ФЕДЕРАЛЬНОЕ АГЕНСТВО ПО ОБРАЗОВАНИЮ
Государственное образовательное учреждение
высшего профессионального образования
УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет математики и информационных технологий

В.В. Угаров

Технология программирования
часть 1

Учебно-методическое пособие

Ульяновск – 2010

УДК 681.142.2

Печатается по решению Ученого Совета
факультета математики и информационных технологий
Ульяновского государственного университета

Рецензенты:
доктор технических наук, профессор И.В. Семушин;
кандидат педагогических наук Г.А. Жаркова

Угаров В.В.
Технология программирования. Часть 1: учебно-методическое пособие / В.В.Угаров.–Ульяновск: УлГУ, 2010.–83 с.

Данное учебное пособие представляет собой курс лекционного материала по дисциплине «Технология программирования» с некоторыми сокращениями, в который добавлено значительное количество примеров, заданий для самостоятельного выполнения и приложений.

Пособие состоит из введения, 4 глав, списка литературы и двух приложений.

В первых главах пособия рассматриваются некоторые вопросы информационных технологий, затем краткое изложение основ языка программирования C++. В основном же пособие посвящено вопросам формирования структур данных и алгоритмов их обработки.

Отличие данного пособия от многих аналогичных состоит в том, что в нем более глубоко рассматриваются структуры данных и методы их обработки, а не собственно язык программирования, который служит в основном для реализации алгоритма. Тем не менее, средства языка C++ в значительной мере влияют на способы реализации тех или иных алгоритмов обработки. Естественно, в некоторых случаях такие моменты в данном пособии оговариваются.

В пособии рассматриваются свойства, как простых типов данных, так и более сложных структур данных: последовательностей, одномерных и двухмерных массивов, структур, файлов. Параллельно с рассмотрением структур данных подробно анализируются алгоритмы их обработки и связанные с этим средства языка программирования: механизм функций, модулей, библиотек. Представлен широкий спектр алгоритмов обработки последовательностей, массивов и файлов

Примеры программ по текущим темам могут быть использованы для изучения раздела темы студентами соответствующих курсов, а задания - для получения навыков в практике составления программ.

СОДЕРЖАНИЕ

Введение

Учебная дисциплина "Технология программирования" читается на 1 курсе университета для информационных специальностей.

Назначение дисциплины: изучение основных (базовых) структур данных, грамотное программирование задач объемом до нескольких (2-5) тысяч строк, освоение и изучение базовых алгоритмов, лежащих в основе большинства программных продуктов, освоение методики создания программ прикладного назначения.

Этот курс посвящен программирования как таковому.

Не рассматриваются вопросы вычислительных методов приближенных вычислений, операционных систем, методы построения компиляторов, и т.д. Эти темы могут слегка затрагиваться при изучении основных разделов.

В этой дисциплине предполагается изучение текстов готовых программ и создание новых, учебных программ объемом несколько тысяч строк.

Дисциплина "Технология программирования" читается в 1 семестре учебного года, а во втором семестре читается тесно связанная с ней "Технология программирования-2".

Во втором семестре рассматриваются более сложные вопросы построения прикладных программ. В частности, происходит знакомство с элементами объектно-ориентированного программирования, освоение работы с динамическими структурами данных, рассмотрение разнообразных алгоритмов на графах, знакомство с основами программирования в среде Windows на основе инструментального пакета Visual C++, изучение методов индустриального производства программных продуктов.

Наряду с лекционным курсом параллельно проводятся семинарские и практические занятия, в том числе в компьютерных классах. На них студенты выполняют работы по созданию учебных программ, осваивают среду программирования, получают навыки грамотного программирования.

В конце каждого семестра студент сдает экзамен. К экзамену допускаются только студенты, выполнившие необходимый объем работ как на семинарах (контрольные работы), так и на практических занятиях (лабораторные проекты).

Основные требования к программным проектам, которые создаются в ходе учебного процесса: во-первых, это работоспособность программ, т.е. программы должны выполнять то, что задано, без каких либо исключений, во-вторых, проект должен быть выполнен полностью самостоятельно.

Количество защищенных работ влияет на оценку экзамена.

Для работы в дисплейном классе студентам необходимо будет приобрести средства для хранения своих учебных программ либо пару дискет, либо флэш-память, если в компьютерном классе есть вход USB.

Поскольку время работы в компьютерном классе ограничено учебным планом, то для выполнения домашних заданий и доработки проектов, студен-

там необходимо получить доступ к компьютеру вне университета. Это может быть личный домашний компьютер, компьютер на работе у родителей, компьютер у друзей и т.д.

1. Элементы информационных технологий

1.1. Системы счисления.

1.1.1. Исторический аспект.

Под системой счисления понимается способ записи чисел с помощью цифр и символов (букв).

Исторически человечество наряду с созданием письменности разрабатывало также и способы записи чисел, требуемых для хранения данных о количестве предметов. В этом случае появляется возможность сравнивать количества различных множеств объектов.

Самой древней, пожалуй, системой счисления, является так называемая унарная система (или счетно-импульсная). Унарная система является однородной и непозиционной. Число в этой системе строится прибавлением единственного знака к предыдущему такому же числу:

I II III IIIIIIIII IIIIII IIIIIII

Система позволяет представить любое, сколь угодно большое число. Однако представление чисел крайне неэффективно с точки зрения компактности.

Следующим этапом в развитии систем счисления является римская система. Она также относится к непозиционным системам. В римской системе сохраняются некоторые элементы унарной системы и в то же время предпринимаются меры к более компактному представлению числа путем замены совокупности элементарных цифр на определенные символы (иероглифы).

I II III IV V VI VII VIIIIX X XI XII XIII XIV XV XVI XVII

Таким образом, в этой системе латинские буквы соответствуют следующим количествам:

L – 50, C – 100, D – 500, M – 1000

Например:

2007 – MMVII, 444 – CDXLIV.

В этой системе очень трудно выполнять арифметические операции.

Дальнейшее развитие представления чисел привело к распространению позиционных систем счисления, в которых значение ("вес") цифры зависит от ее положения в числе.

Первой известной позиционной системой счисления была шестидесятеричная система у древних вавилонян, использующая клинопись. Так, число 32 они записывали: **<<<VV,** а число 92: **V<<<VV.**

Таким образом, основанием системой счисления называется количество цифр (назовем его **q**). Величина **q** показывает, во сколько раз возрастает вес

цифры, если ее переместить в следующий разряд числа. Отметим, что данная система требует наличия цифры ноль.

Позиционная система является однородной в смысле применения правил построения числа независимо от места расположения текущей цифры.

Именно однородность позволяет конечным числом знаков (в десятичной системе – 10) выразить бесконечное количество чисел.

Рассмотрим более подробно десятичную систему счислению.

Она имеет 10 цифр: **0,1,2,3,4,5,6,7,8,9.**

Например, число **111.**

Вес цифры изменяется в 10 раз: **100+10+1 = 111.**

Поэтому в позиционной системе счисления любое число можно представить в виде следующего выражения:

$$R_i = a_{k-1} \cdot q^{k-1} + a_{k-2} \cdot q^{k-2} + \ldots + a_1 \cdot q^1 + a_0 \cdot q^0$$

Таким же образом можно представить и дробное число:

$$R_i = a_{k-1} \cdot q^{k-1} + \ldots + a_0 \cdot q^0 + a_{-1} \cdot q^{-1} + a_{-2} \cdot q^{-2} + \ldots$$

Например, число 16,37 можно представить в виде:

$$16{,}37 = 1 \cdot 10^{+1} + 6 \cdot 10^0 + 3 \cdot 10^{-1} + 7 \cdot 10^{-2}$$

1.1.2. Двоичная система счисления.

В компьютерной технике повсеместно используется двоичная, восьмеричная и шестнадцатеричная системы счисления. Это связано с несколькими причинами.

Математически доказано, что самая экономная система счисления с точки зрения количества знаков имеет основание q = 2,711. ЭВМ, построенная на основе такой системы, теоретически будет иметь наименьшее количество оборудования[1]. Ближайшей к ней системой будет двоичная с q = 2.

Компьютер представляет собой прибор, функционирующий на основе тех или иных физических эффектов[2]. Чтобы компьютер мог выполнять вычислительные операции, он должен хранить у себя в памяти данные. Хранение данных основано на устойчивых состояниях определенных физических эффектов. Например, вода может находиться в трех агрегатных устойчивых состояниях: твердом, жидком и газообразном. Но в природе нет физических эффектов, имеющих много устойчивых состояний, например 8 или 10. Зато имеется множество электрических, магнитных, оптических и т.п. эффектов, имеющих два устойчивых состояния, например: направление намагниченности металла, заряд конденсатора, замкнутое или разомкнутое состояние контактной группы и т.д.

[1] Кстати, в 60-е годы в МГУ была построена ЭВМ "Сетунь", работающая в троичной системе счисления. Один экземпляр такой машины купили американцы для исследования.

[2] Существуют механические, пневматические, гидравлические, электрические вычислительные и управляющие устройства.

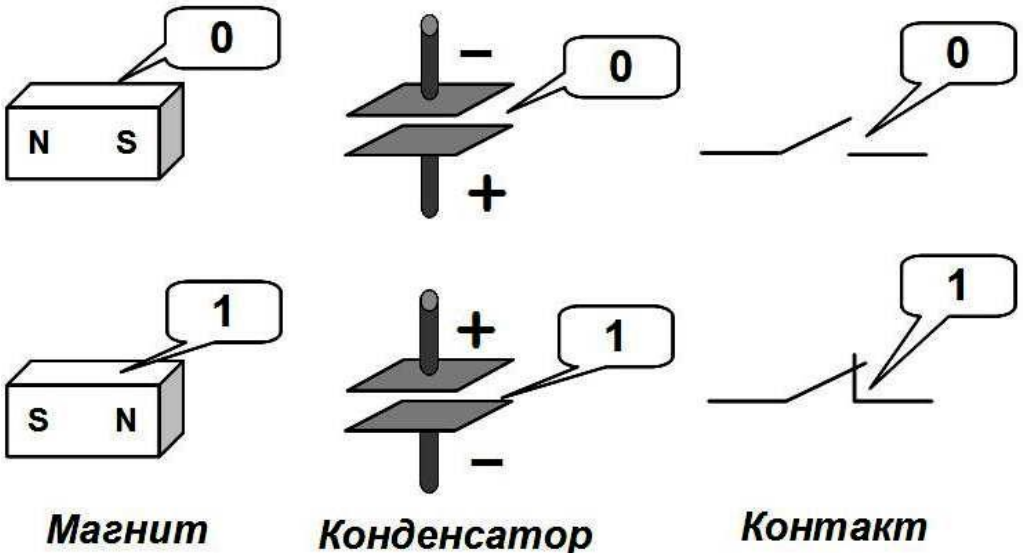


Рис.1.1.2.

Если обозначить качественные состояния физических явлений двоичными цифрами 0 и 1, то с их помощью можно хранить некоторые численные значения данных в физическом устройстве.

В двоичной системе используются две цифры: 0 и 1. Основание системы: q = 2. Тогда некоторое число в двоичной системе может выглядеть следующим образом:

$$1101_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 13_{10}$$

1.1.3. Арифметические операции в двоичной системе счисления.

Рассмотрим теперь арифметические действия в двоичной системе счисления. Для их выполнения приведем таблицы сложения и умножения:

Таблица сложения				
1 операнд				
2 операнд				
Результат				0

Таблица умножения				
1 операнд				
2 операнд				
Результат				

Хотя в двоичной системе можно выразить любые, сколь угодно большие числа, в компьютере количество разрядов, необходимых для представления чисел, ограничено. Поэтому при выполнении арифметических операций необходимо следить за величиной чисел, участвующих в вычислениях, не допуская переполнения разрядной сетки.

0	0	0	0	1	1	0	1	0	1	1	1	0	0	1	1	1	1	0	1	0	1	0	0	0	1	1	0	1	0	0	1

Большинство арифметических операций выполняются в двоичной системе счисления так же, как в 10-тичной.

Примеры арифметических действий.

Сложение:

```
    1 0 0 1 0 0 1 1    -   147
+
    0 1 0 1 1 1 1 0    -    94
  -------------------      ----
    1 1 1 1 0 0 0 1    -   241
```

Вычитание

```
    1 0 0 1 0 0 1 1    -   147
-
    0 1 0 1 1 1 1 0    -    94
  -------------------      ----
    0 0 1 1 0 1 0 1    -    53
```

Умножение

```
     0 0 0 1 1 0 1     -    13
*
     0 0 0 0 1 0 1     -     5
  -------------------      ----
     1 0 0 0 0 0 1     -    65
```

1.1.4. Восьмеричная и 16-ричная система счисления.

Большая длина двоичных чисел неудобна для их чтения и названия. Поэтому для того, чтобы удобнее было оперировать с двоичными кодами, используют восьмеричную и шестнадцатеричную системы счисления. Особенность их состоит в том, что основания этих систем кратно двум:

$2^3 = 8;$ (1.4.a)

$2^4 = 16;$ (1.4.b)

Восьмеричная система с основанием **q = 8** имеет цифры:

0, 1, 2, 3, 4, 5, 6, 7.

16-ричная система с основанием **q = 16** имеет цифры:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

1.1.5. Перевод чисел из одной системы счисления в другую.

Как известно, в реальной практике все числовые данные представлены в десятичной системе счисления. Однако компьютерные системы работают в двоичной системе счисления. Поэтому возникает необходимость перевода чисел. Значения чисел при этом не меняются, изменяется только форма представления.

Входные данные из входного потока поступают в десятичном виде, затем переводятся в двоичный вид, в котором и выполняются все вычисления и преобразования данных, результаты затем переводятся в десятичный вид для вывода на печать или монитор.

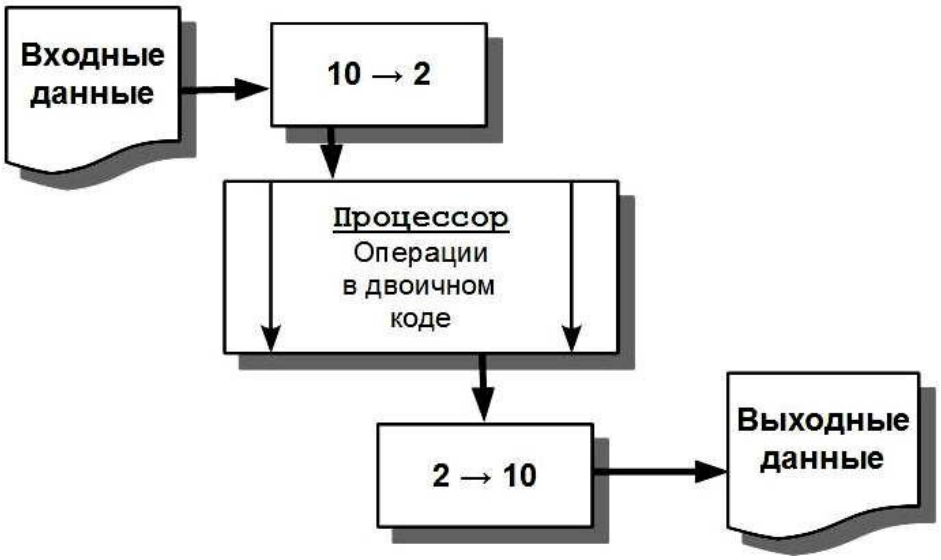


Рис. 1.1.5.

Восьмеричная и 16-тиричная системы счисления используются для индикации, документирования, программирования и отладки программ, для представления в удобном виде длинных двоичных наборов данных.

Определим правила переводы чисел из одной системы в другую. Для выполнения перевода нам будет нужна следующая таблица:

Таблица 1.1.5.

10-тичное число	2-ичное число	8-ричное число	16-ричное число	2^x
0	0000	0	0	1
1	0001	1	1	2
2	0010	2	2	4
3	0011	3	3	8
4	0100	4	4	16
5	0101	5	5	32
6	0110	6	6	64
7	0111	7	7	128
8	1000	10	8	256
9	1001	11	9	512
10	1010	12	A	1024
11	1011	13	B	2048

12	1100	14	C	4096
13	1101	15	D	8192
14	1110	16	E	16384
15	1111	17	F	32768
16	10000	20	10	65536

1) Перевод 2 → 10.

