprintf(FILE *fp, const, char *string, …) – форматированная запись в файл (функция содержит указатель на файл и управляющую строку со списком);

int fscanf(FILE *fp, const,char *string) – форматированное чтение из файла (или **fscanf _s**);

unsign fread(void *buf, int bytes, int c, FILE *fp) – читает блок данных из потока (буферный обмен); **buf** – указатель на область памяти, откуда происходит обмен информации; **c** – количество единиц записи длиной bytes для считывания;

unsign fwrite(void *buf, int bytes, int c, FILE *fp) – пишет блок данных в поток;

int remove(char *filename) – уничтожается файл, возвращается нуль при успешной операции;

unsign rewind(FILE *fp) – устанавливает указатель на начало файла;

int fseek(FILE *fp, long numbyte, int orig) – устанавливает указатель позиции файла в заданное место, numbyte – количество байт от точки отсчета (0,1,2), orig – макрос: 0 – начало, 1 – текущая позиция, 2 – конец;

void abort() (из библиотеки <stdlib.h>) – немедленное прекращение программы без закрытия файлов и без освобождения буферов.

После окончания работы с файлом необходимо его закрыть функцией **int fclose(FILE *fp)**, которая возвращает нуль, если операция выполнена успешно, и иное значение в противном случае.

Пример работы с данными из файлов

Вычислить суммы элементов в столбцах двумерного числового массива A[2][4], заданного в файле FILE1.txt. Полученные значения поместить в файл FILE2. txt. Для решения задачи разработана СА (рис. 14).

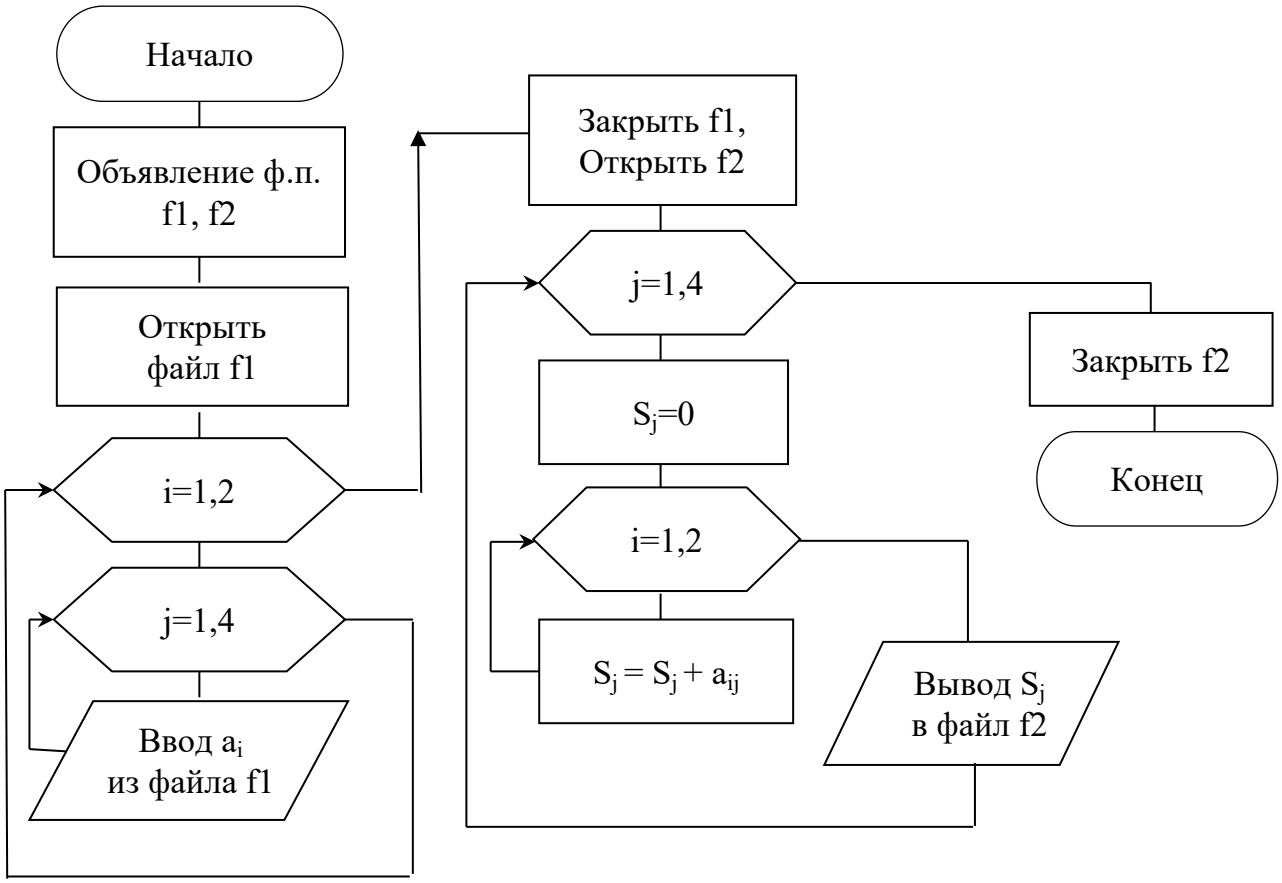


Рис. 14

Выделенные в алгоритме действия представлены следующей программой:

```
#define _CRT_SECURE_NO_WARNINGS
```

/* Директива используется при обработке программы компилятором Visual Studio для отключения предупреждений компилятора об использовании функций, которые Visual Studio считает устаревшими или небезопасными. Эту директиву рекомендуется применять, когда компилятор выдаёт сообщение об ошибке, что функция fopen является небезопасной */

```
#include <conio.h>
#include <stdio.h>
void main()
{
    int a[2][4];            // объявление переменных
    int s[4];
    int i, j;
```

```
    FILE * f1,* f2;                 // объявление файловых переменных
    f1 = fopen("file1. txt", "r"); // открыть файл file1.txt для чтения
    for (i=0; i<2; ++i)
       for ( j=0; j<4; ++j )
              fscanf(f1, "%d", &a[i][j]);     // чтение из файла, связанного с  f1
    fclose(f1); // закрыть файл, связанный с  f1
    f2 = fopen("file2. txt","w");       // открыть файл для записи
    fprintf(f2, "1 2 3 4 – столбцы \n");     // нумерация столбцов матрицы
    fprintf(f2, "суммы по столбцам: \n");
    for ( j=0; j<4; ++j )
    {          s[j] = 0;
               for (i=0; i<2; ++i)
                     s[j] = s[j] + a[i][j];
               fprintf(f2, "%d", s[j]);
    }          // на экран ничего не выводится
fclose(f2); // закрыть файл, связанный с f2
    // результат действий программы надо смотреть в file2. txt
}
```

Задание 10 для решения задач со вводом данных из файла и/или выводом в файл

Задача 1 (программа10_1)

Выполнить задачу из лабораторной работы 9 (программа9_2) при считывании исходных данных из одного файла, записать полученные результаты в другой текстовый файл. Файл с исходными данными должен быть подготовлен предварительно и находиться в текущей папке.

Задача 2 (программа10_2)

1. В текстовом файле T1 записана последовательность целых чисел, разделенных пробелами. Написать программу, переписывающую в текстовый файл T2 все положительные числа из T1.

2. В текстовом файле f1 записана последовательность чисел. Сформировать текстовый файл f2, содержащий все числа в обратном порядке.

3. В текстовом файле T1 записана последовательность чисел. Разработать программу, переписывающую в текстовый файл T2 сначала все положительные числа из T1, нуль, затем – все отрицательные числа.

4. В текстовом файле f1 дана последовательность целых чисел, разделенных пробелами. Записать в текстовый файл F2 все положительные числа, а в файл F3 – отрицательные числа и нулевые элементы.

5. Сформировать файл из девяти строк, в первой из которых – одна единица, во второй – две двойки,…, в девятой – девять девяток.

6. Даны два файла F1, F2. Сформировать третий файл F, в который сначала записать все элементы файла F1, а затем – элементы файла F2.

7. Элементы каждого из заданных в двух файлах массивов X и Y упорядочены по возрастанию. Объединить элементы этих двух массивов в один файл F с упорядочением их по возрастанию.

8. В заранее подготовленном файле K задан массив из 15 целых чисел. Выделить в другой файл те элементы массива K, индексы которых являются степенями двойки.

9. В текстовом файле S1 записана последовательность из N*K чисел. Написать программу, формирующую в файле S2 матрицу из N строк по K чисел в каждой строке.

10. В файле W задан массив из девяти элементов. В новом файле WW сформировать массив, в котором каждому элементу будет присвоено значение соседнего с ним справа. Последнему элементу присвоить значение первого.

11. В файле D1 задана квадратная матрица вещественных чисел. Сформировать файл D2, в котором поменять местами первую и последнюю строки матрицы.

12. В текстовом файле F подготовить последовательность целых чисел, разделенных пробелами. Написать программу, записывающую в текстовый файл FF все различные значения, встречающиеся в файле F.

13. В файле X задан массив из 20 чисел. Упорядочить его по возрастанию и занести в файл Y.

14. В файле X1 задана матрица целых чисел. Сформировать файл X2, в котором все элементы с четными номерами строки и столбца матрицы заменить нулем (стереть).

15. Задана в файле строка текста. Выполнить все циклические сдвиги слов в заданном тексте влево. Каждый полученный при этом текст вывести в другой файл.

16. В файле Y1 задана матрица вещественных чисел. Сформировать файл Y2, в котором строки матрицы упорядочены по убыванию элементов первого столбца.

17. В файле дан текст из 20 символов. Напечатать сначала все цифры, входящие в него, затем латинские буквы, далее русские буквы и все остальные литеры, сохраняя при этом взаимное расположение символов в каждой из четырех групп. Создать четыре новых файла, каждый из которых содержит свою группу символов.

18. Имеется текстовый файл BOOK. Написать программу, которая игнорирует исходное деление этого файла на строки, переформирует его, разбивая на строки так, чтобы каждая строка оканчивалась точкой либо содержала ровно 40 литер, если среди них нет точки.

19. Имеется текстовый файл T. Вывести в новый файл и напечатать первую из самых коротких его строк.

20. Имеется текстовый файл, содержащий фразу. Написать программу, которая считывает из файла литеры до первой точки и записывает их (без точки) в другой текстовый файл, формируя в нем строки по 20 символов (в последней строке литер может быть меньше).

3.3. Типы, определяемые пользователем

3.3.1. Строки и символы

3.3.1.1. Символьный тип данных

Символьный тип (**char**) – это тип данных, используемый для описания отдельных символов (знаков, букв, кодов) [1, 2]. Символьный тип содержит элементы, состоящие из одного символа. Значением является сам символ, знак или литера, заключенная в апострофы, либо код символа по международной таблице кодировки ASCII (American Standart Code Information Interchange).