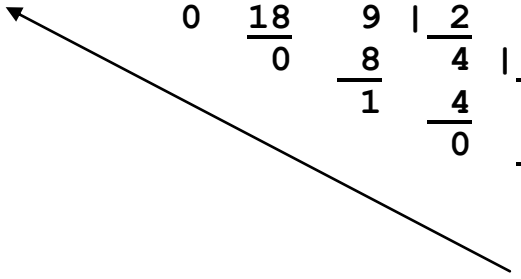
Для перевода необходимо двоичное число представить в виде выражения по степеням основания и вычислить это выражение в десятичной системе счисления. Например:

$$10010011_2 = 1\cdot 2^7 + 0\cdot 2^6 + 0\cdot 2^5 + 1\cdot 2^4 + 0\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 = 147_{10}$$

2) Перевод 10 → 2.

Для перевода необходимо десятичное число циклически делить на основание новой системы (**q=2**) до тех пор, пока частное не станет равным нулю. Остатки, полученные в операциях деления, начиная с последнего, представляют собой изображение двоичного числа. Например:

```
147 | 2
146  73 | 2
  1  72  36 | 2
     1  36  18 | 2
        0  18   9 | 2
           0   8   4 | 2
               1   4   2 | 2
                   0   2   1 | 2
                       0   0   0
                           1
```



= 10010011_2

3) Перевод 8 → 10.

Для перевода необходимо восьмеричное число представить в виде выражения по степеням основания **q=8** и вычислить это выражение в десятичной системе счисления. Например:

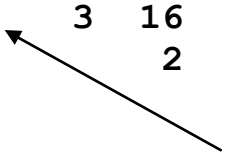
$$235_8 = 2\cdot 8^2 + 3\cdot 8^1 + 5\cdot 8^0 = 157_{10}$$

4) Перевод 10 → 8.

Для перевода необходимо десятичное число циклически делить на основание новой системы **(q=8)** до тех пор, пока частное не станет равным нулю.

Остатки, полученные в операциях деления, начиная с последнего, представляют собой изображение двоичного числа. Например:

```
 147 | 8
 144  18 | 8
   3  16   2 | 8
        2   0   0
            2
```



$= 223_8$

5) Перевод 2 → 8.

Пользуясь свойством (1.4.a), для перевода необходимо двоичное число разделить условно на триады (по три разряда), начиная с младшего разряда, а затем каждую триаду представить в виде восьмеричной цифры из таблицы 1.5.1. Например:

$10.011.101.110.0012 = 23561_8$

6) Перевод 8 → 2.

Пользуясь свойством (1.4.a), для перевода необходимо каждую цифру восьмеричного числа представить в виде соответствующей триады двоичного числа из табл. 1.5.1.

Например:

$27453_8 = 10.111.100.101.011$

7) Перевод 16 → 2 и 2 → 16.

Пользуясь свойством (1.4.b), перевод выполняется аналогично 5) и 6), но 16-тиричное число в этом случае должно быть представлено в виде соответствующей тетрады двоичного числа, взятой из табл. 1.1.5. Например:

$2FC8E_{16} = 10.1111.1100.1000.11102$

$1.1001.1101.1111.1010.0010 = 19DFA2$

8) Перевод дробного числа.

При переводе десятичной дроби дробную часть необходимо циклически умножать на основание новой системы. Получающиеся при этом целые части, начиная с первой, являются изображением двоичного дробного числа. Пример: перевод числа 0,34:

```
0 | 34
0 | 68
1 | 36
0 | 72
1 | 44
0 | 88
1 | 76
1 | 52
1 | 04
```

```
0 | 08
0 | 16
0 | 32
```

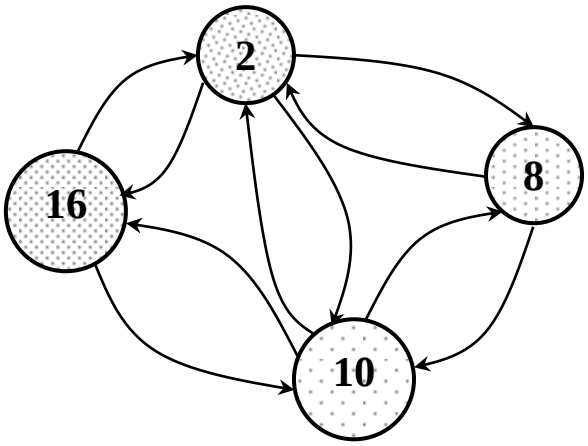
Выполним проверку полученного двоичного числа:

$$0{,}34 = 0{,}01010111000_2$$

$$0{,}34 \approx \overset{0}{0},\overset{-1}{0}\overset{-2}{1}\overset{-3}{0}\overset{-4}{1}\overset{-5}{0}\overset{-6}{1}\overset{-7}{1}\overset{-8}{1}\ldots \approx 1\cdot 2^{-2}+1\cdot 2^{-4}+1\cdot 2^{-6}+1\cdot 2^{-7}+1\cdot 2^{-8} = \frac{1}{4}+\frac{1}{16}+\frac{1}{64}+\frac{1}{128}+\frac{1}{256} =$$

$$= 0{,}25+0{,}0625+0{,}0156+0{,}00781+0{,}0039 \approx 0{,}33981$$

Итак, в результате мы получили набор правил перевода чисел в следующем схематичном виде:



1.2. Элементы информационной метрики.

1.2.1. Понятие информации.

"Информация" – (лат. разъясняю, излагаю) термин, широко применяемый при изучении различных современных дисциплин.

В широком смысле, информация – это отражение реального (материального, предметного) мира, выражаемого в виде сигналов и знаков.

Эти сигналы отражают те или иные характеристики различных явлений или процессов. Вся информация отображается в виде некоторых конечных совокупностей знаков (символов), из которых образуется бесконечное множество информационных объектов (текстов, изображений и т.д.).

Правила формирования из знаков информационных сообщений определяются соответствующим языком. Языки делятся на разговорные (естественные) и формальные. Последние применяются в математике и информационных науках.

Познавая окружающий мир, человек наделяет явления и объекты этого мира именами. Это приводит к тому, что в сознании людей объект замещается именем, от которого требуется, чтобы оно помогло опознать названный объект. Вот эта возможность разделения предмета и его имени является основой фор-

мализации, позволяет рассматривать язык или систему языков как универсальную моделирующую среду.

Для хранения и передачи информации необходимо использовать ту или иную систему кодирования. Кодирование – это процесс преобразования сигналов без изменения их содержания.

Поскольку сообщения состоят из конечных последовательностей знаков (символов), необходимо преобразование их в сигналы электрической, магнитной, оптической или другой природы. Поэтому процессы кодирования и декодирования являются важнейшими в передаче и хранении информации.

1.2.2. Методы измерения количества информации

Информация может быть представлена в различных формах. Существуют классификации по различным признакам, например, по тематическому: физическая, биологическая, химическая и т.д., по способу восприятия: зрительная, слуховая, тактильная, вкусовая и т.д.

Практический интерес представляет классификация форм информации по метрическим свойствам:

Параметрическая форма – наборы числовых оценок, полученных при исследовании, анализе, контроле, учете, измерениях и т.д.

Топологическая форма – получаемая при создании графических образов: карты, чертежи, фотографии, планы и т.д.

Абстрактная форма – математические соотношения, обобщенные образы, понятия, применяемые при исследованиях на высоком теоретическом уровне.

В основе методик измерения количества информации лежит понятие неопределенности события.

Если взять некоторый информационный объект и провести над ним опыт, то в результате опыта может произойти или не произойти заданное событие. Если результат опыта заранее известен, то в этом случае никакой информации мы не получаем.

Перед опытом мы не знаем, какое событие произойдет, то есть существует неопределенность относительно исхода опыта. Когда же опыт будет произведен, его исход становится известен и неопределенность снимается. Если в результате опыта может быть реализован один из нескольких исходов, то неопределенность возрастает, а, следовательно, мы получаем большее количество информации.

Например, при бросании монеты может быть два исхода: орел или решка. При бросании кубика количество исходов возрастает до шести. При чтении русской буквы количество исходов составляет уже 32.

Поэтому в качестве меры информации можно взять количество возможных исходов опыта. Назовем это количество мощностью информационного источника. Однако эта мера оказалась неудобной в практическом использовании,

и в 1928 году Р. Хартли[3] предложил в качестве меры информации использовать двоичный логарифм от мощности источника информации:

$$I = \log_2 M \qquad (1.2.1)$$

где M – мощность источника информации

В качестве примера возьмем опыт с двумя равновероятными исходами. В этом случае $I = \log_2 2 = 1(bit)$. Р. Хартли назвал единицу информации "бит", от английского выражения "binary unit".

Если взять опыт с шестью исходами (игральный кубик), то количество информации в результате опыта:

$$I = \log_2 M = \log_2 6 = 2{,}58\,(bit)$$

Мера Хартли может быть применена только в случае, когда исходы опыта равновероятны. В этом случае это удобная в практическом плане единица измерения.

1.2.3. Структурные меры информации

Структурная мера информации определяет максимально возможное (потенциальное) количество информации в заданных структурных объемах, называемое информационной емкостью.

В качестве примера рассмотрим трехмерный информационный комплекс, в качестве которого возьмем помещение библиотеки, в котором размещены книги.

Пусть размеры помещения: X_q, Y_q, Z_q метров, а размеры одной книги $\Delta X, \Delta Y, \Delta Z$. Тогда количество книг, размещенных по осям X, Y, Z:

$$m_x = \frac{X_q}{\Delta X}; \quad m_y = \frac{Y_q}{\Delta Y}; \quad m_z = \frac{Z_q}{\Delta Z};$$

Здесь $m_x; m_y; m_z;$ - мощности по осям X, Y, Z.

Тогда общая мощность информационного источника равна:

$$M = m_x \cdot m_y \cdot m_z$$

Чтобы определить количество информации, необходимой для указания на определенную книгу в этом помещении, используем меру Хартли:

$$I = \log_2 M = \log_2 (m_x \cdot m_y \cdot m_z) = \log_2 (m_x) + \log_2 (m_y) + \log_2 (m_z) = I_x + I_y + I_z$$

Таким образом, мера Хартли обладает свойством аддитивности, т.е. мощности от независимых источников перемножаются, а количества информации от независимых источников складываются.

1.2.4. Комбинаторные меры информации

Комбинаторная мера основана на комбинировании объектов. Образование комбинаций есть одна из форм кодирования информации, и количество ее определяется как количество комбинаций элементов.

[3] Теория информации и ее приложения (Сборник переводов). Под редакцией А.А.Харкевича. Государственное издательство физико-математической литературы. Москва. 1959 г.

Комбинирование возможно группированием неодинаковых элементов, изменением позиции элемента в данной комбинации, изменением количества элементов и т.д. В комбинаторике рассматриваются различные виды соединений элементов. Рассмотрим их подробнее.

1. Сочетания из "n" элементов по "m",
различающиеся только составом элементов. Количество комбинаций:

$$C_n^m = \frac{n!}{m!(n-m)!} = \frac{n\,(n-1)\,(n-2)\,(n-3)\ldots(n-m+1)}{1\cdot 2\cdot 3\cdot \ldots \cdot m};$$

Пример. Составить сочетания:
а) из 3-х элементов {A,B,C} по 2. Имеем: AB, BC, AC. Всего 3 комбинации.
б) из 4-х элементов {A,B,C,D} по 2. Имеем: AB, AC, AD, BC, BD, CD. Всего 6 комбинаций.

2. Перестановки из "N" элементов, различаются только их порядком в комбинации. $P = 1\cdot 2\cdot 3\cdot \ldots \cdot N = N!;$
Пример. Составить перестановки из 3-х элементов. Всего 3!=1*2*3=6.
Имеем следующие комбинации: ABC, ACB, BAC, BCA, CAB, CBA.

3. Размещения из "n" элементов по "m", различаются и составом элементов и их порядком. $A_n^m = \frac{n!}{(n-m)!};$

Пример. Составить размещения из 4-х элементов по 2. Всего можно составить 12 комбинаций.
Имеем следующие комбинации:
AB, AC, AD, BC, BD, CD,
BA, CA, DA, CB, DB, DC.

4. Размещения с повторениями отличаются тем, что элементы в комбинации могут повторяться. $Q_n^m = n^m;$

Количество информации при комбинировании элементов многократно возрастает. Так, в случае сочетаний из 10 элементов по 0,1,2,3,…9 элементов имеем общее число комбинаций:

$$Q = \frac{10!}{0!(10-0)!} + \frac{10!}{1!(10-1)!} + \ldots + \frac{10!}{9!(10-9)!} + \frac{10!}{10!(10-10)!} =$$

= 1+10+45+120+210+120+45+10+1=1024;
Перестановки тех же 10 элементов дают:

$$Q = n! = 10! = 1\cdot 2\cdot 3\cdot \ldots \cdot 10 = 3628800;$$

Размещения с повторениями из 10 элементов по 10 приводят к еще большему числу комбинаций:

$$Q_{10}^{10} = 10^{10} = 10000000000;$$

1.2.5. Статистические меры информации

Недостатком структурного метода измерения информации является то, что в нем не учитывается вероятность наступления того или иного исхода. Для определения количества информации в случае, если исходы опыта имеют разную вероятность, используется статистическая мера К.Шеннона, предложенная им в 1948 году.

В основе статистического метода определения информации лежит положение о том, что получение информации снимает часть некоторой (априорной, до опытной) неопределенности.

Большинство источников информации характеризуется неопределенностью, связанной с неодинаковой вероятностью происходящих событий. Естественно, что с ростом числа возможных исходов неопределенность должна возрастать. Меру степени неопределенности называют энтропией.

Пусть мы имеем опыт, имеющий "N" равновероятных исходов.

Такую неопределенность называют равной:

$H = \log_2 N$;

Это выражение можно записать в виде:

$$H = \log_2 N = N\frac{1}{N}\log_2 N = N\left(\frac{1}{N}\log_2 N\right) = N\left(-\frac{1}{N}(\log 1 - \log N)\right) = N\left(-\frac{1}{N}\log\frac{1}{N}\right);$$

Из теории вероятностей известно, что $\frac{1}{N} = p$ - вероятность любого из N возможных исходов опыта, поэтому выражение (1) переписываем в виде:

$H = N(-p\log p)$;

При N=2 имеем:

$H_1 = \log_2 2 = 1$; (бит))

Бит - это единица для измерения степени неопределенности опыта.

А как же измерить неопределенность в случае разновероятных исходов?

Пусть некоторый опыт характеризуется следующей таблицей вероятности:

Исходы опыта:	A_1	A_2	A_3	. . .	A_i	. . .	A_N
Вероятность:	p_1	p_2	p_3	. . .	p_i	. . .	p_N

Естественно, что $p_1 + p_2 + p_3 + \ldots + p_i + p_N = 1$.

Тогда в соответствии с формулой (2) меру неопределенности этого опыта запишем в виде:

$H_\alpha = -p_1\log_2 p_1 - p_2\log_2 p_2 - p_3\log_3 p_3 - \ldots - p_i\log_2 p_i - \ldots - p_N\log_2 p_N$;

или

$$H_\alpha = -\sum_{i=1}^{N} p_i \log_2 p_i; \qquad (1.2.5)$$

Полученное выражение имеет вид, совпадающий с видом выражения для энтропии в статистической физике, причем это несет не только формальный, но и содержательный характер.

Поэтому величину H_α называют энтропией опыта α.

Свойства выражения (1.2.5): Любое слагаемое всегда положительно, т.к. $0 \le p_i \le 1$, а следовательно $\log p_i$ всегда отрицателен. При $p_i \to 1$ выражение $p_i \log p_i$ убывает и стремиться к 0, т.к. $\log 1 = 0$.

Пример. Пусть мы имеем следующий опыт: к нам пришло следующее сообщение: $A_1\ A_3\ A_1\ A_3\ A_3\ A_2\ A_3 A_4$

Требуется определить количество информации в данном сообщении.

Алфавит этого сообщения состоит из 4 букв: A_1 , A_2 , A_3 , A_4 .

Следовательно, для кодирования этих букв достаточно будет двух двоичных разрядов: $A_1 - 00$, $A_2 - 01$, $A_3 - 10$, $A_4 - 11$.

Если применить меру Хартли, то для передачи данного сообщения при применении равномерного кода необходимо будет 16 двоичных разрядов, т.е. 16 бит. Причем на одну букву приходится 2 бита: $I = \log_2 4 = 2\ (bit)$

Однако такой подход не учитывает неравной вероятности появления букв в сообщении и поэтому не может считаться правильным.