Переменная символьного типа определяется ключевым словом **char:**

```
char   c1, c2, c3= 'A';
```

символом считается управляющая последовательность, например: ‘\n’ (см. табл. 1).

Символы с кодами 0...31 относятся к служебным. Наиболее часто используются значения следующих кодов:

007 – звонок (выдается звуковой сигнал),

009 – горизонтальная табуляция,

010 – перевод строки,

013 – возврат каретки,

026 – признак конца файла данных,

027 – код клавиши «Esc».

Код ASCII семибитовый и позволяет кодировать 128 символов, включает цифры, латинские буквы. Латинские буквы: ‘A’ имеет код 65, ‘a’ – код 97.

В C/C++ применяется 8-битовый код (256 символов), причем национальные алфавиты помещены во вторую половину таблицы кодировки символов после номера 128. Эта часть таблицы зависит от страны-производителя компьютера. Одна из кодировок для русских букв содержит следующие значения: ‘А’ – 128…’Я’ – 159; ‘а’ – 160…’п’ – 175; ‘р’ – 192…’я’ – 207.

Элементы символьного типа обладают свойством упорядоченности.

Цифры и буквы упорядочены так же, как и их коды:

‘0’ < ‘1’ < ‘2’ < … < ‘9’ и коды 48 < 49 < 50 < …. <57;

‘A’ < ‘B’ < ‘C’ < … < ‘Y’ < ‘Z’ < ‘a’ < ‘b’ … ’y’ < ‘z’ и

65 < 66 < 67 < … < 89 < 90 < 97 < 98 < … 121 < 122;

‘А’ < ‘Б’ < ‘С’ < … < ‘Ю’ < ‘Я’ < ‘а’ < ‘б’ … ’ю’ < ‘я’.

К символам типа char применимы присваивания и все операции отношений: <, >, <=, >=, ==, !=.

3.3.1.2. Функции для работы со строками

Строка – это массив символов, завершающийся признаком конца строки ‘\0’. Поэтому со строкой можно работать как с массивом символов. Однако в C/C++ предусмотрены функции, которые позволяют выполнять действия целиком над строкой с применением специальной библиотеки <string.h>.

Основные функции для работы со строками (<string.h>)**:**

unsigned **strlen(const char *str);** – определяет длину строки **str**;

char ***strcpy(char *sp, const char *si);** – копирует строку **si** в строку **sp**;

char * **strncpy(char *sp, const char *si, int kol);** – копирует первые **kol** символов строки **si** в строку **sp**, «хвост» отбрасывает или строка дополняется пробелами;

int **strcmp(const char *str1, const char *str2);** – сравнивает строки **str1** и **str2**. Результат отрицателен, если str1< str2; равен нулю, если str1= str2, и положителен, если str1 > str2 (сравнение беззнаковое);

int **strncmp(const char *str1, const char *str2, int kol);** – сравнивает строку **str1** с **kol** символами **str2**;

char ***strcat (char *sp, const char *si);** – объединяет строки (конкатенация);

char * **strncat(char *sp, const char *si, int kol);** – приписывает **kol** символов строки **si** к строке **sp**;

char * **strstr (const char *str1, const char *str2);** – ищет в строке **str1** подстроку **str2**. Возвращает указатель на тот элемент в строке str1, с которого начинается подстрока str2;

char ***strchr (const char *str, int c);** – ищет в строке **str** первое вхождение символа **c**;

char * **strrchr (const char *str, int c);** – ищет в **str** последнее вхождение **c**;

int **strpbrk (const char *str1, const char *str2);** – ищет в строке **str1** первое появление любого символа из строки **str2**.

Приведем некоторые дополнительные функции для работы со строками, размещенные в библиотечных файлах string.h, stdlib.h:

double **atof** (const char *str); – преобразует строку str в вещественное число типа double;

int **atoi** (const char *str); – преобразует строку str в целое число типа int;

long **atol** (const char * str); – преобразует строку str в целое число типа long;

char * **ultoa** (unsigned long v, char *str, int baz); – преобразует беззнаковое длинное целое v в строку str;

char * **strlwr**(char *str); – преобразует буквы верхнего регистра в строке в соответствующие буквы нижнего регистра;

char * **strupr**(char *str); – преобразует буквы нижнего регистра в строке str в буквы верхнего регистра.

Пример формирования новой строки из исходной

Пример. Ввести строку **st1**. Сформировать строку **st2** перестановкой символов исходной строки в обратном порядке и заменой всех строчных латинских букв прописными.

```
#include <locale.h>      // для  setlocale
#include <conio.h>
#include <stdio.h>
```

```
#include <string.h>
#include <ctype.h>
void main()
{
setlocale(LC_ALL, “RUS”);                    // для русского шрифта
    int i, j;
    char st1[255], st2[255];
printf("Введите строки \n");
gets_s(st1);                                          // ввод строки
printf ("исходная строка : %s\n",st1);
j = strlen(st1)-1;              //максимальный индекс элемента в строке
for ( i=0; i<strlen(st1); ++i)
{
   st2[i] = toupper(st1[j]);                          // изменить регистр
   j -= j;                    // j = j-1
}
st2[i] = '\0';                                        // символ конца строки
printf("новая строка %s ",st2) ;
printf ("\n");                                         // перевод строки
_getch();
}
```

Задание 11 для решения задач с символами и строками

Задача 1 (программа11_1)

1. Дана строка из 20 символов. Вывести из нее на печать только строчные буквы латинского алфавита.

2. Вывести на печать все строчные, а затем все прописные буквы русского и латинского алфавитов.

3. В заданной строке подсчитать частоту появления букв «a», «b».

4. Дан текст из 60 литер. Напечатать только строчные русские буквы, входящие в этот текст.

5. Дана последовательность символов, содержащая символ «я». Определить порядковый номер символа «я» в последовательности.

6. Дана последовательность символов. Определить в ней символ, который по алфавиту предшествует другим.

7. Напечатать в алфавитном порядке все различные строчные буквы, входящие в заданный текст из 100 литер.

8. Определить, является ли заданная последовательность символов в строке симметричной: читается одинаково слева направо и справа налево.

9. Напечатать текст, образованный символами с порядковыми номерами 66, 89, 84 и 69, и текст с изменением регистра.

10. Даны две строки c1 и c2, содержащие до 5 цифр каждая. Преобразовать их к данным целого типа. Вычислить арифметическое выражение c3 = (c1-c2)/(c1+c2).

11. Вычислить суммы кодов всех букв, входящих в слова SUM и ALFA. Сравнить слова и определить, какое из них больше.

12. Напечатать заданный текст с удалением из него всех букв «b», непосредственно перед которыми находится буква «c».

13. Имеется символьная переменная d, присвоить логической переменной T значение true, если значение d – цифра, и значение false в противном случае.

14. Если в заданный тексте входит каждая из букв слова key, тогда напечатать «yes», иначе – «no».

15. Написать программу, которая предварительно запрашивает ваше имя, а затем приветствует вас по имени.

16. Ввести вещественное число, преобразовать его в строку. Подсчитать количество разрядов в целой и дробной частях. Найти представление числа в виде мантиссы (по модулю меньшее единицы) и порядка.

17. Ввести строку, состоящую из нулей, единиц и десятичной точки. Преобразовать ее в десятичное число.

18. Ввести строку, состоящую из арабских цифр и десятичной точки. Преобразовать ее в десятичное число.

19. Дана строка символов, заменить каждую букву следующей за ней по алфавиту.

20. Ввести три пары строк (строчными или прописными буквами, в разных регистрах для одинаковой и разной длины). Напечатать результат сравнения.

Задача 2 (программа11_1)

1. Дана строка, содержащая не более двадцати латинских букв. Все вхождения «max» в ней заменить на «min» и «макс» на «мин». Подсчитать число таких замен.

2. Дана строка, содержащая сорок латинских букв. Подсчитать все вхождения «abc» в строку и их удалить. Вывести на экран два варианта полученных строк, заполняя образовавшуюся «дыру» последующими буквами с добавлением в конце пробелов и оставляя на месте удаленных символов пробелы.

3. Определить, сколько различных символов входит в заданный текст, содержащий не более 100 литер и оканчивающийся точкой.

4. Определить номера позиций гласных букв в заданном тексте.

5. Напечатать заданный текст из 60 символов, удалив из него повторные вхождения каждой литеры.

6. Дана строка, состоящая из слов, разделенных пробелами, в конце строки – точка. Определить, сколько в строке слов, содержащих четное число символов.

7. Дан набор слов на английском языке, разделенных пробелами, в конце – точка. Выделить в последовательности нечетные слова прописными буквами.

8. Дан непустой текст из строчных букв, за которыми следует точка. Определить, упорядочены ли эти буквы по алфавиту. Напечатать результат проверки и исходный текст прописными буквами.

9. Дана последовательность от двух до восьми слов, в каждом из которых от одной до десяти строчных букв, между соседними словами – не менее одного пробела, за последним словом – точка. Напечатать слово с максимальной длиной.

10. Дано несколько слов, в каждом из которых от одной до семи строчных букв, между соседними словами – не менее одного пробела, за последним словом – точка. Напечатать эти слова в алфавитном порядке.

11. Дана последовательность, содержащая от одного до восьми слов, в каждом из которых 1–5 строчных букв, между соседними словами – запятая, за последним словом – точка. Напечатать эту же последовательность, удалив из нее повторные вхождения слов.

12. В заданном тексте (слова разделены пробелами) поменять местами первое и последнее слово.

13. Даны число K и текст из слов, разделенных пробелами, в конце – точка. Определить количество слов в тексте, состоящих из K букв.

14. Дана последовательность, содержащая от двух до десяти слов, в каждом из которых 1–8 строчных букв, между соседними словами – не менее одного пробела, за последним словом – точка. Напечатать те слова, в которых буквы слова упорядочены по алфавиту.

15. Дана последовательность, содержащая от двух до десяти слов, в каждом из которых 1–5 строчных букв, между соседними словами – запятая, за последним словом – точка. Напечатать эту же последовательность слов, но в обратном порядке.

16. В заданный текст входят только цифры и буквы. Определить, является ли текст десятичной, шестнадцатеричной или двоичной записью целого либо вещественного числа (указать при выводе какого).

17. В заданном тексте найти и, если есть, напечатать все слова-палиндромы (слова, которые одинаково читаются слева направо и справа налево).