Определим вероятности появления букв в сообщении:

$P_1 = 0{,}25$; $P_2 = 0{,}125$; $P_3 = 0{,}5$; $P_4 = 0{,}125$;

В этом случае количество информации, приходящейся на одну букву в этом сообщении, равно:

$$I = -p_1 \log_2 p_1 - p_2 \log_2 p_2 - p_3 \log_3 p_3 - p_4 \log_2 p_4 =$$

$$= -0{,}25 \cdot \log_2 0{,}25 - 0{,}125 \cdot \log_2 0{,}125 - 0{,}5 \cdot \log_2 0{,}5 - 0{,}125 \cdot \log_2 01{,}25 =$$

$$= -\frac{1}{4}\log_2 \frac{1}{4} - \frac{1}{8}\log_2 \frac{1}{8} - \frac{1}{2}\log_2 \frac{1}{2} - \frac{1}{8}\log_2 \frac{1}{8} =$$

$$= 0{,}25 \cdot 2 + 0{,}125 \cdot 3 + 0{,}5 \cdot 1 + 0{,}125 \cdot 3 =$$

$$= 0{,}5 + 0{,}5 + 0{,}75 = 1{,}75\ (bit)$$

И таким образом, общие количество информации в этом сообщении составляет 8*1,75 = 14 (bit), что меньше, чем при равномерном коде.

Отсюда следует, что неравная вероятность появления букв в сообщении приводит к уменьшению избыточности количества информации.

В том же 1948 году К.Шеннон (на фото) доказал теорему о том, что возможен такой способ кодирования, который приводит к уменьшению длины двоичного кода сообщения, в котором наблюдается неравная вероятность появления букв. Тогда же он совместно с Фано предложил алгоритм оптимального кодирования, позволяющий уменьшать длину сообщения. Этот алгоритм широко применяется в программах архивирования данных.

Мера Шеннона и алгоритмы, разработанные им для кодирования информации, широко применяются в практике программирования, в частности, при разработке алгоритмов архивации файлов, например таких, как pkzip, arj, zip, 7zip, rar и ряда других, а также в системах обнаружения и исправления ошибок при передаче данных.

1.3. Основы вычислительной техники.

1.3.1. Кодирование числовой и символьной информации.

Компьютеры могут хранить и обрабатывать только двоичные коды. Числовая информация переводится в двоичный вид и в виде двоичных кодов обрабатывается. Но чтобы на этой технической базе имелась возможность обработки и текстовой информации, для этого необходимо каждый символ текста закодировать некоторым двоичным набором.

Для этого используют соглашения, оформленные в виде кодировочных таблиц. Для практических целей достаточно включить в набор символов большие и маленькие латинские буквы (26+26 шт.), русские буквы (33+33), цифры (10), знаки препинания и математические операции (около 40 шт.), набор служебных символов (несколько десятков), символы псевдографики и другие. Т.е. набирается около 250 символов. Поэтому для кодирования данного набора был выбран 8-битный двоичный набор. Такой набор носит название байт. Это более крупная единица измерения информации, чем бит, широко используется в современных информационных технологиях.

Применяются также и более крупные производные единицы:

1 байт – 8 бит.

1 Кбайт – 1024 байт.

1 Мбайт – 1024 Кбайт или 1 048 576 байт.

1 Гбайт – 1024 Мбайт или 1 073 741 824 байт.

1 Тбайт – 1024 Гбайт или 1 099 511 627 776 байт.

На основе 8 бит можно сгенерировать 256 различных двоичных наборов. Каждому двоичному набору сопоставляется определенный символ. Наиболее распространенной таблицей соответствия является таблица кодов ASCII (American Standard Code for Information Interchange) – стандартный код для информационного обмена. В России используются кодировочные таблицы Windows-1251 и КОИ-8.

Например:

Цифры:

"0" – 48 – 00110000

"1" – 49 – 00110001

....-....-.........

"9" – 57 – 00111001

Буквы:

"A" – 65 – 01000001

"B" – 66 – 01000010

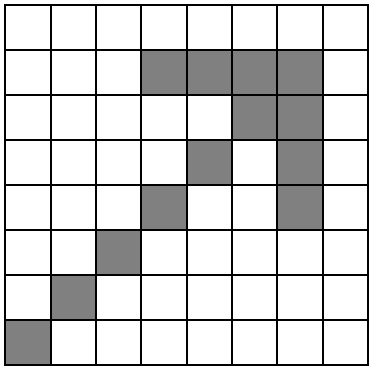
"C" – 67 – 01000011

....-....-.........

"Z" – 90 – 01011010

Кодирование изображений также выполняется в виде двоичных наборов. Поскольку любое изображение состоит из отдельных элементов – пикселов, достаточно описать цвет каждого пиксела в виде двоичного кода и мы получим последовательность двоичных наборов, хранящих информацию об изображе-

нии. Пример: закодировать двухцветное изображение, представленное на рисунке 1.3.1.



16		*10*
00	-	0
1E	-	30
06	-	6
0A	-	10
12	-	18
20	-	32
40	-	64
80	-	128

Рис. 1.3.1.

В результате получаем следующую последовательность десятичных чисел: (0, 30, 6, 10, 18, 32, 64, 128). Затем последовательности упаковываются архиваторами и в таком виде хранятся. Существуют различные виды архиваторов изображений, с полным восстановлением – GIF, PCX, BMP, и с восстановлением с потерями – JPG, DVI и множество других.

На этом или подобном принципе происходит кодирование звуковой и видео информации.

1.3.2. Архитектура ЭВМ.

Компьютер – это устройство для исполнения алгоритмов, реализованных на том или ином языке программирования. Как всякий исполнитель, компьютер обладает определенным набором команд, которые выполняются под управлением программы (Software). Аппаратура компьютера (Hardware) построена на основе микроэлектронных элементов. Для выполнения программ в составе компьютера должны быть следующие, связанные между собой устройства:

1. Процессор (CPU или АЛУ) – выполняет арифметические и логические операции над двоичными наборами.
2. Оперативная память (Memory или ОЗУ) – устройства для хранения данных в виде двоичных наборов.
3. Внешняя память (Input-Output или УВВ) – устройства для долговременного хранения данных.
4. Внешние устройства для связи компьютера с внешним миром.

В основе архитектуры современных компьютеров лежат принципы, сформулированные в 1945 году американским ученым Джоном Фон Нейманом:

1. Принцип двоичности - Для представления данных и команд используется двоичная система счисления.
2. Принцип программного управления – последовательное выполнение команд в автоматическом режиме.
3. Принцип однородности памяти – программы и данные хранятся в единой памяти в виде двоичных наборов и могут быть модифицированы.

4. Принцип адресуемости памяти – память состоит из ячеек, каждая ячейка имеет свой адрес и процессор может произвольно их выбирать для операций записи или чтения (Произвольный доступ).

В современных ЭВМ и персональных компьютерах в том числе, за основу построения принята шинная структура компьютера. Шина (Bus) – в переводе означает магистраль (сравните: автобус). Эта структура основана на адресном принципе доступа к элементам памяти и внешним устройствам и позволяет гибко менять конфигурацию компьютера. Каждое его устройство имеет один или несколько адресов в адресном пространстве.

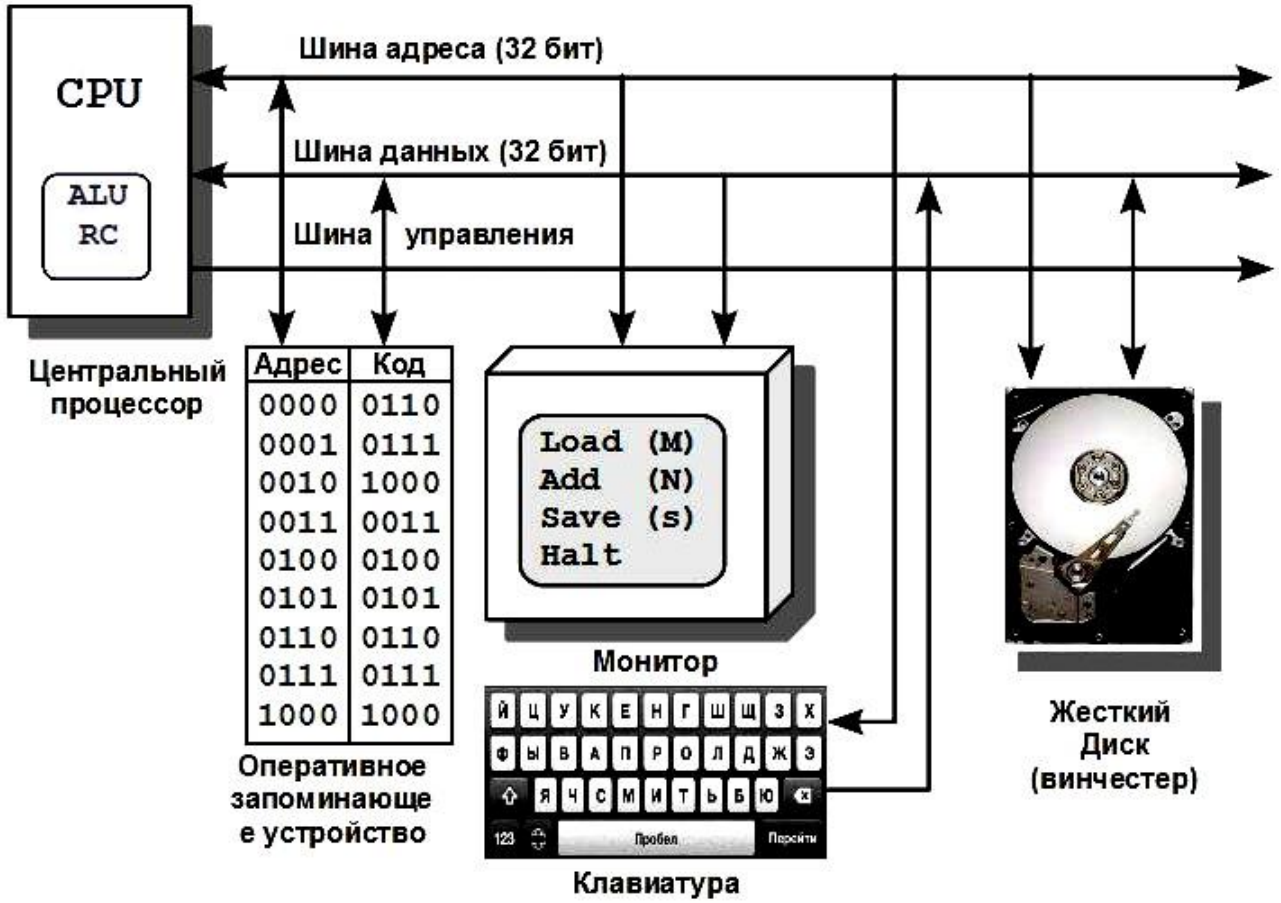


Рис. 1.2.3.

Очень коротко приведем основные параметры процессоров для персональных компьютеров:

1. Внутренняя структура процессора: CISC, RISC.
2. Фирма Intel: 8080, 80286, 80386, 80486, Pentium I, II, III, IV, Celeron, Xeon, Core 2 Duo, Intel Core 2 Quad, Intel Core i7 и т.д.
3. Фирма AMD: Athlon, Duron, Opteron, Sempron, AMD Phenom II X6 и т.д.
4. Частота процессора: 100, 166, 566, 1000, 1400, 2400, 3400 и т.д.
5. Объем оперативной памяти: 64 Кбайт, 640 Кбайт, 1 Мбайт, 32 Мбайт, 256 Мбайт, 512 Мбайт, 2 Гбайт, 4 Гбайт и т.д.
6. Емкость жесткого диска: 100 Мбайт, 540 Мбайт, 2 Гбайт, 16 Гбайт, 80 Гбайт, 320 Гбайт, 500 Гбайт, 1 Тбайт и т.д.

1.3.3. Исполнение программы на компьютере

В качестве примера рассмотрим исполнение простейшей программы на некоторой учебной ЭВМ.

Учебная ЭВМ имеет следующую систему команд, основанную на использовании аккумулятора A:

0001	+	ADD	сложить (A) и (N) →A
0000	Stop	HALT	Останов
0010	→A	LOAD	Загрузка в аккумулятор
0011	←A	SAVE	Запись в ОЗУ

Пусть требуется сложить два числа: S = M + N.

На основе системы команд напишем программу сложения двух чисел:

- LOAD (M) // Загружаем число из ячейки M в аккумулятор
- ADD (N) // Складываем аккумулятор с числом из ячейки N
- SAVE (S) // Записываем результат из аккумулятора в ячейку S
- HALT // Останов исполнения программы

Затем распределяем память компьютера следующим образом:

	Адрес ОЗУ	**Символ**	**Значение ячейки ОЗУ**	**Примечание**
0	**0000**		**0010 0110**	**LOAD (M)**
1	**0001**		**0001 0111**	**ADD (N)**
2	**0010**		**0011 1000**	**SAVE (S)**
3	**0011**		**0000 0000**	**HALT**
4	**0100**		**0000 0000**	
5	**0101**		**0000 0000**	
6	**0110**	**M**	**0000 0011**	**число 3**
7	**0111**	**N**	**0000 0101**	**число 5**
8	**1000**	**S**	**0000 0000**	**0000 1000**

Табл. 1.3.3.

Программа запускается с адреса 0000. Компьютер последовательно выбирает команды из памяти, загружает их в процессор, в котором они исполняются, при этом выбираются операнды из памяти, вычисляется сумма чисел и результат записывается в ячейку памяти.

1.3.4. Файловые системы

При обработке данных на компьютере необходимым компонентом является система хранения информации, позволяющая с малыми затратами добавлять данные в места их хранения и оперативно извлекать их для обработки. Для этих целей разработаны так называемые файловые системы.

Понятие файла является фундаментальным в информатике. Файл определяется как конечная последовательность двоичных наборов, хранящихся на некотором физическом носителе, и имеющая уникальное имя. Длина последова-

тельности может быть равна нулю. Сверху она ограничена только техническими параметрами устройства хранения.

Файлы хранятся на магнитных лентах, на магнитных дисках, флэш-памяти и других устройствах внешней памяти.

На магнитной ленте файл имеет следующую структуру:

Заголовок	Атрибуты	Данные	контроль. Сумма	#26
010101001	000111001	10101010101001010010100101010	1001001010101001	11010

Рис.1.3.4.

В то же время наибольшее распространение получили файловые системы, ориентированные на хранение данных на магнитных дисках. Они имеют структуру, использующую механизм каталогов. В недавнем прошлом широко использовалась файловая система FAT16, затем на смену ей пришла FAT32. В настоящее время для операционной системы Windows широко используется файловая система NTFS.

Принцип работы дисковой системы можно рассмотреть на примере файловой системы FAT16.

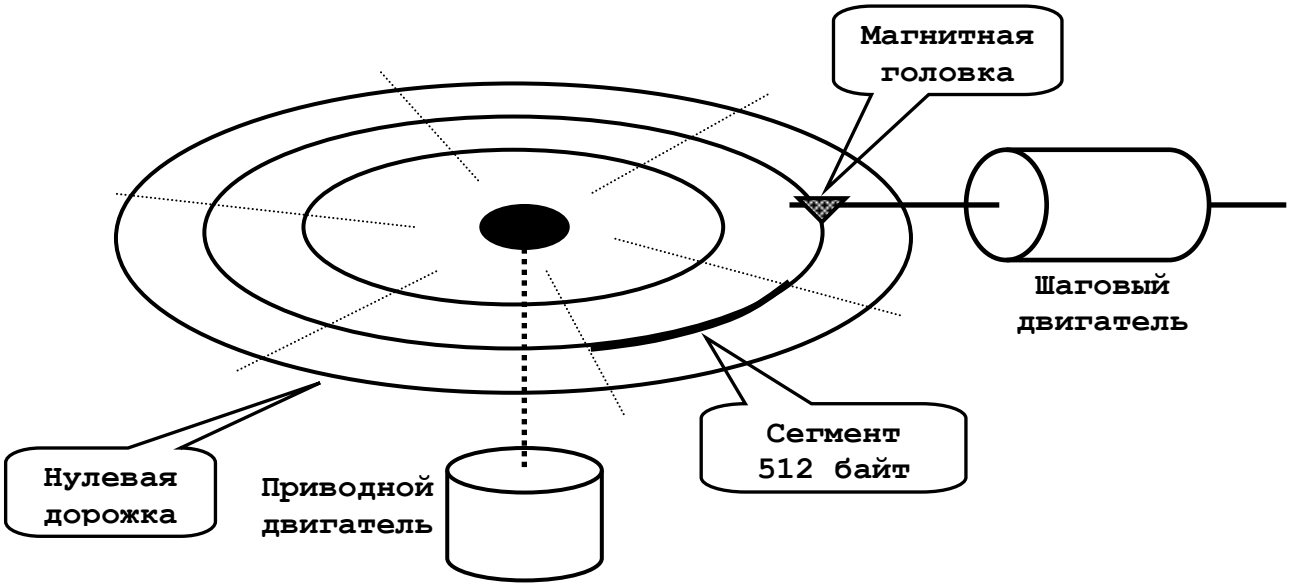


Рис. 1.3.4.