18. Дан текст из слов, разделенных пробелами, в конце – точка. Найти слово наименьшей длины (содержащее наименьшее количество букв).

19. Разработать программу шифровки-дешифровки текста путем замены каждой буквы текста другой, с кодом на N больше (меньше) исходной.

20. Дана непустая последовательность слов, в каждом из которых 1–6 строчных букв, между соседними словами – запятая, за последним – точка. Напечатать те слова, у которых одинаковые «соседи». Определить функцию, которая вводит очередное слово и присваивает его шестилитерной строке, а запятую или точку присваивает некоторой глобальной переменной.

3.3.2. Структуры

Структурой называется совокупность логически связанных переменных, возможно различных типов, сгруппированных под одним именем.

Шаблон для задания структуры в языке C/C++ описывается в следующем виде:

```
struct <имя структуры>
  {
        тип1 <имя_поля1>;
        тип2 <имя_поля2>;
        типN <имя_поляN>;
  }  ;
```

Определение шаблона структуры заканчивается точкой с запятой.

Элементы структуры называются полями или членами структуры и могут иметь любой тип, кроме типа этой же структуры, но могут быть указателями на него. Структура может содержать элементы разных типов. Частными видами полей структуры являются битовые поля и объединения (union).

Пример определения структуры STUD (сведений о студенте)

```
struct STUD {
                    char Name[30];
   char Groupe[5];
   int Age;
          };
```

Такая запись не задает никакой переменной, и выделения памяти не происходит. Под именем STUD задается частный вид структуры или шаблон структуры, т. е. определен новый тип struct STUD. Чтобы оперировать структурой в программе, необходимо объявить переменные. Для объявления конкретных переменных в новом типе можно написать:

struct <имя_структуры> <список переменных>;

например: struct STUD stud1, stud2;

Переменные stud1, stud2 также могут задаваться одновременно с шаблоном (второй способ задания)

```
struct STUDT {
                char Name[30];
                char Groupe[5];
                int Age;
        } stud1, stud2;
```

Таким образом, объявляются две переменные и компилятор автоматически выделит память под них.

Внешние статические структуры можно инициировать, помещая следом за определением список начальных значений элементов:

struct stud1 = {"Петров Е.","ИВТ-231",18};

Доступ к элементу поля осуществляется с помощью ***операции*** «точка» или ***операции*** «выделения элемента»:

<имя_переменной_стр>.<имя_поля>

Например: strcpy(stud2.name, "Симонов К.");
при этом выполняется копирование строки «Симонов К.» в поле name переменной stud2.

Для печати содержимого поля номера группы элемента структуры запись будет иметь вид:

printf(" %c", stud1.Groupe);

Структурные переменные могут объединяться в массивы структур. Для объявления массива структур сначала задается шаблон структуры, далее объявляется массив:

struct STUD st[30]; // в памяти создается 30 переменных [0…29].

Если объявлены две переменные типа структура с одним шаблоном, то их можно присваивать друг другу: **stud2 = stud1;**

В этом случае происходит побитовое копирование каждого поля одной структурной переменной в соответствующее поле другой переменной. Структурные переменные, описанные под разными именами (даже идентичные друг другу), присваивать нельзя.

Переменная типа структуры может быть глобальной, локальной или формальным параметром. Любое поле структуры может быть параметром функции, параметром может являться и адрес одного поля. Можно в качестве формального параметра передать по значению всю структуру, создать указатель на структуру и передать аргумент типа структуры по ссылке. Объявление указателя на структуру имеет вид:

struct <имя структуры> * <имя указателя>;

например, struct STUD m, *uk; // uk – переменная типа указатель на структуру STUD.

Если передается структура по значению, то все ее элементы заносятся в стек. Если она содержит в качестве своего элемента массив, стек может переполниться. Поэтому рекомендуется использовать ссылки. При передаче по ссылке в стек заносится только адрес структуры, при этом копирование структуры не происходит, но появляется возможность менять содержимое элементов.

Указателю можно присвоить адрес переменной: **uk = &m**.

Для получения значения поля Age переменной **m** используется операция доступа к полю через указатель:

(*uk). Age или **uk ->** Age**;**

Структура операции доступа к полю по указателю

переменная_указатель -> имя_поля;

(переменная_указатель -> элемент_структуры;).

Операция «стрелка» применяется, когда необходим доступ к значению элемента структуры через переменную-указатель.

В качестве элементов структуры можно использовать массивы, другие структуры и массивы структур. Например:

```
struct Adr char            // структура для задания адреса
              {    city[30];
                    int ind;
char adres[40];
              };
struct STUDadr char Name[30];
                        {       struct Adr addr;
                                char groupe[6];
                        }       st1, st2;
```

Adr – шаблон структуры для адреса, который должен быть определен до объявления структуры STUDadr.

Для присваивания значения элементу ind структуры STUDadr значения надо записать: st1.addr.ind=50.

Пример программирования задач со структурами

Разработать программу формирования списка студентов с указанием дат рождения и выборки студентов по году рождения и по фамилии (имени).

```
#include <stdio.h>
#include <conio.h>
#include <string.h>      // для работы со строками или <cstring>
#include <iostream>    //для потокового  ввода /вывода
#include <windows.h>
using namespace std;
```

```
struct student                    //определение нового типа
{
        char name[25];
        struct                    // вложенная структура
        {
                int den, mes;
                int god;
        } denRogd;
};
void main()
{
        SetConsoleCP(1251);          /* позволяет корректно отображать
кириллицу при вводе данных */
        SetConsoleOutputCP(1251);      // при выводе
        struct student z[20];
        int g, i, n;                      // g - год
        char st[25];
        printf("\n   Введите количество студентов:\n");
        scanf_s("%d", &n);
        for (i = 0; i < n; ++i)
        {
                printf("  Введите имя  %d студента\n", i + 1);
                cin >> z[i].name;                    //потоковый  ввод
                printf("  Введите день рождения\n ");
                scanf_s("%d", &z[i].denRogd.den);
                printf("  Введите месяц рождения\n");
                scanf_s("%d", &z[i].denRogd.mes);
                printf("  Введите год рождения\n");
                scanf_s("%d", &z[i].denRogd.god);
        }
        printf("\n   Введите год рождения для поиска:\n ");
        scanf_s("%d", &g);
```

```
	printf("\n   Список    \n");
	for (i = 0; i < n; ++i)
	{
		if (z[i].denRogd.god == g)
		{
			printf("Имя студента : ");
			printf("%s\n", z[i].name);
			printf("Дата рождения: ");
			printf("%d. ", z[i].denRogd.den);
			printf("%d. ", z[i].denRogd.mes);
			printf("%d", z[i].denRogd.god);
			printf("\n");
		}
	}
	printf("\n   Введите имя студента для поиска:\n ");
	 cin >> st;                                  //потоковый  ввод
	printf("\n Данные о студенте  \n");
	for (i = 0; i < n; ++i)
	{
		if (strcmp(z[i].name,st) == 0)
		{
			printf("Имя студента : ");
			printf("%s\n", z[i].name);
			printf("Дата рождения: ");
			printf("%d. ", z[i].denRogd.den);
			printf("%d. ", z[i].denRogd.mes);
			printf("%d", z[i].denRogd.god);
			printf("\n");
		}
	}
	_getch();
}
```

Задание 12 для программирования задач со структурами

Задача 1 (программа12_1)

1. Ввести оценки студента по 5 экзаменам. Определить средний балл и подсчитать количество удовлетворительных, хороших и отличных оценок. Напечатать название предмета, если есть «неуд».

2. Дан список студентов и оценка каждого на экзамене (оценки на N экзаменах). Подсчитать количество удовлетворительных оценок, хороших, отличных и средний балл в группе. Напечатать фамилии неуспевающих студентов.

3. Дан список студентов группы. Заполнить его следующими сведениями: фамилия, имя, отчество, имеет ли компьютер (если имеет, то какой и с какого года). Подсчитать, сколько студентов имеют ПК.

4. Сформировать список студентов группы со следующими сведениями: фамилия, имя, отчество, знает ли языки программирования Piton, C, C++ (если да, где обучался и сколько лет). Подсчитать, сколько студентов знают язык C, сколько – C++, сколько – три языка.

5. Сформировать список студентов группы, в котором указать фамилию (имя, отчество), город, в котором получил среднее образование (номер школы, если обучался в Омске). Подсчитать, сколько в группе иногородних студентов.

6. Сформировать структуру «ключевые слова C/C++»: слово и перевод. Подсчитать их количество. Организовать поиск: по ключевому слову – перевод, и наоборот.

7. Сформировать телефонный справочник. По номеру организовать поиск владельца, и наоборот.

8. Сформировать запись «даты – праздники». Организовать в программе ввод дат, подсчитать число праздничных дней и рабочих.

9. Сформировать запись «английское слово – перевод». Вводя слово (английское или русское), найти перевод или выдать сообщение «нет в словаре». По возможности предусмотреть пополнение словаря.

10. Сформировать запись «операторы C/C++»: оператор, действие (разновидность). Подсчитать их количество и максимальное количество в разновидности.

11. Сформировать запись «типы C/C++»: имя, тип, операции, разрешенные в данном типе. Подсчитать количество разных операций и вывести списки типов для каждой операции.

12. Сформировать пополняемую базу данных «Континент – страны», в которой указать столицы, численность населения, крупные города. Организовать поиск страны по городу, стран или городов на континенте.

13. В файле задан список книг (автор, название, год издания). Разработать программу выбора книг, выпущенных ранее заданного года.

14. Сформировать список граждан, в котором указать фамилию, имя, отчество, адрес, профессию. Организовать в программе выборку и подсчет граждан с одинаковой профессией.

15. Сформировать список студентов группы, в котором указать фамилию, имя, отчество, день, число и месяц рождения. Организовать выборки по месяцу и году рождения.

16. Дан список студентов группы. Заполнить его следующими сведениями: фамилия, имя, отчество, день, число и месяц рождения. Распечатать список в порядке возрастания дат рождения.

17. Сформировать базу данных «Единицы измерения», в которой указать название единицы, обозначение, назначение, соотношение. Организовать поиск по любому полю.

18. Разработать базу данных «Астрономия», в которой указать название звезды, величину, созвездие. Организовать поиск звезд по созвездию.

19. Разработать базу данных «Планеты Солнечной системы», в которой указать название планеты, величину, удаленность, спутники.

20. Сформировать базу данных «Химические элементы», в которой указать название элемента, формулу, группу, организовать выборки.

Задача 2 (программа12_2)

Дополнить программу12_1 заполнением и обработкой файлов. Имя файла вводить с клавиатуры в процессе работы программы.

Задача 3 (программа12_3)

Реализовать программу9_2 с возвратом нескольких значений из функции через поля структуры.

3.3.3. Поля битов

В отличие от других языков программирования C/C++ обеспечивает доступ к одному или нескольким битам в байте или слове. Если переменные принимают только два значения (например, логические), можно использовать **один** бит. Такие переменные называют **флагами**.

Доступ к биту обеспечивают **поля битов** (bit fields) – это специальный тип элементов структуры, в котором определено, из скольких битов состоит каждый элемент. Полем считается последовательность соседних двоичных разрядов в числе типа int или unsigned (signed). Оно объявляется как элемент структуры.

Основная форма объявления структуры битовых полей:

struct < имя структуры >
{ <тип имя 1>: <ширина>;
…
<тип имя N>: <ширина>**;**
};

где <ширина> – целое число от одного до 16; <тип> – ключевые слова int или unsigned. <Имя> может отсутствовать, тогда отведенные биты не используются (пропускаются). Длина структуры всегда кратна восьми.

Пример 1. Для переменной obj будет выделено восемь бит, но используется только первый.

```
struct onebit
        {
                unsigned b: 1;
        }obj;
```

Пример 2

```
struct M{                       // значения диапазонов полей
        int a:4;                //  a[-8, 7]
        int b:1;                //  b[-1, 0]
        unsigned c:5;           //  c[0, 31]
        int :2;                 //  пусто
        int d :2;               //  d[-2, 1]
        unsigned e :2           //  e[0, 3]
}
```

Операции для полей битов такие же, как и для структур:

– присваивание: **struct M byte;** то **byte.a = 4;**
byte.b = 0;

– доступ к полю **(**точка**)**: **byte.c = 20;**

Не допускаются массивы полей битов, указатели на поля битов и функции, возвращающие поля битов. К битовым полям **не может** применяться операция «**&**» (адрес).

Обычно поля битов используются, когда необходимо установить несколько объектов (полей структуры) в одно машинное слово.

В структуре могут быть смешаны обычные переменные и поля битов. Поля битов не могут располагаться на пересечении границ объявленных для них типов (они располагаются в оставшемся пространстве предыдущего поля или начиная с новой целой границы).

3.3.4. Объединения

Объединение – это структура для хранения значений различных типов в одной и той же области памяти, но не одновременно (все элементы ее имеют нулевое смещение, а сама структура достаточно велика, чтобы вместить самый большой элемент).

Объявляется объединение ключевым словом **union**:

union <имя структуры>{
<тип 1> <имя 1>;
<тип 2> <имя 2>;
. . .
<тип N> <имя N> **;**
}

Для объявления переменной записывается:

union <имя объединения> <имя переменной>;

В объединении не допускаются элементы типа полей битов.

Можно объявлять переменные (как и в структуре) одновременно с заданием шаблона.

Для переменной типа union выделяется столько места в памяти, сколько необходимо элементу объединения наибольшего размера в памяти. Остальные переменные будут располагаться в том же месте памяти, начиная с одного и того же адреса (с наложением).

При инициализации объединения задается значение первого элемента в соответствии с его типом.

```
union u    {    char name [10];
                int  t;
           }    u1 = "Объединение";
```

Объединения могут входить в структуры и массивы, и наоборот.

Ссылки на объединения можно использовать так же, как и на структуры.

Объединения обычно применяются для экономии памяти и в случаях, когда надо выделить часть целой переменной или загрузить целое значение в структуру, описанную полями битов.

3.3.5. Перечисления

Это множество поименованных целых констант, называемых перечислимыми константами. Перечислительный тип, созданный программистом, определяет все допустимые значения, которые могут иметь переменные этого типа. Основная форма объявления:

enum <имя типа> **{список названий}** [список переменных];

(список переменных может быть пустым).

Пример: **enum** test{test1, test2, test3,test4}; // типа
enum test **t**; // переменной **t** типа test.

Каждое из имен test1,… ,test4 эквивалентно целой величине (если они не определены по-другому). По умолчанию они соответствуют номерам **0, 1, 2, 3**.

Во время объявления типа можно одному или нескольким перечисленным константам присвоить другие значения (константными выражениями).

Пример. **enum** Ages **{**stas = 18, ira, alex = 19, Nina = alex-1**}**;

Когда нет явного инициализатора, то применяется правило по умолчанию: каждая следующая перечислимая константа увеличивается на единицу (+1) по сравнению с предшествующей. В примере вычисляется ira=19; Nina=18.

С переменными перечислимого типа можно проводить следующие **операции**:

– присвоить одну переменную типа enum другой переменной того же типа;

– провести **сравнение** с целью выяснения равенства или неравенства;

– некоторые **арифметические** операции с константами типа enum.

Например: i = test4 - test2;

Оператор **print f**(''%d %d", test1, test4); выдаст на экран числа 0 и 3.

```
t = test3;
t++ ;          //  теперь t= test4;
t -=2 ;        //  t = test2;
```

С перечислениями можно работать как с целыми типами. Перечисления неявно могут преобразовываться в обычные целочисленные типы, но не наоборот.

Нельзя использовать другие арифметические операции. Перечислимые константы могут быть объявлены **анонимно** (без имени), например: **enum {false, true} boolean;** объявляет переменную boolean с допустимыми значениями false, true.

Основное назначение перечислений – улучшить читаемость программы.

3.3.6. Переименование типов typedef

Язык C/C++ позволяет дать **новое название** уже существующим типам данных. Для этого используется ключевое слово **typedef** (новый тип не создается):

typedef < имя ранее определенного типа >< имя нового типа1>
[,<имя нового типа2>…];

Новое имя становится синонимом ранее определенного имени.

Например:

typedef float real; // теперь вместе float можно использовать real
typedef char symbol;

Часто используется для переопределения структур.

Область действия зависит от расположения оператора typedef. Если определение находится внутри функции, то область действия локальна и ограничена этой функцией. Если вне, то глобальна.

С typedef может быть объявлен любой тип, включая указатели, функции и массивы, структуры и объединения.

Пример 1

typedef char arr [40]; // FIO – массив символов
arr FIO, *adres; // adres – указатель на массив символов

Это эквивалентно char FIO[40], *adres;

Пример 2

typedef int* Pi; // объявлен новый тип Pi – указатель на целое;

typedef void (*pfn) ( ); // объявлен новый тип pfn – указатель на функцию, не возвращающую значения, с любым списком типов аргументов;

typedef void (*pfnI) (int); // объявление типа pfnI – указатель на функцию с одним аргументом типа int, не возвращающую значения;

typedef void (*pptn[10]) (); //объявление типа pptn – массив из 10 указателей на функцию, не возвращающую значения, с любым списком аргументов.

3.4. Модульное программирование. Структура сложной программы и время жизни переменных

Реальный программный продукт состоит из множества исходных файлов, которые могут компилироваться раздельно, а затем объединяются в исполняемую программу редактором связей.

При такой структуре функции из разных файлов могут использовать глобально доступные внешние переменные. Сами функции по определению всегда внешние и доступны из любых файлов.

Структура программы, состоящей из двух исходных файлов, может иметь следующий вид:

```
file1.c                          //  первый файл    (file1.cpp)
директивы препроцессора
# include <включения>            // подключаемые библиотеки
# include "file2.c"              // подключаемые файлы
# define                         // макроопределения
Описание переменных на внешнем уровне
    int main()                   или void main()  Главная функция
       { . . . }
    int func1()                  //  функция 1
       { . . . }
file2.c                          //  второй файл    (file2.cpp)
Описание переменных на внешне уровне
float func2()                    //  функция 2
{ . . . }
 int func3()                     //  функция 3
{ . . . }
```

Функция main() может обратиться к любой из трех функций. Каждая из них может вызвать любую из двух оставшихся. Связь между функциями осуществляется как через передаваемые параметры, так и через глобально доступные внешние переменные.

Если внешняя переменная второго файла определена классом **extern** (ссылкой) на внутреннем уровне в любой функции первого файла, то она станет видимой в той функции, где есть на нее ссылка.

Переменная, объявленная спецификатором класса памяти extern, является ссылкой на переменную с тем же именем, но определенную на внешнем уровне в любом исходном файле программы. Цель внутреннего объявления extern – сделать определение переменной внешнего уровня видимым внутри блока. Если объявления на внешнем уровне нет, пере-

менная класса extern видима только в блоке, в котором она объявлена, т. е. является локальной. Переменные, определенные классом памяти extern, не могут инициироваться на внутреннем уровне.

Статические переменные могут инициализироваться константой или константным выражением. Если явной инициализации нет, то статическая переменная устанавливается в нуль автоматически. Инициализация выполняется один раз и не повторяется при повторном входе в блок.

При объявлении переменных на внешнем уровне не допускается применение спецификаторов классов auto и register. Объявление переменных на внешнем уровне – это или определение переменных (выделение памяти и, возможно, инициализация), или ссылка на определения, сделанные в другом месте.

Переменная, определенная на внешнем уровне, видима в пределах остатка исходного файла (в котором определена). Она невидима ни выше своего определения в этом файле, ни в других файлах программы, если не объявлена ссылка, которая сделает ее видимой.

Любая из внешних переменных должна определяться лишь один раз, другие файлы (функции других файлов) могут содержать для обращения к переменной объявления класса **extern**. Определение может быть сделано в любом из исходных файлов, составляющих программу.

Если переменная, объявленная внутри блока, имеет то же имя, что и внешняя (или из объемлющего блока), то определение переменной в блоке заменяет определение объемлющего уровня на протяжении всего блока. Видимость внешней переменной (или из объемлющего блока) восстанавливается при выходе из блока.

Функции всегда определяются на внешнем уровне и могут объявляться на внешнем и на внутреннем уровнях со спецификатором класса памяти **static** или без спецификатора.

Пример разработки программ, состоящих из нескольких файлов

Разработать программу, вычисляющую соотношение между максимальными элементами для массивов A и B.