Накопитель на магнитном диске представляет собой электромеханическое устройство. Диск с нанесенным магнитным слоем магнитной головкой, управляемой шаговым двигателем, размечен так, что на нем образуются несколько сотен концентрических магнитных дорожек. Каждая дорожка разделена на одинаковое количество сегментов. Каждый сегмент дорожки - это неделимая далее часть хранения данных. Объем сегмента - 512 байт.

Чтобы определить местоположение определенного файла, на нулевой дорожке на диске располагается таблица размещения файлов (FAT), в которой записаны имена файлов, их атрибуты (размер, время создания, тип: архивный, скрытый, системный и т.д.) и номера сегментов, с которых начинается файл.

В файловой системе FAT16 для адресации секторов отведено 16 разрядов, что эквивалентно 65536 адресам сегментов. Таким образом, в системе FAT16 максимальная емкость диска составляет 65536*512 ≈ 33 Мбайт.

Но с увеличением емкости жестких дисков (HDD) пришлось модернизировать систему адресации. Для этого была введена более крупная единица памяти - кластер, объединяющий 2 или 4, 8, 16, 32 или 64 сегмента. Тем самым в рамках FAT16 можно использовать HDD емкостью до 2,1 Гбайт.

Большие потери дискового пространства из-за кластерного строения привели в 1997 году к созданию файловой системы FAT32, в рамках которой можно иметь 232 = 4294967296 адресов. В системе FAT32 можно использовать HDD до 160 Гбайт. В настоящее время файловая система NTFS позволяет работать с дисками терабайтной емкости.

1.3.5. Операционные системы

Операционные системы предназначены для управления исполнением программ, для управления файловой системой, для управления ресурсами ЭВМ. Операционные системы - это программные комплексы из большого количества различных файлов, связанных между собой.

В секторе OS ведущее положение занимают несколько крупных софтверных фирм:

- Microsoft: MS DOS, Windows-95, 98, 2000, XP, NT, Vista, 7.
- IBM: OS/2, OS/400.
- Apple: MacOS: (версии 8,...,10 Leopard).
- Bell: Unix.
- Другие фирмы: FreeBSD, Linux, BeOS, NetWare, Solaris и т.д.
- Для портативных компьютеров: Windows Mobile, Symbian OS, IPhone OS и др.

По способу взаимодействия OS разделяют на пакетный и диалоговый режимы работы. По реализации интерфейса различают текстовый и графический.

Текстовый режим используется в MS DOS, Unix, Linux и некоторых других OS. Графическими оболочками снабжены почти все существующие OS: Windows, MacOS и т.д.

Приведем в качестве примера работу в диалоговом текстовом режиме в операционной системе MS DOS:

При вводе команды: **`C:>DIR D:`** система выполнит вывод на экран списка файлов, находящихся на диске D.

При работе с файловой системой операционная система присваивает устройствам компьютера следующие имена:

- Дисковод для гибких дисков: A: , B:
- Жесткий диск: C: , D: , E:
- Оптический дисковод CD-ROM: F:
- Последовательный порт: COM1: , COM2:
- Параллельный порт: LPT1: , LPT2:
- Консоль терминала: CON:

- Нулевой порт: NUL:

Тогда полное имя доступа к файлу выглядит следующим образом:

C:\ARC\ZIPFILE\myhelp.zip

Для удобства работы с OS разработаны т.н. операционные оболочки, такие как Norton Commander, Total Commander, FAR и им подобные. В этом случае управление файловой системой выполняется не только вводом команд, но и использованием режима меню, а также наборов функциональных клавиш, манипулятора мышь и ряда других интерфейсных элементов, позволяющих наглядно и оперативно управлять операционной системой.

Различают однозадачные и многозадачные OS, вторые позволяют реализовать либо полную многозадачность (параллельное выполнение задач), либо вытесняющую многозадачность (приостановка одного приложения и запуск другого).

1.3.6. Инструментальные системы

Для разработки программных продуктов требуется специальное программное обеспечение – инструментальные системы (ИС), позволяющие сократить затраты и обеспечить качество и надежность программных продуктов. В ИС входят средства создания приложений: компиляторы языков программирования, интерпретаторы, редакторы текстов программ, отладчики, средства поддержки и масса других специализированных программ.

Языки программирования условно можно разделить на следующие классы:

1. Машинные языки – computer language.
2. Машинно-ориентированные языки - computer oriented language – ассемблеры.
3. Алгоритмические языки – algorithmic language – языки, независимые от архитектуры компьютера: Fortran, Pascal, Delphi, C, C++, Visual Basic, и т.п.
4. Процедурно-ориентированные языки - procedure oriented language – для создания программ как совокупности процедур (подпрограмм): dBase, Basic, SQL, Visual Fox Pro и т.д.
5. Языки программирования для Интернета, называемые Script (скрипт-языки) интерпретирующего типа, такие как HTML – язык разметки для создания WEB страниц, JavaScript, PHP – для создания скриптов и ряд других.
6. Проблемно-ориентированные языки для решения определенных задач специального назначения: LISP – обработка списков, Simula, SmallTalk – язык объектно-ориентированного программирования, Prolog – решение задач искусственного интеллекта, Ada (имя первой программистки - Августа Ада Байрон) – для применения в военной сфере, UML – язык моделирования процессов и множество других языков.

Специальные программы для перевода программных текстов и их дальнейшего исполнения на компьютере разделяются на компиляторы и интерпретаторы.

Интерпретатор (interpreter) в процессе исполнения выбирает очередной оператор из текста программы и немедленно его выполняет (интерпретирует), после чего переходит на выполнение следующего оператора. Программа не может быть запущена без интерпретатора. Исполнение программы происходит медленно.

Компилятор (compiler) сначала переводит весь текст исходной программы на машинный язык (трансляция), а затем машинный вариант программы исполняется без участия компилятора. В результате законченная программа получается эффективной и быстрой, работает без вспомогательных программ и может быть перенесена на другие компьютеры. Недостаток компилятора – его высокая сложность и отсюда относительная медленность процесса трансляции. Процесс компиляции программы состоит из нескольких этапов:

Препроцессор выполняет директивы препроцессора, включает содержимое файлов, указанных в директивах #include, подготавливает текст программы к трансляции.

Компиляция программного текста в файл на языке ассемблера и дальнейшее преобразование в объектный файл.

Программа компоновщика (Linker) добавляет в объектному файлу коды функций из библиотек и формирует исполняемый файл с расширением ".exe".

Исполняемый файл может быть загружен в память компьютера и выполнен без участия компилятора.

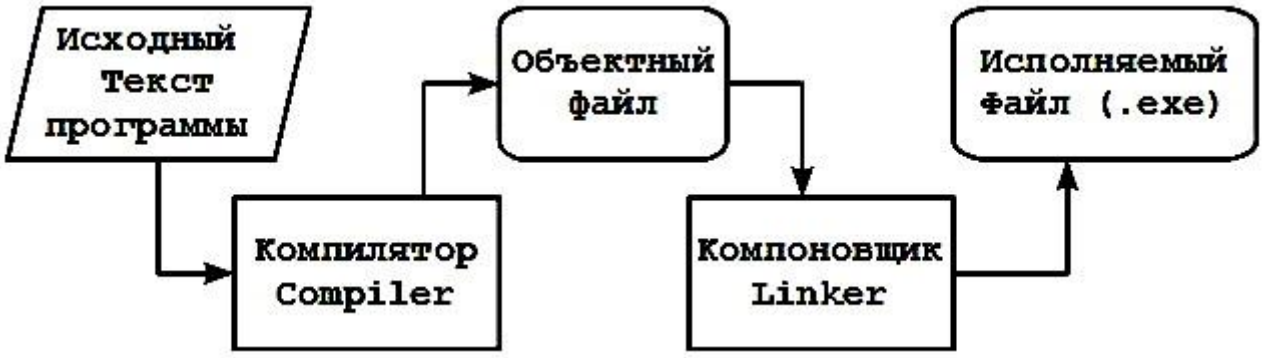


Рис. 1.3.6.

Программы - отладчики (debugger) предназначены для анализа выполнения программ и исправления в них ошибок.

2. Основы программирования

2.1. Алгоритмы и их свойства

Алгоритм - это определённая последовательность команд, при выполнении которых можно получить определенный результат. Алгоритм может быть некоторой последовательностью вычислений, а может - последовательность действий нематематического характера.

Для любого алгоритма справедливы следующие свойства алгоритма.

1. Дискретность - свойство алгоритма, при котором его исполнение разбивается на отдельные элементарные действия, число которых конечно.

2. Понятность - каждое элементарное действие должно быть понятно исполнителю.

3. Детерминированность - каждое элементарное действие должно выполниться только строго определенным образом, и никак иначе.

4. Массовость - применимость данного алгоритма к целому классу задач.

5. Результативность - любой алгоритм обязательно должен приводить к результату.

Алгоритм может быть записан различными способами: на естественном языке в виде описания; в виде графических блок-схем; на специальном алгоритмическом языке.

2.2. Способы описания алгоритмов

Сначала два определения:

- Правила, определяющие структуру текста, называются синтаксисом языка.
- Правила, управляющие смыслом текста, называются семантикой языка.

2.2.1. Метаязык Бэкуса-Наура (язык БНФ)

Для описания синтаксиса алгоритмического языка требуется, в свою очередь, соответствующий язык. Такой язык назовем метаязыком (над'языком). Для этой цели не подходят естественные языки из-за их многозначности, и поэтому используются специальные, так называемые формальные языки. Наиболее распространенными являются язык металингвистических формул Бэкуса-Наура (БНФ) и язык синтаксических диаграмм Н. Вирта.

Язык БНФ похож на математические формулы и поэтому его называют языком метаформул. В левой части метаформулы указывается понятие, а в правой – множество значений этого понятия. Все понятия (метапеременные) обычно заключаются в угловые скобки **< >**.

Например, понятия: **<Число>** или **<Арифметическое выражение>**.

Левая и правая часть соединяются знаком **::=** (метасимвол) – "это есть".

Все возможные значения метапеременной разделяются вертикальной чертой, **|** - (либо, или).

Пример: **<Цифра> ::= 0|1|2|3|4|5|6|7|8|9**

Определим выражения из двух переменных, **A** и **B**. Для этого напишем определение синтаксиса на языке БНФ:

```
<переменная>::=A|B
<выражение>::=<переменная>|<переменная> + <переменная >|
<переменная> - <переменная>
```

В соответствии с этим определением синтаксиса выражения могут иметь следующий вид:

A, B, A+B, A+A, B+A, B+B, A-A, A-B, B-A, B-B

Другой пример иллюстрирует описание синтаксиса для представления двоичных кодов, то есть любых непустых последовательностей нулей и единиц:

<двоичная цифра>::= 0 | 1

<двоичный код>::= <двоичная цифра>|<двоичный код> <двоичная цифра>

Для задания синтаксических конструкций произвольной длины используются два метасимвола "**{**" и "**}**". Заключенная в эти скобки конструкция может повториться нуль или более раз.

<двоичный код>::= <двоичная цифра>|{< двоичная цифра >}

Может использоваться также метапеременная **< пусто >** .

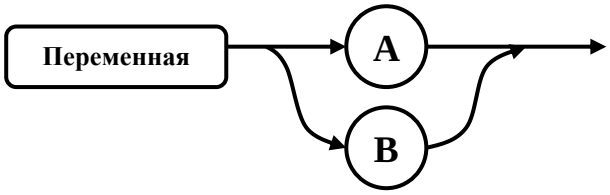
Нотация БНФ широко используется в описании синтаксиса компиляторов и интерпретаторов.

2.2.2. Синтаксическая диаграмма Н. Вирта

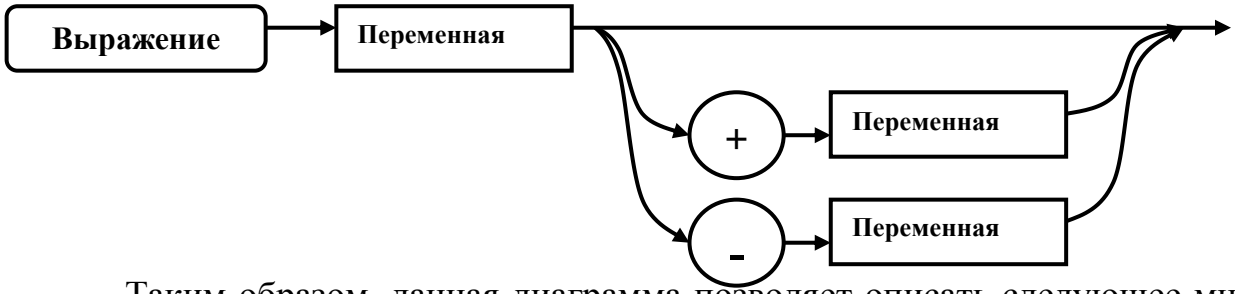
Используется для графического изображения структуры синтаксических конструкций. Основной элемент диаграммы: основной символ или понятие языка.

Из каждого элемента выходит одна или несколько стрелок, указывающих на элементы, непосредственно следующих за данным элементом.

Например, определение переменной может быть выполнено следующим образом:



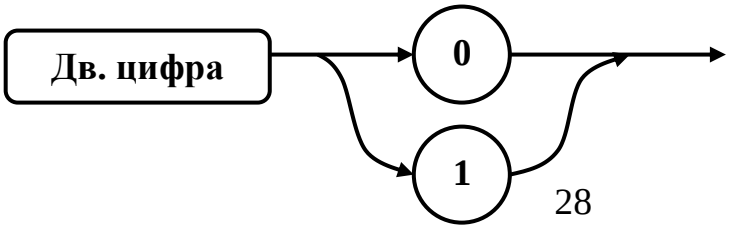
В свою очередь, арифметическое выражение определяется следующей синтаксической диаграммой:

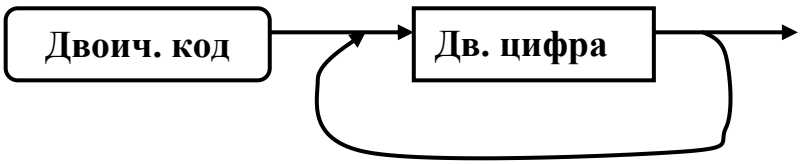


Таким образом, данная диаграмма позволяет описать следующее множество арифметических выражений:

A, B, A+B, A+A, B+A, B+B, A-A, A-B, B-A, B-B.

С помощью синтаксической диаграммы удобно задавать и конструкции произвольной длины, например, определить такое понятие, как двоичный код:





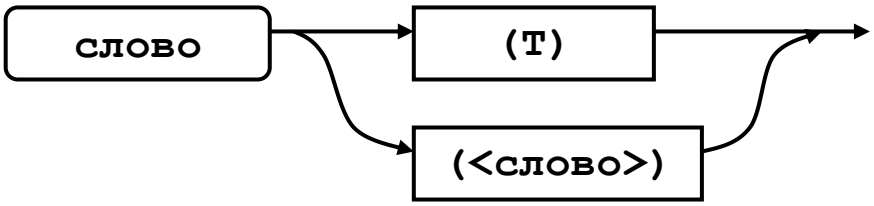
Еще один пример показывает так называемое рекурсивное определение, когда определяемый элемент выражается через самого себя. Например, необходимо отобразить понятие, под которым подразумевается буква Т, заключенная произвольное количество раз в круглые скобки:

(Т), ((Т)), (((Т)))

В БНФ такое определение выглядит таким образом:

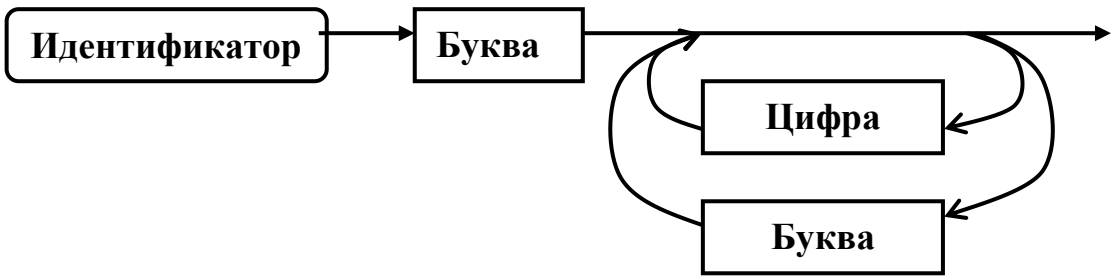
<слово>::=(Т)|(<слово>)

Синтаксическая диаграмма:



Метаязык БНФ более строг и точен, язык синтаксических диаграмм более нагляден. Например, идентификатор переменной, используемый в языках программирования, должен начинаться обязательно с буквы, и может состоять из латинских букв и цифр:

<Идентификатор>::= <Буква>|<Идентификатор><Цифра>|<Идентификатор><Буква>



2.2.3. Графические блок-схемы

Еще один, очень распространенный способ описания алгоритмов - это использование графических блок-схем. Он обладает высокой наглядностью, но для сложных алгоритмов довольно громоздок. Тем не менее, этот способ широко используется для описания алгоритмов. Смотрите рис. 2.4.1.