Выделим вычисление наибольшего элемента в подпрограмму, которую вынесем в отдельный модуль.

```
#include <stdio.h>
#include <conio.h>
#include "r_pos.h"  // подключение модуля из текущего каталога
int N;
void main()
{
        int a[8], b[8];
        int  s;
        printf("Введите количество элементов в массиве (не более 8) \n");
        scanf("%d", &N); // или scanf_s      – ввод размера массива N
        printf("Введите массив A из %d чисел типа int \n", N);
        for ( i = 0; i < N; ++i )
                scanf("%d", &a[i]);      // или scanf_s
        printf("Введите массив B из %d  чисел типа int \n", N);
        for (i=0; i< N; ++i)
                scanf("%d", &b[i]);      // или scanf_s
        printf("Сообщение=");
        printf(" max(A)/max(B) = %6.3f \n", (float)max(a)/max(b));
        __getch();
}
// Интерфейсная часть модуля r_pos
typedef int * mass;
int max(mass d);
extern int N;
// Исполняемая часть модуля
int max(mass d)
{
```

```
    int  mx;
    mx = d[0];
    for (i=1; i< N; ++i)
        if (mx<d[i]) mx = d[i];
    printf("%d:", mx);
    return mx;
}
```

Задание 13 для разработки задач в виде программ, состоящих из нескольких файлов

Задача 1 (программа13_1 и 13_2)

Для программ8_3 и 9_2 разработать новые модификации, скрыв подпрограммы в отдельном файле (модуле), при этом использовать ссылки на переменные, объявленные в другом модуле.

Задача 2 (программа13_3)

1. Составить подпрограмму для перемножения двух квадратных матриц, с помощью которой вычислить вторую, третью и четвертую степени матрицы M[5][5]. Для каждой матрицы вычислить сумму и среднее арифметическое значение элементов.

2. Даны массивы A[6], B[6], C[6], вычислить суммы положительных и отрицательных элементов. Получить A*B, B*C, A*C.

3. Даны две матрицы целых чисел V[2][3], W[3][2]. Найти суммы элементов строк и столбцов в этих матрицах. Сравнить общие суммы элементов в соответствующих матрицах, вычисленные через суммы строк и суммы столбцов.

4. По заданным вещественным массивам A[6], B[6] и C[6] вычислить

$$Y = \begin{cases} (\max B) / \min B + (\max (C + B)) / \min(B + C) & \text{при } \min A < \max B, \\ \max(B + C) + \max A / \min A & \text{– в противном случае.} \end{cases}$$

5. Даны две матрицы целых чисел V[2][3], W[2][3]. Найти общие суммы элементов строк, из функции вернуть две суммы.

6. Даны массивы A[5], B[6]. Получить новые массивы путем сдвига элементов в массивах на два разряда вправо, освободившиеся слева элементы обнулить. Сдвиг элементов в массиве оформить подпрограммой, из подпрограммы вернуть выпавшие справа разряды.

7. Даны две целые квадратные матрицы четного порядка. Элементы массивов с четными номерами строки и столбца заменить нулем (стереть). Напечатать исходные и полученные массивы. Подсчитать количество четных и нечетных чисел в каждом.

8. Вычислить значение функции $Z = x1 + e^{x2}$, где $x1$, $x2$ – корни уравнения $A_i x^2 + B_i x + C_i = 0$, где $i = 1, 2, ..., N$. Коэффициенты уравнения заданы в массивах A[N], B[N], C[N]. Для вычисления корней использовать подпрограмму.

9. Даны две целые квадратные матрицы шестого порядка. Распечатать элементы главных диагоналей каждой из них и вычислить суммы элементов, попадающих на главную диагональ и отдельно на побочную.

10. Даны два одномерных массива A[6] и C[6]. Получить A^2, C^2, A*C. Подсчитать число четных и число нечетных чисел в новых массивах. Перемножение массивов и подсчет выполнить в подпрограмме.

11. Даны массивы A[4][4], B[4][4]. Получить новые массивы путем сдвига элементов в массивах на два разряда вправо, освободившиеся слева элементы обнулить. Сдвиг элементов в массиве оформить подпрограммой, из подпрограммы вернуть суммы элементов выше главной диагонали и отдельно ниже.

12. Даны три одномерных массива вещественных чисел A[6], B[8] и C[7]. Найти среднее геометрическое значение положительных элементов и номера ближайших к ним элементов для каждого массива.

13. Даны две матрицы целых чисел S[3][2], K[3][2], в каждой из которых имеется по два одинаковых числа. Распечатать это значение и индексы.

14. Даны массивы A[6], B[6], C[6]. Преобразовать их, каждому элементу массива присваивая значение соседнего с ним справа. Последнему элементу присвоить значение первого. Напечатать исходные, результирующие массивы и отдельно последние два элемента результирующего.

15. По заданным элементам вещественных матриц A, B и C вычислить

$$Z = \begin{cases} (\max(b_i)) / \max(a_i) + (\max(c_i)) / \max(b + c)_i & \text{при } \min A_i < \min B_i, \\ \min(b + c) + \min(c_i) + \min(a_i) & \text{при } \min A_i > \min B_i. \end{cases}$$

16. По заданным вещественным массивам A[6], B[6] и C[6] вычислить

$$Z = (\min A_i) / \max A_i + (\max C_i) / \min(C_i) + \max(B + C)_i / \min(B + C)_i.$$

17. Даны две целые квадратные матрицы четного порядка. Напечатать массивы, транспонированные матрицы, количество четных и число нечетных чисел в каждой.

18. Даны массивы A[6], B[8]. Выбрать из них положительные элементы и записать соответственно в массивы A[k] и B[k], где $k < 6$, $n < 8$; из отрицательных элементов сформировать массивы A2[6-k], B2[8-n]. Напечатать суммы и произведения элементов для каждого.

19. Даны квадратные матрицы A, C и B размером 4×4 каждая. Напечатать средние арифметические значения в них отдельно для положительных и отрицательных элементов.

20. Даны массивы целых чисел F[8], B[8]. Найти и напечатать значения и индексы минимальных элементов. Определить в каждом массиве количество таких же элементов.

4. ДИНАМИЧЕСКАЯ ПАМЯТЬ

Статические переменные занимают место в памяти компьютера в течение всего времени работы программы. Для динамических переменных можно по мере необходимости резервировать место в свободной памяти, а когда какая-либо переменная уже не нужна, то фрагмент памяти, отведенный для нее, освобождается.

Область памяти, в которой располагаются динамические переменные, называется *динамически распределяемой областью памяти* или хипом (heap – куча).

4.1. Резервирование и освобождение памяти в стандарте языка С

Функции резервирования и освобождения памяти в стандарте языка С помещены в библиотечный файл **stdlib.h**:

void *malloc (size_t size); – выделяет память и возвращает указатель на первый байт свободной памяти требуемого размера,

void * free (void *p); – освобождает ранее выделенную область памяти.

void *calloc (size_t n, size_t b); – выделяет место в хипе для n элементов по b байтов каждый.

void *realloc (void * block, size_t size); – делает попытку установить новый размер **size** блока, на который указывает **block**.

Приведем небольшой пример динамического распределения памяти:

```
# include <stdio.h>
# include <stdlib.h>
void main()
{
    int t, *p;
    printf("\n Динамич. распределение памяти\n");
    p = malloc(10*sizeof(int));  //Выделение памяти для 10 целых чисел
```

```
    if (!p)
    {
    printf("Недостаточно памяти \n");
    exit(1);
  }
  for (t=0; t<10; ++t) *(p+t)=t;    // Использование памяти
  for (t=0; t<10; ++t)          printf(" %d", *(p+t));
  free(p);                              // Освобождение памяти
}
```

4.2. Операторы динамического распределения памяти в C++

Оператор **new** выделяет блок памяти для размещения переменной или массива (необходимо указывать тип и, в случае массива, размерность), при этом можно присваивать вновь созданной переменной начальное значение.

new type_name [(инициатор)**];** или **new (type_name [**(инициатор)**]);**

Оператор **delete** освобождает ранее выделенную память. Размер занятого блока для правильной работы delete записывается в его начало и обычно занимает дополнительно 4 байта.

Следует помнить, что реальный размер занятого блока не произволен, а кратен определенному числу байтов (в C++3.0 – 16), поэтому с точки зрения расхода памяти невыгодно резервировать много блоков под небольшие объекты.

В случае успешного выполнения **new** возвращает адрес начала занятого блока памяти, увеличенный на количество байтов, занимаемых информацией о размере блока (т. е. возвращает адрес созданной переменной или адрес нулевого элемента созданного массива). Когда new не может выделить требуемую память, он возвращает значение (void *), в этом случае рекомендуется предусмотреть в программе реакцию. Например:

```
# include <stdio.h> //
# include <iostream.h>                    // для ввода/вывода  в C++
int main()
{
    int *u_i;
    double *u_d;
u_i=new int;       //Зарезервировать место под переменную типа int и
                   //присвоить u_i ее адрес (Значение *u_i не определено)
u_d=new double(3.1415);// -"- и *u_d инициализируется знач. 3.1415
if (!(u_i && u_d))
    {
       cout<<"Не хватает памяти для всех динамически"
             "размещаемых переменных!";
       return 1;
    }
cout<<"\n u_i="<<u_i<<" случайное значение *u_i="<<*u_i ;
delete u_i; //Освободить блок памяти, на который указывает u_i
cout<<"\n u_d="<<u_d<<" *u_d="<< *u_d ;
delete u_d; //Освободить блок памяти, на который указывает u_D
return 0;
}
```

Если указатель, на который действует оператор delete, не содержит адреса блока, зарегистрированного ранее оператором new, или же не равен NULL, то последствия будут непредсказуемыми.

Помимо проверки возвращаемого значения в C/C++ для обработки ситуации нехватки памяти программист может определить специальную функцию обработки, которая будет вызываться при неудачном выполнении оператора *new* и «пытаться» изыскать необходимую память либо предусмотреть сообщение и завершить выполнение программы одной из библиотечных функций exit или abort.

4.3. Динамические структуры данных

Одновременно с применением динамического распределения памяти возникла необходимость в разработке способов управления этой памятью. Динамические переменные чаще всего реализуются как связные структуры.

4.3.1. Связанные динамические данные. Списки

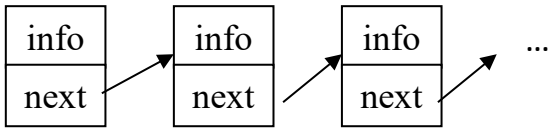
Список – это набор связанных между собой динамических элементов.

Обращение к динамической переменной происходит через ссылочную переменную, которая содержит адрес такой переменной. Под ссылочную переменную компилятор отводит место в памяти; эта переменная имеет имя и явно упоминается в программе. Ссылочные переменные образуют **тип данных – ссылки** или **указатели**.

Для организации связей между элементами динамической структуры данных требуется, чтобы каждый элемент содержал кроме информационного значения как минимум один указатель. Следовательно, в качестве элементов таких структур необходимо использовать структуры, которые могут объединять в одну переменную разнородные элементы.

Динамические переменные, как правило, имеют тип «**структура»**, так как должны содержать не менее двух полей: помимо значения (целого, вещественного), ссылку на другую динамическую переменную связанной структуры.