2.3. Язык программирования высокого уровня С++

Язык **C** был разработан в 1972 году сотрудником Bell Laboratories Денисом Ричи для использования в разработанной им совместно с Кеном Томпсоном операционной системой Unix. На языке **C** написан компилятор этого языка, а также операционная система Unix для мини-ЭВМ PDP-11. Этот язык получил развитие в виде созданного фирмой Borland для семейства микропроцессоров 8086, 8088 и операционной системы MS DOS языка программирования под названием "Turbo-C".

Затем в начале 80-х годов также сотрудник Bell Laboratories Бьерн Страуструп (см. фото) на основе языка **C** разработал значительно более мощный язык **C++**. Этот язык реализует объектно-ориентированный подход к программированию. В конце XX века **C++** приобрел статус стандартного языка программирования.

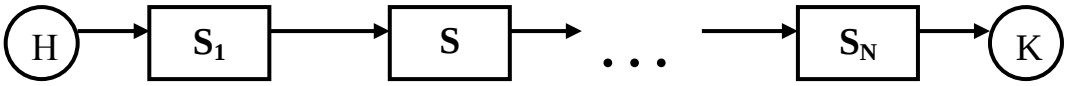
В начале XXI века появился еще один преемник языка **C** – это **C#** (произносится: си шарп). Музыкальный знак диез указывает на повышение тона. Этот язык предложен фирмой Microsoft как конкурент языка Java и представлен как язык компонентной сборки. Его дальнейшая судьба прояснится со временем.

В настоящее время язык C++ является мощным и эффективным средством разработки разнообразного программного обеспечения.

2.3.1. Основные принципы языка C++

Язык программирования C++ создан на основе нескольких базовых принципов, которые в настоящее время характерны для современных языков.

1. Используется структурное программирование – это метод проектирования программ в виде последовательной структуры функционально законченных блоков, исполняемых один за другим.



2. Реализован принцип проектирования: "Сверху – вниз". Это значит, что сначала проект разбивается на несколько простых задач, решаемых отдельно, т.е. для каждой задачи создается отдельный программный модуль. Затем следует сборка модулей в единый программный продукт.

3.Применяются концепции объектно-ориентированного программирования. Сущность их состоит в следующем:

- Использование классов – типов, в которых объединяются структуры данных и методы их обработки с ограничением доступа к данным класса. Эта концепция называется инкапсуляцией. Конкретные переменные типа "класс" называются объектами.
- Наследование - это механизм создания новых классов, использующих свойства и методы предыдущих классов.
- Использование сходных методов для разных классов и объектов называется полиморфизмом.

4. В язык C++ встроена возможность расширения базовых конструкций языка за счет использования большого количества внешних библиотек.

5. Предусмотрены способы переносимости программ с небольшими изменениями на другую операционную или аппаратную среду.

Основное назначение языка **C++** - это системное программирование. Поэтому считается, что это относительно низкоуровневый язык. По объему и скорости выполнения программы, написанные на C++, приближаются к программам, написанным на ассемблере.

2.3.2. Структура программы на языке C++

Приведем два определения:

- Базисные элементы языка называются лексемами: **for, if, while, sizeof, far, const, ...**
- Слова, используемые для именования любых объектов в программе, называются идентификаторами.

Программа на C++ состоит из директив препроцессора, объявлений глобальных переменных, одной главной функции **{main()}** и совокупности неглавных функций. Пример программы:

```
/* Заголовки и комментарии */
/* директивы препроцессора */
#include <stdio.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>
#define PI 3.14159
/* глобальные переменные */
int Count;
// ---------- Неглавная функция ---------------
int Mult(int X, int Y) { return X*Y;}
// ---------- Главная функция ---------------
void main()
{ int A,B;                  // Локальные переменные
  A=2; B=3;                 // Операторы
  Count=Mult(A,B)/(A+B);    // Операторы
  prinf("Count = %i\n", Count);
}
```

Функции, вызываемые из функции **main()**, имеют ту же структуру, что и главная функция. Препроцессор, который входит в состав компилятора, выполняет дополнительные действия над текстом программы, определяемые директивой **'#'.** В частности, директива **#include <имя библиотеки>** включает в текст программы прототипы библиотечных функций. Чтобы включить содержимое из файла, находящегося на диске, необходимо вызов записать следующим образом:

```
#include "mylib.cpp"
```

Директива компилятора **#define <stroka1> <stroka2>** в тексте программы все встреченные включения **<stroka1>** заменяет на **<stroka2>**.

Система программирования Borland C++ 3.1., реализованная в виде интегрированной среды, состоит из следующих компонентов:

- Компилятор BIN\BC.exe
- Редактор текста
- Редактор связей (компоновщик, сборщик)
- Библиотеки различного назначения
- Файлы документации *.doc

2.3.3. Типы данных языка C++

В математике мы различаем переменные в соответствии с их характеристиками, т.е. вещественные, целые, логические, множества и т.д. Программа, являясь моделью явлений внешнего мира, представляет его характеристики, выражая их в виде типизированных данных.

C++ является языком с сильной типизацией, т.е. все данные, получаемые извне, должны принадлежать заранее известному типу. В нем имеются и стандартные (предопределенные) типы, и существует возможность создания собственных типов данных, наиболее подходящих для конкретных практических приложений.

Например, подсчет количества каких-либо объектов в реальном мире возвращает данные целого типа, а операции измерения (т.е. сравнения с эталоном) – данные вещественного типа. В то же время потоки текстовой информации формируют данные символьного типа, а реакции на события – данные логического типа.

Рассмотрим более подробно типы данных, применяемых в практике программирования на языке C++.

2.3.4. Стандартные типы данных

Различают базовый тип данных и модификатор типа. К базовым типам относятся char (символьный), int (целый), float (вещественный с одинарной точностью), double (вещественный с двойной точностью), void (пустой тип).

Модификаторы: unsigned (беззнаковый), signed (знаковый), short (короткий), long (длинный).

Спецификатор типа – одно или несколько ключевых слов, определяющих тип переменной, например: signed short int, long int, unsigned char и т.д. По умолчанию целые типы определяются как signed.

- Данные целого типа:

Спецификатор типа	Объем, байт	Диапазон Представления числа	Примечание
Signed int	2	-32768 . . 32768	целое знаковое
unsigned int	2	0 . . 65535	целое без знака
Signed long int	4	-2147483648 . . 2147483647	целое длинное зн.
unsigned long int	4	0 . . 4 294 967 295	целое длинное без.

Знаковый тип представлен в дополнительном коде.

- Данные вещественного типа:

Спецификатор типа	Объем, байт	Диапазон представления числа	Точность, знаков
Float	4	1.17E-38 ... 3.37E+38	7
Double	8	2.23E-308 ... 1.67E+308	15
Long double	10	3.37E-4932 ... 1.2E+4932	19

- Данные символьного типа **(char)** занимают в памяти 1 байт и кодируются на основе международного кода ASCII. Например, символ 'A' имеет значение: **10000001**. Символы можно сравнивать между собой отношениями: **==, <, >, <=, >=, !=**.

Константой называется переменная, значение которой не может быть изменено в процессе выполнения программы. Константы записываются в следующем виде:

Целая десятичная константа - целое число, начинающееся со значащей цифры: **123, +45, -317, 883, 19** и т.д.

Целая 8-ричная константа начинается с нуля: **+017,-087,0197,...**

Целая 16-ричная константа начинается с '0x': **0xF,-0x4A6EC2, ...**

Вещественная константа должна содержать десятичную точку или букву E(e): **-0.13, +4612.09, +5., 3.14159, 0.314159E+1, ...**

Символьная константа: **'A','9','+','%','#','k','q', ...**

Строковая константа: **"while","КЛАСС","AAE","A+B*C",...**

2.3.5. Указатели.

Указатель – особый вид переменной, которая хранит адрес элемента памяти, в котором может быть записано значение другой переменной.

Определение указателя:

<тип> * <имя переменной указателя>;

Здесь ***** - операция доступа по указателю.

Обратная операция – взятие адреса: **&.**

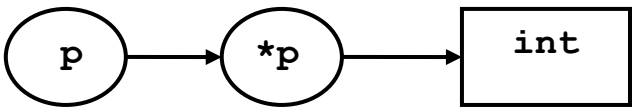
NULL – нулевое значение указателя, которое никуда не указывает.

Приведем примеры:

```
int *p;        // Объявляем указатель p. Пока он никуда не указывает.
int A;         // Объявляем переменную A.
A=19;          // Присвоим ей значение 19.
p=&A;          // Указатель p теперь указывает на переменную A.
int *t;        // Объявляем указатель p. Пока он никуда не указывает.
int B;         // Объявляем переменную B.
B=5;           // Присвоим ей значение 19.
t=&B;          // Указатель t указывает на переменную B.
int C;         // Переменная C.
C=*p+*t;       // C = 19 + 5 =24; Содержимое переменной по
               // указателю p, учитывая, что *p=19, и содержимое
               // переменной по указателю *t=5, складываются и
               // результат записывается в переменную C.
p=NULL;        // Теперь указатель p никуда не указывает.
```

Возможно создание указателя на указатель:

int **p;



С указателями можно выполнять операции.

1. Изменение указателя на следующий или предыдущий адрес: **p+i p-i p++ p--**
   Значение указателя будет увеличиваться или уменьшаться на размер элемента.
2. Указатели могут быть сравнены между собой на равенство и неравенство: **p>t p==t p!=t p==NULL** и т.д.

Следующий фрагмент иллюстрирует изменение указателей:

```
int D[7]={12,14,19,46,53,11,3};
int p;
p=&D[0];
for (int i=0; i<7; i++) {cout<<*p<<" "; p++};
cout<<endl;
```

2.4. Операторы языка программирования C++

Оператор – законченная фраза языка программирования, которая определяет некоторый этап обработки данных. Все операторы разбиваются на 2 группы:

- основные операторы, не содержащие других операторов;
- производные, в состав которых входят другие операторы: составной оператор, оператор процедуры, оператор цикла. Составной оператор заключается в операторные скобки: **{ }**.

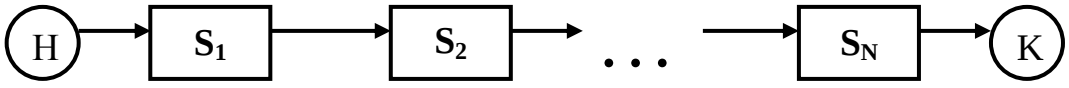
2.4.1. Принципы структурного программирования.

Еще в 60-х годах прошлого века выдающийся советский ученый, впоследствии академик АН СССР, А.П.Ершов предложил для разработки программ использовать так называемый операторный метод, в рамках которого он показал преимущества использования предопределенных структур.

Известный американский ученый Эдсгер Дейкстра в 1965 году предложил убрать из языков программирования оператор безусловного перехода **goto**, а в дальнейшем он разработал принципы структурного программирования. Базовым в предложениях и А.П.Ершова и Э.Дейкстры является возможность включения одних структур в другие, т.е. реализация многоуровневой вложенности структур.

Э.Дейкстра предложил использовать при проектировании программ 4 базовых структуры, или *конструкты*, как он их назвал:

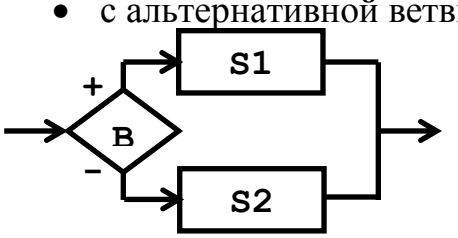
1. Конструкта "Следование".



Все операторы выполняются друг за другом. Эта конструкта реализует основной принцип фон Неймана – последовательное выполнение команд компьютером.

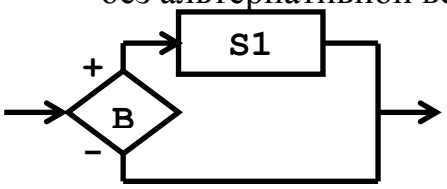
2. Конструкта "Ветвление". Имеет две формы:

- с альтернативной ветвью;




```
If B then S1
     else S2;
```

- без альтернативной ветви.

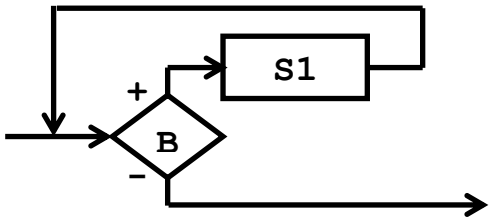



```
If B then S1;
```

3. Конструкта "Цикл".

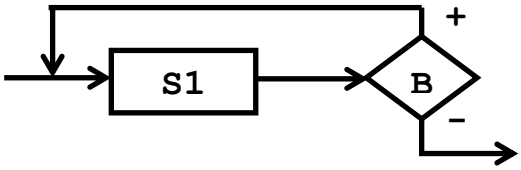
Эта конструкта также имеет две формы:

Цикл с предусловием:



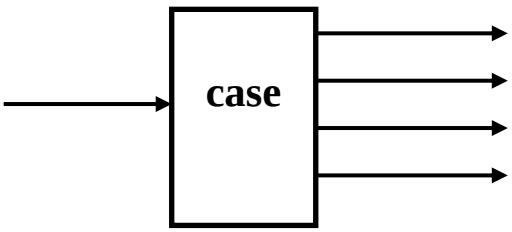

```
while B do S1;
```

и цикл с послеусловием:




```
do S1 while B;
```

3. Конструкта "Выбор". В зависимости от выполнения условия происходит выбор той или иной альтернативы:




```
switch (A)
{ case 0;
  case 1;
  case 2;
}
```

Конструкты могут быть вложены друг в друга. Например, в следующей задаче: определить в диапазоне чисел от 1 до N количество чисел, кратных заданному числу M. Составим блок-схему алгоритма (Рис. 2.4.1.):

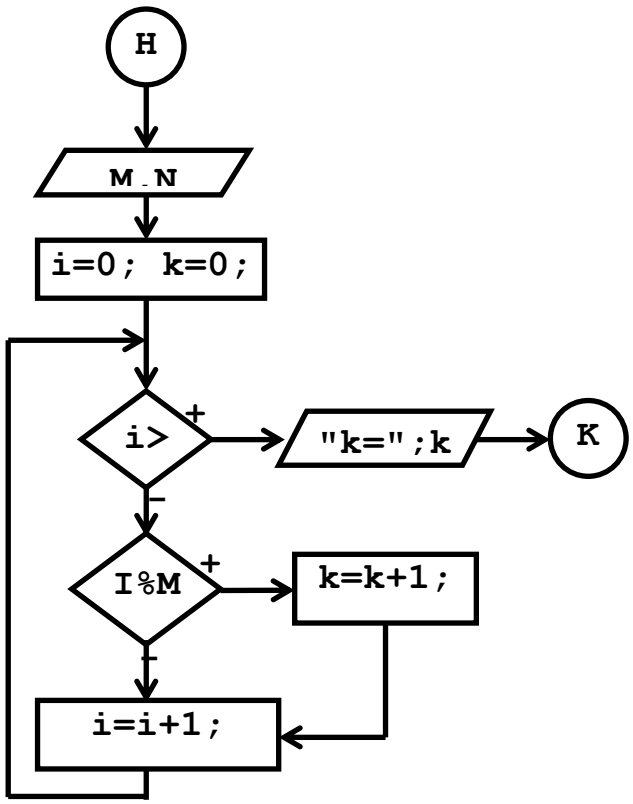


Рис. 2.4.1.

На основе данной блок-схемы можно составить программу:

```
void main ()
     { int n, m, k, i;
       cout<<"N="; cin>>n;
       cout<<"N="; cin>>n;
       k=0;
       for (i=0; i<n; i++;)
         if (i%m==0) k++;
       printf("K = %i\n",k);
       getch();
     }
```

2.4.2. Оператор ветвления

Позволяет изменять ход вычислительного процесса в зависимости от выполнения определенных, заранее заданных условий.

В языке C++ используется оператор ветвления **`if`** в двух формах и оператор выбора **`switch`**.

Оператор if имеет следующий синтаксис:

```
if (<условие>) <оператор-1>;
[ else <оператор-2>];
```

При выполнении оператора вначале вычисляется выражение условия. Если результат равен значению "Истина (true)", т.е. любое, отличное от нуля значение, то выполняется оператор-1. Если же результат равен значению "Ложь (false), т.е. равен 0, то выполняется оператор-2. В качестве условия может быть использовано арифметическое, логическое выражение, сравнение, целое число и т.д.

Другой оператор, switch, позволяет выбрать один из нескольких вариантов. Пример реализации простейшего калькулятора:

```
// Easy Calculator - Простой калькулятор
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#include <math.h>

void main()
 { clrscr();
   int f; float x,y,res; char op;
   cout<<"\nInput x, operator, y : ";
   cin>>x>>op>>y;
   f=1;
   switch (op)
     { case '+': res=x+y; break;
       case '-': res=x-y; break;
       case '*': res=x*y; break;
       default : cout<<"operator unknown!\n";f=0;break;
     }
   if (f==1)
     printf("%.3f %c %.3f = %.3f.\n",x,op,y,res);
   getch();
}
```

2.4.3. Операторы повторения

Многократно повторяющийся участок вычислительного процесса называется циклом. Каждый проход по телу цикла называется итерацией. Цикл, не содержащий других циклов, называется простым, иначе его называют кратным.

Переменную, управляющую циклическим процессом, называют параметром цикла. Если априори известно количество повторений, то организуют циклы со счетчиками. Итерационным называют циклический процесс, в котором количество повторений неизвестно и условие продолжения цикла базируется на результатах текущих вычислений.

Если результаты вычислений очередной итерации используются при вычислении значений в следующей итерации, то это говорит об использовании

рекуррентных соотношений. Как показывает Дональд Кнут, рекуррентность, или возвратность, это свойство решения задачи, когда ее решение зависит от решения той же задачи, но меньших размеров.

В языке C++ представлено три формы операторов цикла:

a) Цикл со счетчиком:

for (<инициализация параметра>**;** <условие>**;** <закон изменения>**)**
<оператор>**;**

```
for (int i=0; i<n; i++) sum+=i*i;
```

b) Цикл с предусловием:

while (<условие>**)** <оператор>**;**

```
while (a!=0) { k++; a/=10;};
```

c) Цикл с послеусловием:

do <оператор> **while (**<условие>**);**

```
do {sum+=a%2; a/=10;}  while (a!=0);
```

В цикле for можно использовать любые ординарные переменные, например, символьного типа:

```
char let;
for (let='z'; let>='a'; let--)
    printf("Код - %i,  символ - %c\n",let,let);
```

2.5. Механизмы циклического процесса

С циклическим процессом могут быть связаны несколько программных механизмов. Наиболее применимы из них следующие:

- Накопитель суммы или произведения;
- Счетчик. Может быть суммирующим, вычитающим, реверсивным, с предустановкой и т.д.;
- Пороговый элемент, служит для определения минимального или максимального значения в потоке данных;
- Фиксатор события в циклическом процессе, имеющий два значения: false или true, 0 или 1.
- Триггер переключает некоторую переменную с одного значения на другое и наоборот, например с 0 на 1, в следующем цикле с 1 на 0.

Накопитель суммы представляет собой переменную, при инициализации которой присвоено нулевое значение, а накопитель произведения инициализирован единицей. В каждом цикле к накопителю добавляется очередное значение, и таким образом, по окончанию цикла накопитель будет содержать сумму всех слагаемых, а накопитель произведения будет содержать произведение

всех значений, полученных в циклическом процессе. Например, сумма только положительных чисел:

```
// sum=sum+a;  или  sum+=a;
sum = 0;
for (i=0;i<n;i++) if (a[i]>=0) sum += a[i];
```

a[i] – переменная с очередным слагаемым.

Счетчик. Может быть суммирующим, вычитающим, реверсивным, с предустановкой и т.д. Его работа аналогична накопителю суммы, только в каждом цикле добавляется единица. Например, количество отрицательных чисел в потоке:

```
// count = count + 1; или  count += 1; или count++;
count = 0;
for (i=0; i<n; i++) if (a[i]<0) count++;
```

Пороговый элемент служит для определения минимального или максимального значения в потоке данных. Начальное значение переменной, которая исполняет роль порогового элемента, обычно равно любому значению из потока данных, среди которых ищется минимальное или максимальное значение. Если значение очередного элемента в потоке данных будет больше значения порогового элемента, то пороговый элемент принимает значение текущего элемента, т.е. порог поднимается. Иначе его значение остается без изменений. По выходе из цикла значение порогового элемента будет в этом случае равно максимальному элементу из потока данных. Например, максимальное число в потоке:

```
PEmax = a[0];
for (i=0;i<n;i++) if (a[i]>PEmax) PEmax=a[i];
```

a[i] – переменная с очередным значение из потока.

Фиксатор события в циклическом процессе фиксирует некоторое событие, которое может произойти или не произойти в этом процессе. Например, появится ли в потоке данных отрицательное число или нет. Перед циклом фиксатор принимает значение false. Если в циклическом процессе произойдет событие, которое надо зафиксировать, то фиксатору придается значение true. Далее это значение остается до конца цикла. Если событие произошло, то значение фиксатора будет true, если не произошло, то фиксатор так и останется в состоянии false. В C++ иногда логические значения true и false заменяют на 1 и 0.

```
fix = 0;
for (i=0;i<n;i++) if (a[i]<0) fix=1;
```

Триггер – переменная, которая может иметь одно из двух значений, которое меняется на противоположное с каждым новым циклом. Например, сложить все числа из входного потока, каждый раз с противоположным знаком. Первое слагаемое положительное. Начальное значение триггера равно 1.

```
trigger=1; sum = 0;
for(i=0;i<n;i++)
  {sum += trigger*a[i];trigger=-trigger;}
```

Эти механизмы можно использовать в одном циклическом процесс в разных сочетаниях.

В качестве примера рассмотрим реализацию решения следующей задачи: для значений функции y=f(x) на отрезке (L,R), имеющем n точек, определить среднее всех значений функции, максимальное значение функции, и определить наличие отрицательных значений функции.

2.6. Функции ввода-вывода

Для операций ввода-вывода на C++ можно использовать функции двух библиотек ввода/вывода — стандартной (scanf, printf) и потоковой (cin, cout). Потоковый ввод/вывод удобен в использовании, но работает медленнее стандартного. Поэтому, если в задаче надо считывать много входных данных (скажем, больше мегабайта) или много выводить, то не следует использовать потоковый ввод/вывод.

Приведем несколько функций ввода-вывода из стандартных библиотек:

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

Функция вывода "printf" выводит форматированные данные в стандартный поток (экран):

```
printf("<шаблон формата>",<список вывода>);
```

Шаблон формата задает тип выводимого значения, точность, положение в строке вывода. Знак '%' - это префикс спецификатора формата. Приведем несколько примеров вывода:

```
int a=197; float b=3.14159; char c='Z';
char s[10] = "Apple";
printf("A=%i  B=%6.3f  C=%c  S=%s \n", a, b, c, s);
```

В строке **'printf'** значение переменной **'a'** согласно шаблону будет выведено как целое число, значение **'b'** как вещественное число длиной 6 знакомест с 3 знаками после запятой с округлением. Значение **'c'** - как символ, а значение **'s'** - как строка. Таким образом на экране получим следующий список значений:

```
 A=197  B=3.142  C=Z  S=Apple
```

Функция **"scanf"** выполняет форматированный ввод данных из входного потока (клавиатура):

```
scanf("%i", &a);
```

В результате в переменную **'a'** будет записано значение, принятое с клавиатуры. Здесь **&a** - адрес ячейки памяти, где размещается переменная **'a'**.

Функция **'gets()'** считывает символьную строку с клавиатуры:

```
gets(s);
```

В потоковой библиотеке **#include <iostream.h>** операция вывода в выходной поток выглядит следующим образом:

```
cout<<"Текст"<<endl;
cout<<[Переменная]<<endl;
```

Параметр **'endl'** выполняет перевод курсора на новую строку.

Оператор вывода может содержать несколько элементов, которые выводятся подряд друг за другом:

```
cout<<"A="<<a<<" B="<<b<<" C="<<c<<" S="<<s<<endl;
```

В этой же библиотеке имеется оператор ввода из входного потока:

```
cin >> a;
cin >> b >> c >> s;
```

Для реализации диалогового режима, в котором программа запрашивает очередное значение переменной, а оператор вводит это значение с клавиатуры, необходимо сочетание операций и ввода, и вывода. Пример:

```
cout<<"A = ";  cin>>a;
cout<<"B = ";  cin>>b
cout<<"C = ";  cin>>c
cout<<"S = ";  cin>>s
```

2.7. Работа в текстовом режиме

В текстовом режиме некоторые библиотечные функции позволяют работать с цветом и координатами текстового экрана. Экран дисплея в текстовом режиме состоит из 25 строк, каждая из них состоит из 80 знакомест. Одно знакоместо предназначено для вывода одного символа.

Приводим небольшой список функций для работы в текстовом режиме.

```
1.clrscr();                          //  Очистка экрана.
2.textcolor(<номер цвета>);          //Установка цвета для вывода
                                       символов.
3.textbackground(<номер цвета>); //Установка цвета для
                                     вывода фона за символом.
4.gotoxy(<координата X>,<координата Y>);//Вывод курсора
                                          на место с координатами X,Y.
5. cprintf("...",...); //Вывод текста с предварительно
                         заданным цветом.
6. 1+rand()%99; //Возвращает целое число в диапазоне от 1 до 99.
```

2.8. Использование символов псевдографики.

Кроме обычных символов латинского алфавита, цифр и знаков, в каждое знакоместо можно выводить так называемые символы псевдографики, на осно-

ве которых на экране можно построить некоторые графические элементы: рамки, диаграммы, линии и т.п. Коды некоторых символов псевдографики приведены в таблице 1.

Таблица 1. Коды символов таблицы ASCII для работы с псевдографикой:

┌ 218	─ 196	┬ 194	─ 196	┐ 191	╔ 201	═ 205	╦ 203	═ 205	╗ 187
│ 179	▒ 176		▉ 219	│ 179	╟ 186		╤ 209		╟ 186
├ 195		┼ 197		┤ 180	╠ 204	╠ 199	╬ 206	╣ 182	╣ 185
│ 179	▒ 177		▓ 178	│ 179	╟ 186		╧ 207		╟ 186
└ 192	─ 196	┴ 193	─ 196	┘ 217	╚ 200	═ 205	╩ 202	═ 205	╝ 188

В качестве примера приводим функцию "**Ramka**" для построения одинарной рамки с координатами левого верхнего угла (x1,y1) и нижнего правого угла (x2,y2).

```
// ------------------------------------------------
// Построение одинарной рамки:
// (x1,y1)- левый верхний угол, (x2,y2)-правый нижний
// ------------------------------------------------
#include "mylib.cpp"
void Ramka(int x1, int y1, int x2, int y2)
  { int i,c;
    // Меняем местами координаты
    if(x1>x2){c=x1;x1=x2;x2=c;};
    if(y1>y2){c=y1;y1=y2;y2=c;};
    // Рисуем рамку
    gotoxy(x1,y1); printf("%c",218);
    for(i=1;i<x2-x1;i++) printf("%c",196);
    printf("%c",191);
    gotoxy(x1,y2); printf("%c",192);
    for(i=1;i<x2-x1;i++)printf("%c",196);
    printf("%c",217);
    for(i=1;i<y2-y1;i++)
      {gotoxy(x1,y1+i); printf("%c",179);};
    for(i=1;i<y2-y1;i++)
      {gotoxy(x2,y1+i); printf("%c",179);};
   }
void main()
{
    clrscr();
    Ramka(12,2,24,8);
    getch();
```

```
}
```

3. Функции

3.1. Определение функции

В основе технологий создания программ на языке C++ лежит понятие функции. Функция, как механизм, является важнейшим средством декомпозиции программ. Мы ранее уже пользовались функциями, в частности в операциях ввода – вывода, при вычислении математических значений и т.д. Собственно и сама программа на C++ является функцией с именем **main().**

Функция – это программный блок, который включает в себя секцию объявлений и определений данных, а также совокупность операторов, ориентированных на реализацию конкретного алгоритма.

В программе функции можно располагать в любой последовательности.

В C++ не разрешается вложенность функций. Спецификация записи функции имеет вид:

```
<спецификатор типа> <имя функции>
([<Список формальных параметров>]) {тело функции};
```

Спецификатор типа определяет тип возвращаемого результата, который может иметь стандартный тип, или тип, определенный пользователем. Возможны функции, не возвращающие никакого результата. Перед именем такой функции ставится тип **void**.

Функция может возвращать только одно значение, которое указывается оператором **return**.

3.2. Оператор `return`

Оператор **return[(<выражение>)];** завершает выполнение функции и передает управление в вызывающую программу.

Оператор **return** в теле функции может быть записан несколько раз.

Если в операторе **return** отсутствует выражение (например: **return;**), то функция ничего не возвращает, поэтому перед ее именем следует поставить тип **void**.

Например, реализуем функцию, определяющую нечетность целого числа, и приведем способ ее использования:

```
// Function odd() - Нечетность
#include "mylib.cpp"
int odd(long int A)
  { if (A%2!=0) return (1); else return (0);
  }
void main()
  { long int B;
    cout<<"\nB = \n"; cin>>B;
    if (odd(B)) printf("Nechet !\n");
    else printf("Chet !\n");
```

```
}
```

3.3. Параметры функции

Параметры функции при ее объявлении называются формальными параметрами (в дальнейшем – параметры), а параметры при ее вызове называются фактическими (в дальнейшем – аргументы).

Их количество и расположение должны быть согласованы.

Как правило, неглавные функции располагаются за главной (**main()**) функцией. Но чтобы компилятор мог правильно определять вызовы функций, в этом случае перед главной функцией помещают прототип вызываемой функции.

Прототип представляет собой заголовок функции без реализации, то есть это своего рода предварительное объявление. Собственно функция описывается в определении, т.е. в разделе полной реализации.

Определение функции может располагаться в любом месте программы. Определение одной функции не может вмещать определение другой функции.

Рассмотрим следующий пример: Задано целое положительное число. Определить, является ли оно простым. (Простым называется целое положительное число больше единицы, которое не делится без остатка ни на одно другое целое положительное число, кроме единицы и самого себя).

```
// Function-1 Prostoe
  #include "mylib.cpp"

int prostoe (long int a);  // прототип функции

void main()                  // Глвная функция
  { long int a;
    cout<<"Prostoe li chislo a = "; cin>>a;
    if (prostoe(a))
         printf("Da !");
    else printf("Net !");
  getch();
  }

int prostoe (long int a)    // реализация функции
   { if (a<=1) return 0;
    for (long int i=2;i<=a/2; i++)
      if (a%i==0) return 0;
    return 1;
   }
```

3.4. Обмен данными между функциями

Функции могут обмениваться данными тремя способами:

1. Используя глобальные переменные.

2. Передавая копии данных.

3. Передавая адреса, по которым записаны данные.

В первом случае глобальные переменные объявляются вне функций и становятся известными во всех нижележащих функциях.

Если же внутри функций будут объявлены локальные переменные с таким же именем, что и глобальные, то приоритет отдается локальным.

Использование глобальных переменных внутри функций не рекомендуется. Следует все данные передавать, используя список параметров.

Локальные переменные видимы только внутри функций. Если имя глобальной и локальной переменной совпадают, то приоритет отдается локальной переменной. Память для них выделяется в стеке и после выхода из функции эта часть памяти освобождается. Параметры, заданные во входном списке функции, также являются локальными переменными. Функция может возвращать только одно значение, определяемое оператором **return**. Для получения доступа к другим переменным, используются переменные типа указатель.

Для того, чтобы из функции можно было возвратить несколько значений, используются указатели (функция меняет местами значения переменных **A** и **B**):

```
void Swap (int &A; int &B) {int C=A; A=B; B=C;};
```

3.5. Рекурсия и рекурсивные алгоритмы

Рекурсия – это способ организации процесса вычислений, при котором функция в ходе выполнения программы обращается сама к себе, прямо или косвенно.