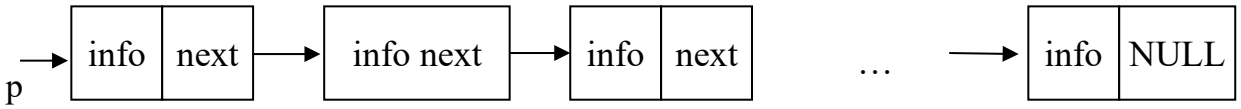
Схематично такую структуру данных можно показать следующим образом:



В простейшем случае элемент динамической структуры данных должен состоять из двух полей (информационного и указательного) и описывается типом «структура».

```
struct L
        { int info;
        struct L *Next;
        }
```

Рассмотренная структура называется цепочкой и является частным случаем динамической структуры, называемой в программировании **линейным однонаправленным** или **односвязным списком**, где каждый элемент, входящий в очередное звено списка, снабжается ссылкой на следующее за ним звено:



В списке предусмотрено заглавное звено. Указатель списка (p), значением которого является ссылка на заглавное звено, представляет список как единый объект.

Линейные списки – это данные динамической структуры, представляющие собой совокупность линейно-связанных однородных элементов, для которых разрешены следующие действия:

1) добавление элемента в начало;

2) добавление элемента в конец (хвост) списка (частный случай вставки);

3) вставка элемента между двумя любыми другими элементами;

4) исключение элемента (удаление любого, как крайнего, так и среднего элемента списка).

Для вставки элемента необходимо:

а) создать новый динамический элемент, которым будет представлено новое звено;

б) в ключевое поле key внести значение;

в) в поле next занести ссылку, взятую из поля next того звена, за которым должно следовать вставляемое;

г) в поле next звена, за которым следует вставка, занести ссылку на это новое звено.

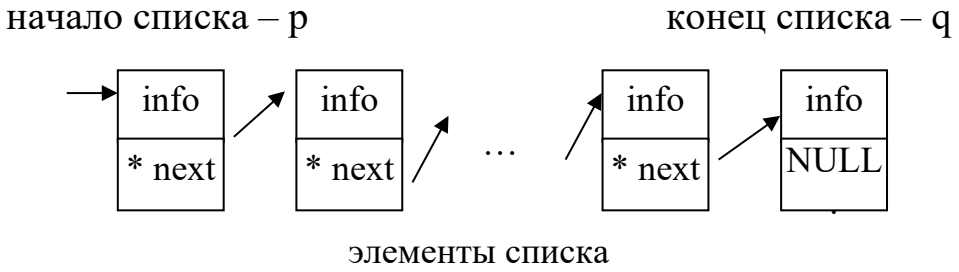
Таким образом, для вставки нового компонента в цепочку данных достаточно изменить две ссылки.

Для исключения элемента достаточно изменить ссылку у предшествующего исключаемому элемента на ссылку, содержащуюся в исключаемом элементе. При исключении элементов необходимо освободить соответствующий участок памяти.

Для работы со связными структурами были разработаны 3 основные динамические структуры, объединенные общим термином **списки**.

Список – это набор динамических элементов (чаще всего переменных типа «структура»), связанных между собой каким-либо способом. В зависимости от вида связей списки бывают линейными, кольцевыми, односвязными и двусвязными, нелинейными и т. д.

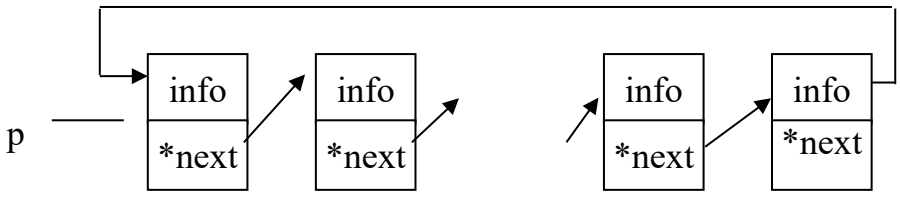
Линейный односвязный список – это список, каждый элемент которого связан только с одним элементом, следующим за данным. Сам элемент по отношению к следующему называется предшествующим или предыдущим. Последний элемент списка содержит признак конца списка **NULL**.



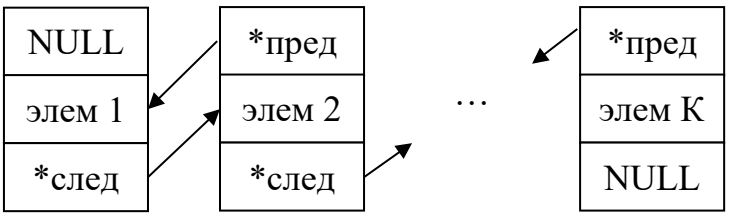
элементы списка

Обязательно должен быть указатель на заглавное звено.

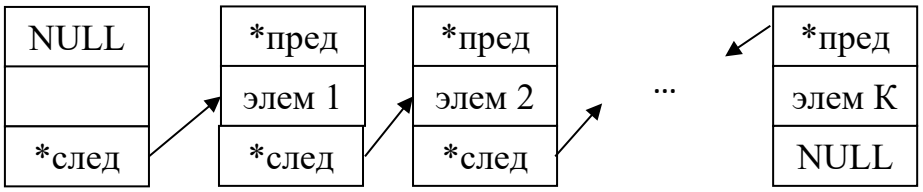
Кольцевые списки – это такие же динамические данные, как и линейные списки, но имеющие дополнительную связь между последним и первым элементами списка:



Линейный двусвязный (двунаправленный) список отличается от односвязного тем, что в нем каждый элемент связан не только со следующим элементом, но и с предыдущим. И первый, и последний элементы списка содержат признак конца списка NULL:

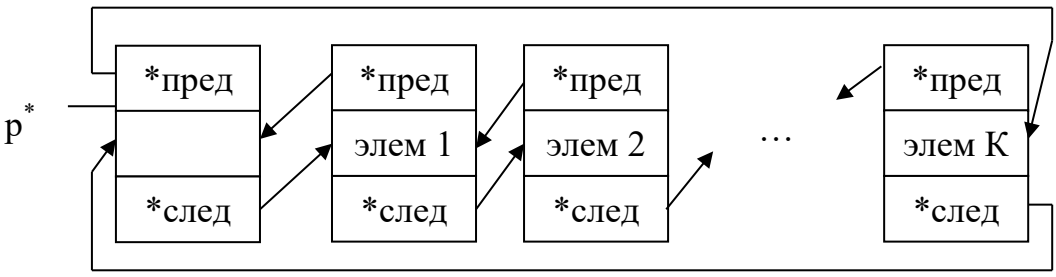


В линейном двусвязном списке заглавное звено может не включать информационного поля:



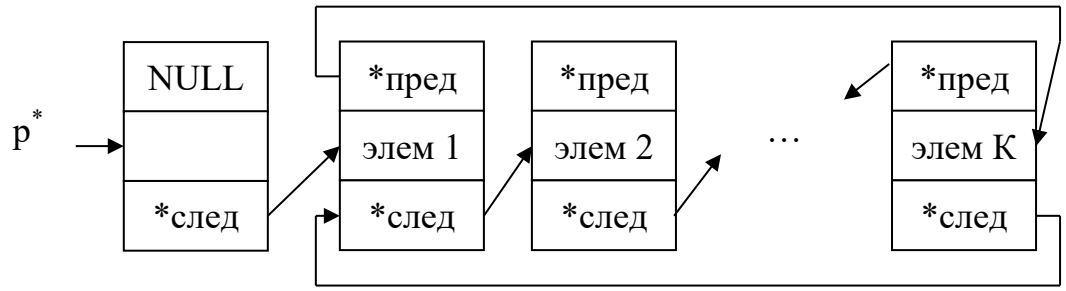
Информационное поле заглавного звена либо вообще не используется, либо может служить для хранения признака наличия заглавного звена и некоторой информации о списке в целом (количестве звеньев).

В двунаправленном кольцевом списке элементом, следующим за последним, считается заглавный или первый, а элементом, предшествующим заглавному или первому, считается последний:



Чтобы сослаться на двунаправленный кольцевой список как на единый программный объект, используют указатель, значением которого является ссылка на заглавное звено списка.

Рассмотренный выше способ образования двунаправленного списка не является единственно возможным: можно не включать заглавное звено списка в кольцо:



Каждый метод имеет свои достоинства и недостатки, в частности при вставке элементов в конец или начало списка.

Над списками определены те же основные операции, что и над цепочками:

– поиск элемента в списке;

– вставка заданного элемента в указанное место списка;

– удаление из списка заданного элемента.

Поиск элемента производится перебором элементов с помощью ссылок, содержащихся в каждом звене.

Для **вставки** элемента порождается новое звено, значение заносится в информационное поле, а в новое звено заносятся ссылки в поле *ПРЕД из следующего за вставляемым звеном и в поле *СЛЕД – из предшествующего звена. Кроме того, необходимо скорректировать ссылки в звеньях, между которыми выполняется вставка (на новое звено).

Удаление элемента. Для исключения элемента достаточно изменить ссылку в поле *СЛЕД у предшествующего ему звена и ссылку в поле *ПРЕД у звена, следующего за исключаемым.

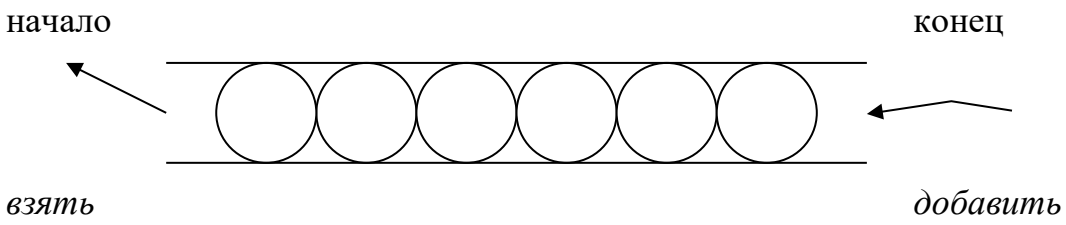
4.3.2. Очередь

Самыми распространенными случаями линейного односвязного списка являются очередь и стек.

Очередь – это частный случай линейного односвязного списка, для которого разрешены только два действия: добавление элемента строго

в конец (хвост) списка и извлечение (удаление) строго из начала (головы) очереди.

Очередь можно представить в виде трубы с двумя открытыми концами, куда помещаются «бочонки» – элементы. Движение по трубе возможно лишь в одном направлении:



Очередь – динамическая структура или дисциплина обслуживания, организованная по принципу **FIFO** (first in – first out). "**Первым пришел – первым ушел**". Основные операции для работы с очередями: добавление нового элемента в конец очереди (занесение заказа на обслуживание); удаление элемента из начала очереди для его обслуживания, при этом выбранный элемент, естественно, исключается из очереди. Таким образом, в очереди доступны только ее начало и конец.