Рекурсия – мощный инструмент разработки программ, позволяющий чрезвычайно компактными средствами определить большие объемы вычислений.

С помощью рекурсии определяются многие понятия. Например, понятие натурального числа можно определить так:

a) 1 есть натуральное число;

b) число, следующее за натуральным, также есть натуральное число.

Известно, что факториал числа n равен произведению всех натуральных чисел от 1 до n.

Таким же образом можно определить функцию факториал:

В этом примере функция определяется через саму себя, т.е. рекурсивно.

```
a) 0! = 1;
b) n! = (n-1)! * n.
```

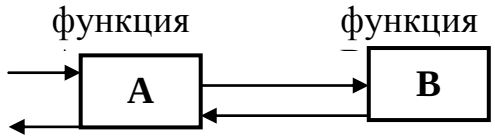
Приведем реализацию функции:

```
long  fact ( long n )
 { if (n==0) return 1;
   return n*fact(n-1);
}
```

Следует отметить, что рекурсию наиболее выгодно использовать для объектов, определяемых рекурсивно: факториал, натуральное число, двоичный

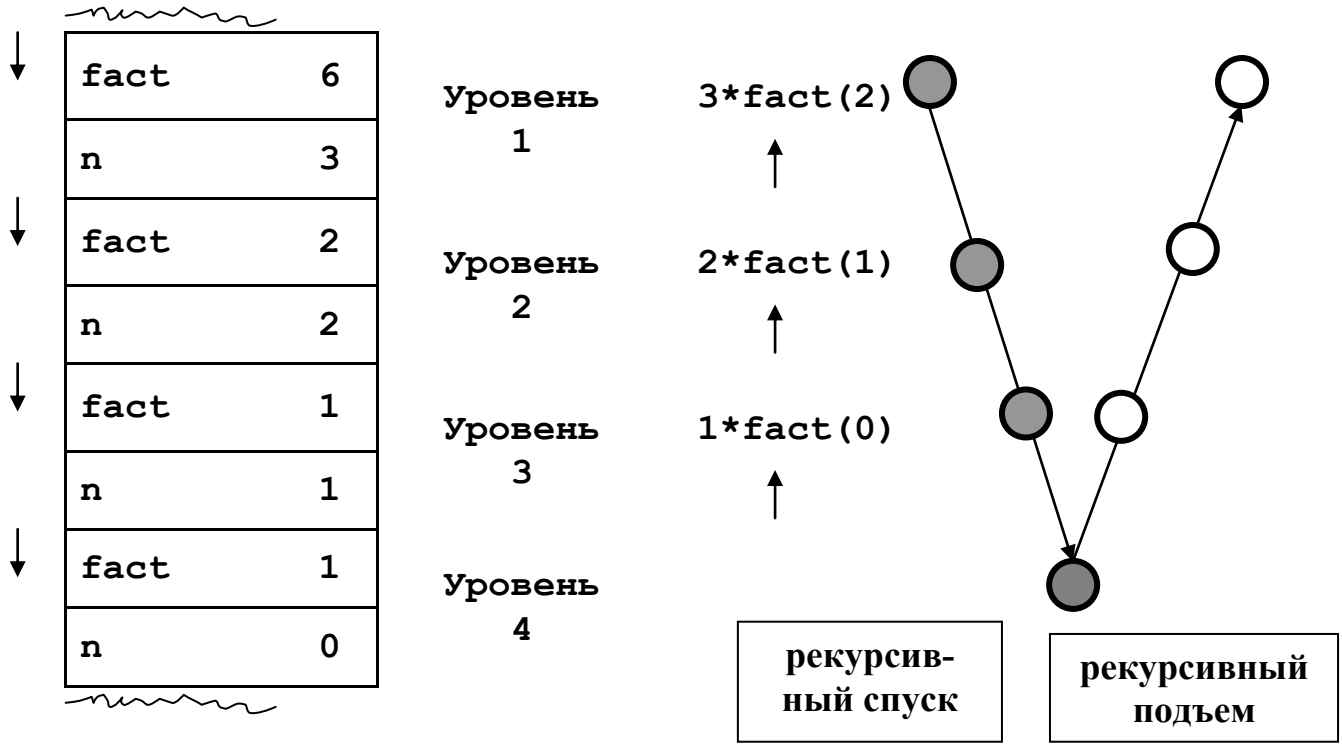
код, дерево, список и т.д. Поскольку глубина рекурсивных вызовов всегда должна быть конечна, в теле рекурсивной функции обязательно должно быть условие окончания этого процесса.

Кроме того возможно создание косвенных рекурсивных процессов, когда функция A вызывает функцию B, а та, в свою очередь, вызывает функцию A.



Рассмотрим более подробно процессы, происходящие в стеке локальных переменных, при вызове рекурсивных функций.

Известно, что при вызове функции в стеке создаются локальные переменные, связанные со списком передаваемых параметров и с именем функции.



При прямом ходе процесса подготавливаются локальные переменные, а вычисления проводятся при обратном возврате. Прямой ход называется рекурсивным спуском, а обратный ход – рекурсивным подъемом. Количество вызовов функции в данном процессе называется глубиной рекурсии.

В качестве примера приведем еще две рекурсивные функции:

```
// ------------------------------------------------
// Функция kcc возвращает колич.цифр числа n.
   int kcc(long n)
        {if (n<10) return 1; return kcc(n/10)+1;}
// ------------------------------------------------
// Функция scc возвращает сумму цифр числа n.
   int scc(long n)
```

```
{if (n<10) return 1; return scc(n/10)+n%10;}
```

3.6. Перегрузка функций

Синтаксис языка программирования обычно требует, чтобы каждая функция имела уникальное имя в программе. В языке C++ появилась возможность иметь в программе несколько функций с одинаковыми именами, но, для правильного вызова функции необходимо, чтобы они отличались количеством и типом аргументов.

Это очень удобно, поскольку компилятор по типу аргументов и их количеству легко определит, какую функцию в данном случае необходимо вызвать. Операция замены функции при ее выполнении называется перегрузкой. Перегруженные функции не могут различаться по типу возвращаемого значения.

Можно привести пример создания функции **"print"**, которая может выводить на экран значения данных различных типов:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
// Перегруженные функции "print"
void print(int i)   {printf("%i",i);}
void print(float x) {printf("%8.3f",x);}
void print(char *s) {printf("%s",s);}
void vk()           {printf("\n");}

int main()
{  int a=-345;             // Целое число
   float pi=3.14159;       // Вещественное число
   char s[]="begin end"; // Символьная строка
   print(a);   vk();
   print(pi);  vk();
   print(s);   vk();
   getch();
}
```

При исполнении компилятор сам выбирает ту или иную функцию **"print"** для правильного вывода данных.

4. Сложные структуры данных

4.1. Последовательность

4.1.1. Свойства последовательности.

Последовательность – набор упорядоченно расположенных однотипных элементов.

Свойства последовательности:

1. Последовательность обладает конечной, но неограниченной длиной.

2. Элемент последовательности может быть получен только один раз.

3. Все элементы последовательности имеют один и тот же тип.

4. Конец последовательности может быть определен:

– длиной последовательности;

– временем доступа к элементам последовательности;

– индикатором окончания последовательности;

– некоторым условием, зависящим от значений элементов.

5. Последовательность может быть пустой, т.е. не содержать ни одного элемента.

6. Последовательность S обозначена так:

В связи с неограниченной длиной последовательности, т.е. возможности большого количества элементов в ней, эти элементы невозможно хранить в памяти компьютера из-за его ограниченного объема. Поэтому обработка элементов последовательности ведется по мере получения очередного элемента. При обработке следует учитывать, что индикатор последовательности не должен обрабатываться.

При обработке элементов последовательности могут быть использованы буферные элементы для временного хранения одного или нескольких элементов. Количество буферных элементов не должно зависеть от длины последовательности.

Последовательность определяется как структура с последовательным доступом, т.е. время доступа к элементу зависит от его местонахождения в последовательности (доступ к нужному элементу возможен только после чтения предыдущих элементов).

4.1.2. Обработка данных последовательности

В качестве примера рассмотрим следующую задачу:

Дана последовательность целых ненулевых чисел. Индикатор окончания – число ноль. Определить сумму элементов и количество отрицательных чисел в последовательности.

Для решения задачи используем накопитель и счетчик для получения суммы элементов и количества отрицательных чисел.

```
// Обработка последовательности - 1

#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#include <math.h>

void main()
 {
  int a; float sum=0; int ko=0; int f=0;
```

```
  cout<<"\nInput posledovatelnost (0)\n";
  cin>>a;

  while (a!=0)
    { sum+=a;
      if (a<0) ko++;
      f=1;
      cin>>a;
   }
  if (f==1)
     printf("Sum = %.3f, Kol.otr. = %i.\n",sum,ko);
  else printf("Posledovatelnost is empty !\n");
  getch();
 }
```

Отметим, что при использовании оператора **while** оператор ввода **cin>>a** приходится использовать 2 раза, первый – вне цикла для обработки пустой последовательности и второй – внутри цикла для ввода очередных элементов.

Для анализа ситуации, когда обрабатывается пустая последовательность, используется механизм фиксации события (флажок **int f=0**).

А теперь рассмотрим вторую форму организации цикла с использованием оператора **do...while.**

```
// Обработка последовательности - 2
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#include <math.h>
void main()
  {
  int a; float sum=0; int ko=0; int f=0;
  cout<<"\nВведите последовательность. (0)\n";
  do
   {cin>>a;
    if (a!=0)
     { sum+=a;
       if (a<0) ko++;
       f=1;
     }
   }
  while (a!=0);

  if(f==1)
    printf("Sum=%.3f,Kol.otr.=%i.\n",sum,ko);
  else
    printf("Последовательность пуста!\n");
```

```
getch();
}
```

В этом случае, чтобы исключить обработку индикатора окончания, необходимо дважды анализировать условие: **if(a<0)и while(a!=0).**

Еще один вариант программы связан с использованием оператора досрочного выхода из цикла - **break.** Другие способы определения окончания последовательности также должны быть реализованы с учетом того, что обрабатываться могут лишь элементы последовательности, а не служебные или иные виды данных.

4.2 Сложные структуры данных – массивы

4.2.1. Одномерные массивы

Массив – это группа элементов одного типа, расположенных друг за другом в памяти, и имеющих одно общее имя. Элемент часто называют компонентой массива.

Каждый элемент массива может быть обозначен с помощью указателя – индекса, который может быть вычислен.

Например, объявим массив данных:

```
int mas[4];
```

Т.е. объявлен массив элементов целого типа, состоящий из 4-х элементов. Размер массива в этом случае составляет 4 х 2 = 8 байт.

Индекс массива начинается с 0.

Как и обычные переменные, элементы массива могут быть инициализированы при объявлении:

```
int mas[4]={12, -7, 19, 5};
int mas[4]={6, 3};
```

Остальные элементы будут равны 0.

Индексы можно вычислять:

```
tmp = mas[i+1];
```

Если **i=2** тогда **tmp=8.**

Например, следующий фрагмент выводит значения массива на экран:

```
for (int i=0; i<4; i++)
  cout<<mas[i]<<"  ";
```

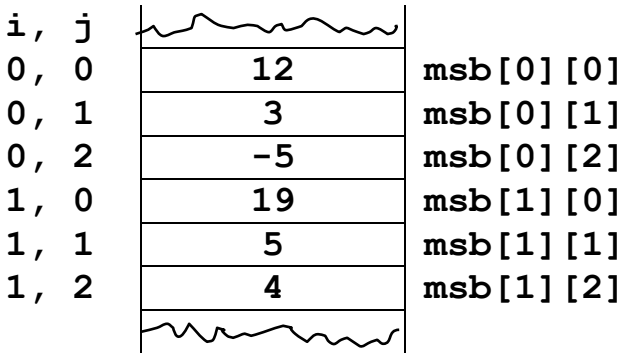
А вот этот фрагмент позволяет вводить данные в массив в диалоговом режиме:

```
for (int i=0; i<4; i++)
  {cout<<i<<" = "; cin>>mas[i];
cout<<"\n";
```

Массив не может быть пустым. В С++ можно использовать одно, двух, трех и более мерные массивы. Например, двухмерный массив 2 х 3 объявляется следующим образом:

```
int msb[2][3]={{12,3,-5},{19,5,4}};
```

Массив **msb** располагается в памяти он так:



Самый правый индекс в паре будет меняться чаще всего.

Чтобы вывести построчно двухмерный массив, можно использовать следующий программный фрагмент:

```
for (int i=0; i<2; i++)
  { for (int j=0; j<3; j++)
      cout<<msb[i][j]<<" = ";
    cout<<"\n";
  }
```

В качестве завершающего примера приведем реализацию программы, которая определяет количество "совершенных" чисел в одномерном массиве. Совершенным называют натуральное число, равное сумме всех своих собственных делителей (т. е. всех положительных делителей, отличных от самого́ числа).

```
// Task 51 Количество совершенных чисел
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#include <math.h>

int mas[12]={12,7,6,67,34,28,32,17,18,9};
int sover (int a);          // -- prototype

void main()                 // --- MAIN ---
  { int i,j;
    int k=0;
    for (i=0;i<9;i++)       // Обработка массива
      {if(sover(mas[i])==1)
         {k++; printf("%4i\n",mas[i]);};
      };
    printf("k=%2i.\n",k);
    getch();
```

```
  }

int sover (int a)  // 1 - если число a совершенное.
   { int Sum=0;
     for(int i=1;i<=a-1;i++)
      {if (a%i==0) Sum+=i;};
     if (Sum==a) return 1;
     return 0;
   }
```

4.2.2. Многомерные массивы

Наиболее часто используются двухмерные массивы, которые аналогичны таблицам, широко применяемых в финансовой сфере, экономике, производстве, государственных учреждениях и т.д.

При обработке двухмерных массивов возможны два подхода, назовем их условно негативный и позитивный.

Позитивный подход отличается тем, что в программе индексы элементов массива принимают только те значения, которые указывают на обрабатываемые элементы.

При негативном подходе индексы принимают все возможные значения, а выбор элементов для обработки определяется набором условий.

Например, для матрицы 5x5 обработать элементы, лежащие выше главной диагонали и выше обратной диагонали.

	j				m
i	0	1	2	3	4
0	#	X	X	X	#
1		#	X	#	
2			#		
3		#		#	
n 4	#				#

Пример реализации:

```
// Pozitiv and Negativ method for matrix
 # include<stdio.h>
 # include<conio.h>
 # include<iostream.h>
 # include<math.h>
 # include<dos.h>

 const n=5;
```

```
int M2[n][n]={{ 2, 4, 7,-2, 5},
              {-7, 1, 7, 9,-5},
              {-2, 3, 6, 8, 1},
              { 7, 1, 0,-4, 3},
              { 0, 2, 1, 8, 9}};

 int Sum_Pozitiv(int n)
   { int i,j, Sum=0;
     for (i=0; i<n/2; i++)
        for (j=i+1; j<n-i-1; j++)
          { Sum+=M2[i][j]; cout<<i<<" "<<j<<endl;}
     return Sum;
   }

 int Sum_Negativ(int n)
   { int i,j, Sum=0;
     for (i=0; i<n; i++)
       for (j=i+1; j<n; j++)
         if((i<j)&&(j<n-i-1)) Sum+=M2[i][j];
     return Sum;
   }
 void main()
 { clrscr();
   printf("Sum=%i\n",Sum_Pozitiv(n));
   printf("Sum=%i\n",Sum_Negativ(n));
   getch();
 }
```

4.3. Сложный тип данных – строки.

Строки предназначены для операций с символьными данными.

В C++ строка представляет собой некоторый массив элементов типа char, в конце которого помещается символ ‘\0’ (нуль-терминатор). Такой массив называется ASCII – строкой.

Инициализацию строки можно выполнить следующим образом:

```
char str[6] = {'B','E','G','I','N','\0'};
char str[6] = "BEGIN";
char str[]  = "BEGIN";
```

В памяти строка выглядит следующим образом:

str	B	E	G	I	N	\0
	0	1	2	3	4	5

Для работы со строками можно использовать следующие библиотечные функции:

gets (str) - ввод строки с клавиатуры;
puts (str) - вывод строки на экран;
printf("%s",str) –
вывод строки на экран, используя шаблон 's';

Рассмотрим пример: Определить, имеется ли в строке хотя бы одна латинская буква, а также напечатать количество цифр в строке. Для решения задачи организуем цикл по элементам строки от первого элемента до нуль-терминатора ('\0').