Для создания очереди и работы с ней необходимо иметь как минимум два указателя: на начало очереди и на конец очереди. Кроме того, для освобождения из памяти удаляемых элементов и удобства работы с очередью требуется дополнительный временный указатель.

Функции, которые могут быть использованы при работе с очередями:

1) вставка элемента (добавить в очередь элемент); при этом отводится память под очередной элемент, в который заносится нужная информация, и элемент ставится в конец очереди;

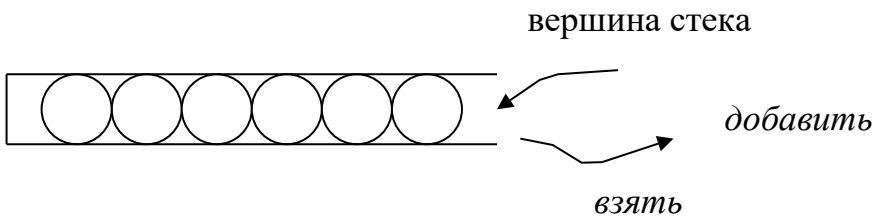
2) удаление из очереди ее первого элемента; при этом выполняется освобождение участка в памяти и перемещение указателя начала очереди на следующий элемент.

4.3.3. Стек

Наиболее часто встречающаяся динамическая структура данных – **стек** (магазин) – отличается от очереди тем, что она организована по принципу **LIFO** (last in – first out): **«последним пришел – первым ушел»**.

Стек – это частный случай линейного односвязного списка, для которого разрешено добавлять или удалять элемент только с одного конца списка – вершины (головы) стека.

Стек можно представить в виде трубы с одним запаянным концом, куда помещаются «бочонки» – элементы:



Операции включения и удаления элемента в стеке выполняются только с одного конца, называемого вершиной стека.

Если число элементов не может превышать некоторой величины, то стек называется **ограниченным**, максимальное число элементов в нем – это **глубина стека**. Стек, в котором нет ни одного элемента, называется **пустым**.

Стек можно представить в виде динамической цепочки звеньев. Вершиной является первое звено цепочки (или последнее), заглавное звено становится излишним. Поэтому значением указателя, представляющего стек как единый объект, является ссылка на вершину стека. Каждое звено содержит ссылку на следующее звено цепочки, причем «дно» стека имеет ссылку NULL:

Если стек пуст, то значением указателя p является ссылка NULL. К началу использования стека его необходимо сделать пустым (p = NULL).

Операции над стеком:

1) занести элемент в стек;

2) посмотреть, что находится на вершине стека (не удаляя);

3) выбрать элемент из стека. При этом элемент, находящийся в вершине стека, должен быть присвоен в качестве значения некоторой переменной, а звено, в котором был представлен этот элемент, должно быть исключено из стека.

Пример программирования задач со списком

Задать списком строку, заканчивающуюся точкой. Напишем программу подсчета числа цифр в строке:

```
#include <conio.h>
#include <stdio.h>
#include <locale.h>
#include <iostream>
#include <malloc.h>
typedef struct zv
{
    char info;
    struct zv * next;
} Zv;
void main()
{
  setlocale(LC_ALL, "RUS");
  Zv * P;                          // указатель на начало
   Zv * t;                         // текущий указатель
  char sym;
  int k;
  system("cls");
printf("Введите строку, оканчивающуюся точкой: ");
  P = NULL;
  do                               // заполнение стека
                  {
```

```
            sym =_getche();
                t = (Zv *) malloc(sizeof(Zv));        //  можно t = new(Zv)
                t->nex t= P;
                t->info = sym;
                P = t;
            }
        While (sym != '.');
            k = 0;
            t = P;
            while ( t != NULL )
        {
            if ( t->info >= '0' && t-> info <= '9' ) k = k+1;
            t = t->next;
        }
        printf("\n");
        printf("Цифр в строке - %d",k);
  _getch();
}
```

Задание 14 для программирования задач с динамическими структурами

Задача 1 (программа14_1)

1. Используя динамическую структуру «список», подсчитать количество русских строчных букв в строке.

2. Используя динамическую структуру «список», проверить, является ли он упорядоченным набором чисел.

3. Используя динамическую структуру «список», подсчитать количество цифр в заданном наборе символов.

4. Сформировать динамический список из элементов целого типа. Удалить из него отрицательные элементы. Распечатать исходный и результирующий списки.

5. Найти сумму четных элементов списка, состоящего не менее чем из двух элементов.

6. Используя динамическую структуру «список», подсчитать сумму чисел в нем.

7. В динамическом списке из каждой группы подряд идущих одинаковых элементов оставить один.

8. Используя динамическую структуру «список», подсчитать количество четных чисел в заданном наборе символов.

9. Используя динамическую структуру «список», подсчитать сумму отрицательных чисел в списке.

10. Используя динамическую структуру «список», подсчитать количество положительных и отрицательных чисел в списке.

11. Задать два динамических списка. Проверить их на равенство.

12. Используя динамическую структуру «список» для хранения символов, напечатать только русские буквы из текста.

13. Используя динамическую структуру «список» для хранения элементов целого типа, найти максимальное значение.

14. Сформировать динамический список из элементов целого типа. Определить, есть ли в списке хотя бы два совпадающих по ключевому полю элемента.

15. Удалить из списка все отрицательные элементы.

16. Дан список целых чисел. Выполнить циклический сдвиг элементов списка влево.

17. Построить динамический список из элементов целого типа с помощью датчика случайных чисел. Найти значение среднего элемента списка.

18. Используя динамическую структуру «список», записать ряд чисел. Удалить элемент с заданным ключевым полем.

19. Сформировать список динамических элементов, упорядоченный по возрастанию. Включить в список новый элемент, сохранив свойство упорядоченности.

20. Дан список целых чисел. Изменить список по правилу: если текущий и следующий элементы списка четны, в текущий элемент записать сумму чисел, а следующий удалить.

Задача 2 (программа14_2)

1. В деке задать слово. Определить, является ли оно словом-перевертышем.

2. Сформировать очередь из элементов целого типа. Четные элементы возвести в квадрат. Распечатать исходную и результирующую очереди.

3. С использованием динамической структуры «очередь» перевести введенную последовательность чисел в слово, состоящее из кодов ASCII.

4. Сформировать динамическую структуру «очередь», элементами которой являются цифры. Извлекая элементы из очереди, печатать их двоичные эквиваленты.

5. Используя динамическую структуру «стек», напечатать элементы в обратном порядке.

6. С использованием динамической структуры «очередь» зашифровать содержимое текста: каждый символ заменить его кодом, увеличенным на единицу.

7. С использованием динамической структуры «стек» зашифровать содержимое текстового файла заменой символом, код которого увеличен на два.

8. С использованием динамической структуры «стек» для хранения символов определить, является ли данная последовательность символов палиндромом.

9. Сформировать очередь из *n* случайных элементов целого типа. Для каждого элемента вывести, является он четным или нечетным.

10. Дана очередь, элементами которой являются имена студентов. Написать программу, проверяющую присутствие конкретного студента в очереди (проверка на вхождение значения во множество).

11. Дан стек, в котором записаны названия книг. Необходимо вынимать элементы из стека до того момента, пока не будет найдена книга, название которой было введено пользователем с клавиатуры. На экран вывести номер элемента, где находится название книги.

12. Дан стек, в котором записана последовательность целых чисел. Найти и вывести на экран максимальное значение из данного стека.

13. Сформировать динамическую структуру «дек», элементами которой являются символы. Написать программу, которая меняет местами первый и последний элементы дека, выполнить сдвиг элементов вправо на три символа. Распечатать исходный и результирующие деки.

14. Дана последовательность целых чисел в виде очереди. Необходимо разделить ее на две очереди. Одна очередь должна содержать положительные элементы, другая – отрицательные. Вывести содержимое обеих очередей.

15. Дана очередь, элементами которой являются строчные латинские буквы (могут повторяться). Необходимо удалить все повторяющиеся (оставить только различные).

16. Дан дек с нечётным количеством элементов. Необходимо вывести центральный элемент дека и все элементы после него.

17. С использованием динамической структуры «стек» сохранить ряд букв латинского алфавита. При извлечении элементов из стека вывести только гласные буквы.

18. С использованием динамической структуры «очередь» подсчитать количество цифр в заданном наборе символов.

19. Дана очередь, элементами которой являются буквы и цифры. Сформировать новую, при этом удваивать цифровые значения.

20. Дан дек с нечётным количеством элементов. Необходимо переставить элементы относительно центрального элемента дека. Распечатать полученный дек.

5. РЕКОМЕНДАЦИИ ДЛЯ ИЗУЧЕНИЯ ДИСЦИПЛИНЫ «ПРОГРАММИРОВАНИЕ»

Для изучения дисциплины «Программирование» учебным планом предусмотрены лекции (два семестра), практические занятия (один семестр), лабораторные работы (два семестра), домашние задания и расчетно-графические работы.

5.1. Порядок выполнения лабораторных работ

При выполнении лабораторных работ рекомендуем придерживаться следующего порядка:

1) изучить теоретические сведения по теме;

2) построить схему алгоритма для решения задач для каждого задания в соответствии с вариантом, выданным преподавателем;

3) подготовить текст программы и набрать его в окне редактирования;

4) откомпилировать программу, исправить синтаксические ошибки;

5) выполнить программу для реальных исходных данных;

6) просмотреть результаты и записать в тетрадь;

7) оформить отчет по текущей лабораторной работе, в который включить:

– номер работы (например: лабораторная работа № 1),

– тему работы,

– условия задач,

– схемы алгоритмов,

– тексты программ,

– результаты выполнения программ, выводы.

Защитить лабораторную работу. Ответить на вопросы преподавателя.

5.2. Задания
для выполнения расчетно-графической работы

Разработать схему алгоритма, написать и отладить программу для расчета и построения графиков двух функций (результаты расчетов должны храниться в виде массивов и распечатываться в виде таблицы) [2], выделить наибольшее и наименьшее значения для каждой из функций. Добавить в программу решение нелинейного уравнения и вычисление приближенного значения определенного интеграла по заданию преподавателя.

Разработать программу нахождения корней уравнения $f(x) = 0$ на интервале $[a, b]$ с точностью $e = 0.001$ (интервал подобрать или рассчитать самостоятельно). При реализации можно использовать метод половинного деления (бисекции) или метод хорд [2]. Рекомендуется использовать следующие варианты заданий:

1. $x - 2e^{-x} - \sqrt{x} = 0$.
2. $x^2 - \ln(x + 1) + \sin x - 2 = 0$.
3. $5x - e^x - 2x\sqrt{x} = 0$.
4. $2x \lg x - 3 = 0$.
5. $2^x - 3x - 2 = 0$.
6. $0{,}5 + \cos x - 2x \sin x = 0$.
7. $\sin x + x - 3 = 0$.
8. $x\, e^x - 2x^2 - 1 = 0$.
9. $\sqrt{\ln(x^2 + 3)} + 2x - 3 = 0$.
10. $2\ln(x + 1) - x + 1 = 0$.
11. $\operatorname{tg} x - e^{x+1} = 0$.
12. $2\ln(x + 1) + \operatorname{arctg} x - 3 = 0$.
13. $x^3 - 2x^2 - 4 = 0$.
14. $5x^2 - 2x \ln x - 7 = 0$.
15. $2\sqrt[3]{x^2 + 1} - 3\sin x = 5$.
16. $2x^3 - 3x^2 - 4 = 0$.
17. $4x - \sin 2x - 3 = 0$.
18. $2x \sin^2 x - 3\sqrt{x + \cos x} = 0$.
19. $2\ln(x + 1) - 3\sin 2x = 4$.
20. $x^3 + 3x + 2 = 0$.

Разработать программу для вычисления значения определенного интеграла на интервале [*a*, *b*] (*a*, *b* подобрать самостоятельно) численными методами прямоугольников и трапеций [8] для следующих вариантов:

1. $\int_a^b \sqrt[3]{x+1}dx$
2. $\int_a^b \sqrt{x^2+3}dx$
3. $\int_a^b \frac{1}{(\sqrt{2+x^3})}dx$
4. $\int_a^b x^2 e^{-x}dx$
5. $\int_a^b x^2 tgdx$
6. $\int_a^b \frac{\cos}{(x^2+1)}dx$
7. $\int_a^b \cos x e^x dx$
8. $\int_a^b x^2 \ln xdx$
9. $\int_a^b \frac{\lg(x+1)}{(1+x^2)}dx$
10. $\int_a^b e^{-x} \ln(x+1)dx$
11. $\int_a^b e^{2x} \sin 2xdx$
12. $\int_a^b \frac{\sin x}{(\sqrt{x^3+1})}dx$
13. $\int_a^b (e^x + e^{-x})dx$
14. $\int_a^b x^2 \operatorname{arctg}(x)dx$
15. $\int_b^a \frac{\cos x}{(\sin x + \cos x)}dx$
16. $\int_a^b \sqrt{x}\sin(x)dx$
17. $\int_a^b \frac{\operatorname{arctg}^2(x)}{x}dx$
18. $\int_a^b \frac{\ln(1+x)}{(1+x^2)}dx$
19. $\int_a^b \frac{1}{(1+\sqrt{\ln(x)})}dx$
20. $\int_a^b \frac{1}{(x+\sqrt{\cos x})}dx.$

В программе необходимо предусмотреть вывод сведений об авторе и заставку. Можно использовать следующие примерные задачи для заставок.

1. Подготовить и отладить программу вычерчивания квадрата, вписанного в окружность. Запрограммировать изменение цветов окружности.

2. Организовать вывод на экран разноцветных цифр поочередно горизонтально и вертикально.

3. Отладить программу вычерчивания нескольких окружностей заданного радиуса в произвольном месте экрана разными цветами. Предусмотреть их мерцание.

4. Разработать программу изображения точки, движущейся по косинусоиде ($y = \cos(x)$).

5. Отладить программу рисования квадрата с диагоналями, пересекающимися в центре экрана. Создать иллюзию его уменьшения.

6. Начертить на экране картинку «круги на воде».

7. Изобразить движущийся на зрителя экран (прямоугольник).

8. Подготовить и отладить программу изображения правильного треугольника, вращающегося вокруг любой медианы.

9. Разработать программу вывода на экран компьютера пяти квадратов по углам и в середине экрана. Запрограммировать поочередный вывод их разными цветами.

10. Разными цветами в различных точках экрана выводить надпись «C/C++» с изменением размера шрифта.

11. Изобразить трубу, образованную разноцветными окружностями.

12. Организовать вывод на экран разными цветами названий континентов в различных точках экрана готическим и обычным шрифтами.

13. Изобразить спутник (точку), вращающийся вокруг планеты.

14. Изобразить НЛО, приземляющийся в различных точках экрана.

15. Организовать вывод на экран разноцветных русских букв разных размеров, поочередно высвечивая или заполняя экран в хаотическом порядке.

16. Начертить графики функций $y = \sin(x)$ и $y = \sin(2x)$ разными цветами в разных осях координат.

17. Организовать вывод на экран формул химических элементов разными цветами в различных точках экрана.

18. Начертить на экране ряд геометрических фигур: квадрат, круг, треугольник, прямоугольник, используя разные цвета.

19. Изобразить схематично на экране движущегося человечка.

20. Изобразить движение объекта по прямоугольной спирали.

ЗАКЛЮЧЕНИЕ

В учебном пособии большое внимание уделено описанию основ языка C/C++, которые являются базовыми и необходимы для написания программ. Поскольку C/C++ – это язык с высокой степенью типизации, приведена классификация типов данных. Помимо функций для форматированного ввода и вывода данных различных типов, синтаксиса операторов определены особенности работы с одномерными и двумерными массивами, со структурами, символами и строками, с оформлением сложных программ, состоящих из нескольких файлов.

Отдельный раздел посвящен использованию динамической памяти, позволяющей резервировать память под переменные по мере необходимости и освобождать ее, если переменная уже не используется. Описаны динамические структуры данных: списки, очередь, стек – и принципы работы с ними, которые проиллюстрированы графически.

Рассмотрены теоретические сведения для изучения программирования на языке C/C++. Приемы программирования конкретных задач разделены на 14 тем, приведены подробные примеры для всех тем, а также задания для выполнения расчетно-графической работы – проектирования и отладки сложной программы, состоящей из нескольких подзадач.

Выполнение одного варианта для всех заданий в указанном порядке позволит освоить основы программирования на C/C++. С более подробными принципами проектирования алгоритмов и оформления их в виде схем можно ознакомиться в методических указаниях к практическим занятиям [7].

Для написания программ хорошего качества не достаточно знать синтаксис и правила выполнения операторов. Полезные практические советы по грамотному написанию эффективных программ содержатся в источниках [3, 4, 6, 8].

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. ГОСТ 19.701–90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения : утв. и введ. в действие Постановлением Гос. комитета СССР по управлению качеством продукции и стандартам от 26 дек. 1990 г. № 3294 : дата введ. 1992-01-01. – Москва : Изд-во стандартов, 2010. – 26 с.

2. Зенков, А. В. Численные методы : учеб. пособие / А. В. Зенков ; Урал. федер. ун-т им. первого Президента Российской Федерации Б. Н. Ельцина. – Екатеринбург : Изд-во Уральского ун-та, 2016. – 123 с. – ISBN 978-5-7996-1781-3.

3. Керниган, Б. Язык программирования С : пер. с англ. / Брайан Керниган, Деннис Ритчи. – 2-е изд., перераб. и доп. – Москва : Диалектика, 2020. – 288 с. – ISBN 978-5-907144-14-9.

4. Кнут, Э. Д. Искусство программирования. Т.1 Основные алгоритмы / Э. Д. Кнут. – Москва : Вильямс, 2020. – 720 с. – ISBN 978-5-907144-23-1.

5. Конова, Е. А. Алгоритмы и программы. Язык С++ : учеб. пособие. – 2-е изд., стер. / Е. А. Конова, Г. А. Поллак. – Москва : Лань, 2017. – 384 с. – ISBN 978 5 8114 2020 9.

6. Павловская, Т. А. С/С++. Программирование на языке высокого уровня : для магистров и бакалавров : учеб. / Т. А. Павловская. – Санкт-Петербург : Питер, 2021. – 460 с. – ISBN 978-5-4461-1350-7.

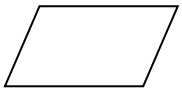
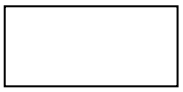
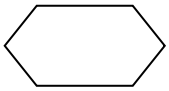
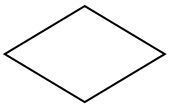
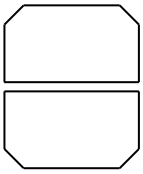
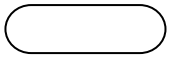
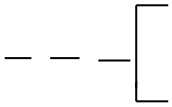
7. Программирование С/С++. Проектирование алгоритмов и программ : метод. указания / сост. О. П. Шафеева. – Омск : Изд-во ОмГТУ, 2017. – 32 с. – 1 CD-ROM (0,49 Мб). – Загл. с этикетки диска.

8. Страуструп, Б. Язык программирования С++. Краткий курс. – 2-е изд. / Бьярне Страуструп. – Москва : Вильямс, 2019. – 320 с. – ISBN 978-5-907144-12-5.

9. Шафеева, О. П. Технологии программирования С++ : учеб. пособие / О. П. Шафеева ; Ом. гос. техн. ун-т. – Омск : Изд-во ОмГТУ, 2017. – Загл. с этикетки диска. – ISBN 978-5-8149-2403-2.

ПРИЛОЖЕНИЕ

Обозначения графические в схемах алгоритмов (по ГОСТ 19.701–90)

Символ	Наименование	Назначение
	Данные	Определяет ввод или вывод на внешнее устройство или любой носитель данных
	Процесс	Отражает обработку данных: выполнение отдельной операции или группы операций
	Предопределенный процесс	Отображает предопределённый процесс, состоящий из одной или нескольких операций программы, которые определены в другом месте (подпрограмме, модуле)
	Подготовка	Отражает инициализацию и модификацию параметра для управления циклом со счётчиком
	Решение	Описывает проверку условия и выполняет переключение по одному из условий. Имеет один вход и два (или более) альтернативных выхода, один из которых активизируется после вычисления условия внутри символа
	Границы цикла	Состоит из двух частей: начала и конца цикла. Обе части имеют один и тот же идентификатор. Изменение значения идентификатора, условия для выполнения или завершения помещаются внутри символов в начале или в конце цикла
	Соединитель	Используется для обрыва линии и продолжения её в другом месте. Должен содержать уникальное числовое обозначение
	Терминатор	Определяет начало и конец схемы алгоритма, программы или подпрограммы
	Основная линия	Отображает последовательность выполнения действий в алгоритме
	Комментарий	Используется для добавления пояснительных записей. Связывается с символом или группой символов, обведённых пунктиром