```
// String Latin && Cifra
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#include <math.h>
char st[36]="";                  // -- String

// Является ли simbol латинской буквой ?
int Latin(char simbol)
  { if (((simbol>='A')&&(simbol<='Z'))||
     ((simbol>='a')&&(simbol<='z')))
      return 1;
    return 0;
  }
// Является ли simbol цифрой ?
int Cifra(char simbol)
  { if ((simbol>='0')&&(simbol<='9')) return 1;
    return 0;
  }
void main()                      //-- Главная функция ---
 { int i; int f=0; int k=0;
   cout<<"Input Line = \n";
   gets(st);
   printf("%s\n",st);
   for (i=0; st[i]; i++)
    { if (Latin(st[i])) f=1;
      if (Cifra(st[i])) k++;
    }
   if (f) printf("Latin Yes!\n");
   else   printf("Latin No!\n");
   printf("Cifr = %i.\n",k);
   getch();
  }
```

Еще один пример показывает способ выборки слов из строки, отделенных друг от друга пробелами. Для этого мы напишем несколько полезных функций для работы со строками:

Len (str) - определяет длину строки;
Cat (str, ch) - конкатенация или склеивание;
Clr (str) - очистка строки.

Задание: распечатать в столбик все слова из строки st.

```
// Task 43 Word from String.
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#include <math.h>
char st[36]="";                 // -- String

int Len(char st[]){ for (int i=0; st[i]; i++); re-
turn (i);}
void Cat(char st[],char S)
        { st[Len(st)+1]='\0';st[Len(st)]=S;}
void Clr(char st[])
        { for (int i=0; st[i]; i++) st[i]='\0';}
void Pri(char st[])
        { for (int i=0; st[i]; i++)
             printf("%c",st[i]);
          printf("\n");
        }
// ------------ MAIN -------------------------
void main()
 { int i;  char wrd[36]="";
   cout<<"Input Line = \n";
   gets(st);
   Cat(st,' ');
   Pri(st);
   int CountWord=0;
   for (i=0; st[i]; i++)
    { if (st[i]!=' ') Cat(wrd,st[i]);
      else
        { if (Len(wrd)>0)
                { Pri(wrd); CountWord++; }
          Clr(wrd);
        }
    }
   printf("Count Word = %i.\n", CountWord);
   printf("Simbols = %i.\n", Len(st));
   getch();
  }
```

Для полноценной работы со строками необходимо использовать специализированные библиотеки функций, например **<string.h>**.

4.4. Сложный тип данных – структура

Очень часто в практике программирования необходимо работать с объектами, имеющими несколько параметров, причем все они могут быть разных типов. Для таких объектов в языке С++ предусмотрен тип данных под названием – структуры.

Сначала объявляется шаблон структуры:

```
struct <имя шаблона>
   { <тип> <имя переменной 1>;
     <тип> <имя переменной 2>;
      ......................;
   }
```

Например, опишем тип под названием «студент»:

```
struct student
{ char fam[20];  // фамилия студента
  char name[15]; // имя студента
  long phone;    // номер телефона
  int kurs;      // номер курса
}
```

Затем объявляется массив переменных типа **student**:

```
struct student MOAIS[24];
struct student IS[25];
```

В структуре возможен прямой доступ к полю структуры:

```
strcpy(IS[3].name,"Viktor");
MOAIS[17].phone=341698;
```

Инициализацию структуры можно выполнять одновременно с объявлением:

```
 struct student MOTS[17]=
{{"Petrova", "Elena", 411682,1}
 {"Demidov", "Fedor", 341672,4}
 {"Safronov","Nikita",421951,3}
 {"Dolgova", "Irina", 563074,1}};
```

При выполнении программы доступ к отдельным полям структуры выполняется селектором класса - "." (точка):

```
int Count=0;
for (int i=0;i<n;i++)
if (MOTS[i].kurs==3) Count++;
cout<<"Кол.студентов 3-го курса МОТС="<<Count<<endl;
```

Применение структур позволяет более четко поделить обрабатываемую информацию по объектам, сгруппировав их.

4.5. Сложный тип данных – файлы

4.5.1. Файл и файловая система

Главная проблема в системах обработки данных – это методы эффективного доступа к данным. Хранение больших объемов данных требует их структуризации как способа быстрого доступа к данным. Такие возможности предоставляют файловые системы. Они являются базовой основой систем обработки данных. Соответственно, инструментальные системы, в том числе и языки программирования, имеют соответствующие наборы операторов для работы с файлами. Файловая система – это совокупность файлов и набора специальных программ, регулирующих и обеспечивающих доступ к файлам.

Файл - это именованная область памяти на внешнем носителе для хранения информации. Содержимое файла представляет собой последовательность двоичных элементов. Эта последовательность при обработке может интерпретироваться как данные того или иного типа.

Замечательным свойством файлов является теоретическая неограниченность их размеров, ограничение только аппаратное. Обычно файлы размещаются на дисковых носителях. Их местоположение на дисках определяется особенностями файловых систем, например таблицами FAT в системах FAT16 или FAT32. Как правило, язык программирования C++ не делает различий между файлами на дисках и устройствами ввода-вывода, т.е. получение данных в виде файла может производиться и с других устройств, например с клавиатуры.

Внешние устройства компьютера обычно обозначены следующим образом:

1. CON – консоль (клавиатура и экран);
2. LPT1 – параллельный порт типа Centronix;
3. PRN – принтерный параллельный порт;
4. COM1, COM2 – последовательный порт типа RS-232;
5. NUL – фиктивное устройство;

В языке программирования C++ файловая система является двухуровневой, имеется уровень логических файлов и уровень физических файлов. Логический файл описывается как переменная файлового типа. Она применяется для связи с физическим файлом, находящемся на диске. И, таким образом, логический файл выполняет роль заместителя физического файла.

После того, как переменная файлового типа объявлена, она может быть связана с любым физическим файлом, независимо от его природы, т.е. на диске, на магнитной ленте, в локальной сети, и т.д.

Работа с данными, расположенными в файлах, имеют следующие особенности:

а) Файлы расположены, как правило, на внешних носителях.

б) Файлы не имеют фиксированной длины, т.е. в процессе работы размер файлов может измениться.

в) Файл может иметь нулевой размер, т.е. может быть пустым.

г) Файлы связаны с каналами передачи данных, поэтому перед работой они должны быть открыты, т.е. связаны с определенным каналом, а после работы закрыты.

д) По внутренней структуре файлы делятся на два вида: текстовые и бинарные.

4.5.2. Текстовые файлы

Текстовый файл предназначен для хранения текстов. Он представляет собой последовательность ASCII-символов, разделенная кодами конца строки и перевода каретки (**0xD, 0xA**).

Поэтому структура текстового файла выглядит следующим образом:

b	e	g	i	n	0x D	0x A	A	=	1	9	7	*	(	x	+	5	)	;	0x D	0x A	e	n	d	.
		^																						

Свойства текстового файла:

1. Введено понятие указателя на элемент файла, который будет обработан в следующий момент.
2. Доступ к элементам файла последовательный.
3. Длина файла определяется в FAT таблице. Ранее для определения конца файла использовался код **#26** или **0x1A**.
4. Текстовые файлы обычно имеют расширение: ***.txt, *.bat, *.cpp, *.asm**,***.html, *.xml** и т.д.
5. Текстовые файлы обычно создаются с помощью разнообразных текстовых редакторов.
6. Текстовые файлы - один из наиболее распространенных способов передачи информации, в том числе по сети Интернет.

Для работы с текстовыми файлами язык C++ имеет специальный набор функций, вот некоторые из них:

Функция **fopen()** – при успешном открытии файла возвращает указатель на структуру типа **FILE**. Функция имеет два параметра. Первый определяет физическое имя файла, т.е. имя файла, который находится на диске. Второй параметр задает способ доступа к данным файла и может принимать следующие значения:

r	Открыть файл на чтение
w	Открыть файл на запись. Создается новый пустой файл.
a	Открыть файл для добавления в конец файла.
r+	Открыть файл для чтения и записи. Файл должен существовать.
w+	Открыть файл для чтения и записи. Если файл существует, то он очищается
a+	Открыть файл для чтения и записи в конец файла. Создается новый файл.

К этому параметру может добавляться ключи, определяющие тип файла:

t	Открыть файл в текстовом режиме
b	Открыть файл в бинарном режиме

Таким образом, могут быть возможны следующие режимы доступа:
"rt", "wt", "rb", "wb", "w+b", "wb+", "a+t' и т.д.

Функция fclose(f) закрывает файл по окончании работы с ним.
Для операций чтения записи используется большое количество функций, например: **fprintf(), fscanf(), fgets()** и другие.

Рассмотрим несколько примеров работы с текстовыми файлами.
Например, фрагмент процедуры чтения текстового файла:

```
//-------------------------------------------------
//  Просмотр текстового файла (View).
//  Чтение текстового файла на экран.
//-------------------------------------------------
#include "mylib.cpp"

void main()
 { char fname[20];  FILE *pin;  char st[80];

   cout<<"Name of File = "; cin>>fname;

   if ((pin=fopen(fname,"rt"))==NULL)
      { printf("File not found"); getch(); return;}

   while (!feof(pin))     // Пока не конец файла
     {
       fgets(st,80,pin);  // Читаем из файла строку
       printf("%s\n",st);// Выводим строку на экран
     }

   fclose(pin);           // Закрываем файл
   getch();
 }

//-------------------------------------------------
// Простой текстовый редактор. Запись текста в файл.
//-------------------------------------------------
#include "mylib.cpp"
```

```
void main()
 { char fname[20];  FILE *out;  char st[80]; int n;

   cout<<"Name File = "; cin>>fname;// Ввод имени файла
   cout<<"Num Lines = "; cin>>n;    // Количество строк

   if ((out=fopen(fname,"wt"))==NULL) // Открываем файл
      { printf("File Error !"); getch(); return;}

   for (int i=0; i<n; i++)
     {
       gets(st);           // Вводим строку на клавиатуре
       fprintf(out,"%s\n",st);// Записываем строку в файл
     }

   fclose(out);                      // Закрываем файл
   printf("Ok!\n");
   getch();
 }

//-----------------------------------------------------
//Чтение числовых данных из текстового файла и вычисление
//их суммы.
//-----------------------------------------------------

#include "mylib.cpp"

void main()
 { char fname[20]; FILE *pin; int A; int Sum=0;

   cout<<"Name of File = "; cin>>fname;

   if ((pin=fopen(fname,"rt"))==NULL)
      { printf("File not found"); getch(); return;}

   while (!feof(pin))          // Пока не конец файла
     {
       fscanf(pin,"%i",&A);    // Читаем число из файла
       Sum+=A;                 // Суммируем
       printf("%i\n",A);       // Печатаем число на экран
     }
   printf("Sum=%i.\n",Sum);    // Печатаем сумму на экран
   fclose(pin);
   getch();
```

```
 }
//-----------------------------------------------------
//  Пример текстового файла с числовыми данными: "t1.txt"
//-----------------------------------------------------
12 45 67 34 -87
123 -456 19 -81 1234
-53 17 -62 101 169
```

4.5.3. Бинарные файлы

Бинарные файлы представляют собой последовательность двоичных кодов, копируемых из оперативной памяти без каких либо изменений. Такой способ передачи данных требует значительно меньше времени.

Так, например, в бинарный файл можно записывать массивы чисел, строк или структур в таком же виде, в котором они хранятся в памяти. Из бинарного файла данные также можно загрузить в память без изменений.

При этом совершенно невозможно по виду бинарного файла определить, какая информация в нем хранится.

Структура бинарного файла представляет собой последовательность записей одинаковой длины, соответствующей данному типу или структуре **(sizeof (int))** или (sizeof (struct)).

158	17	723	46	19	73	7	49
0	1	2	3	4	5	6	7

Для операций чтения отдельной записи бинарного файла предназначена функция, которая возвращает 1, если запись прочитана, иначе возвращает 0:

```
fread(<адрес буфера>,<размер записи>,
      <колич. блоков>,<указатель на файл>);
```

Для операций записи бинарного файла предназначена функция:

```
fwrite(<адрес буфера>,<размер записи>,
      <колич. блоков>,<указатель на открытый файл>);
```

Бинарный файл позволяет производить не только последовательный, но и произвольный доступ к записям файла. Для этого необходимо установить указатель на некоторую запись файла с помощью функции:

```
fseek(<указатель на открытый файл>,<величина смещения в байтах>,<параметр смещения>);
```

Пример: `fseek(f, sizeof(int)*(n-1), SEEK_SET);`

Файлы бинарного типа широко используются для хранения самых разнообразных данных в системах управления базами данных (СУБД). Они характеризуются высокой скоростью обмена, более высокой плотностью размещения данных, чем в текстовых файлах.

4.5.4. Простой пример работы с бинарным файлом

Для хранения данных о студентах используем бинарный файл.

Программа должна выполнять следующие функции:

1. Создавать бинарный файл.
2. Добавлять в него новые записи.
3. Просматривать записи.

Следует предусмотреть расширение функциональности данной задачи, например добавить поиск по фамилии, выборку данных по высоким или низким показателям балла успеваемости, удаление записей, сортировку записей и т.д. Поэтому в программу включим простейшее меню. Создадим его следующим образом. Выведем нумерованный список на экран и по запросу введем номер нужного пункта меню, по этому номеру будет выбираться соответствующая функция обработки нашей простой базы данных.

```
// Бинарный файл. База Данных - Студенты;
#include <stdio.h>;
#include <iostream.h>
#include <conio.h>

char fname[20];  FILE *inout;  char st[80]; int n;

struct student
  {  char fam[16];          // Фамилия студента
     int kurs;              // Курс обучения
     float ball;            // Балл успеваемости
  };
struct student person;
void CreateF();             // Создать файл
void AppendF();             // Добавить в файл
void ViewF();               // Просмотр файла

//-----------------------------------------
void main()
 { int poz=1;               // Позиция меню
   do {cout<<" 1. Create New File.\n";
       cout<<" 2. Append to File.\n";
       cout<<" 3. View File.\n";
       cout<<" 4. Exit Program.\n";
       cout<<"--> "; cin>>poz;

       if (poz==1) CreateF();
       if (poz==2) AppendF();
       if (poz==3) ViewF();
```

```
      } while (poz<4);
 }

void CreateF()        // Файл создается и закрывается
  { cout<<"Name of File = "; cin>>fname;
    if ((inout=fopen(fname,"wb"))==NULL)
       { printf("File Error!"); getch(); return;};
    fclose(inout); return;
  };

void AppendF()        // Файл открывается,
                      // добавляется запись
                      // файл закрывается

  { cout<<"Name of File = "; cin>>fname;
    if ((inout=fopen(fname,"ab"))==NULL)
       { printf("File Error!"); getch(); return; };
    cout<<"FAM  = "; cin>>person.fam;
    cout<<"Kurs = "; cin>>person.kurs;
    cout<<"Ball = "; cin>>person.ball;
    fwrite(&person, sizeof(person),1,inout);
    fclose(inout); return;
  };

void ViewF()          // Файл открывается,
                      // просматриваются все записи
                      // файл закрывается

  { cout<<"Name of File = "; cin>>fname;
    if ((inout=fopen(fname,"rb"))==NULL)
       { printf("File Error!"); getch(); return; };

    while (fread(&person,sizeof(person),1,inout))
       {  cout<<ftell(inout)<<"\n";
      printf("FAM:  %s\n",person.fam);
      printf("Kurs: %i\n",person.kurs);
      printf("Ball: %.2f\n",person.ball);
      printf("-------------------------\n");
       }
    fclose(inout); return;
  };
```

Литература

1. Шилд Г. Программирование на BORLAD C++ для профессионалов. Мн.1998
2. Керниган Б, Ричи Д. Язык программирования Си. М. 1992.
3. Основы алгоритмизации и программирования. Язык Си: учеб. пособие / Демидович Е.М.–СПб.:БХВ-Петербург, 2006.–440с.
4. Страуструп Б. Язык программирования C++.: Бином, Невский Диалект, 2004 г.–1104 стр.
5. Седжвик Р. Фундаментальные алгоритмы на C++.: Diasoft. М. 2004.–1136
6. Стенли Б. Липпман. C++ для начинающих: Пер. с англ. 2тт. - Москва: Унитех; Рязань: Гэлион, 1992,–345с.
7. М. Эллис, Б. Строуструп. Справочное руководство по языку C++ с комментариями: Пер. с англ. - Москва: Мир, 1992. 445с.
8. В.В. Подбельский. Язык C++: Учебное пособие. - Москва: Финансы и статистика, 1995. 560с.
9. У. Сэвитч. C++ в примерах: Пер. с англ. - Москва: ЭКОМ, 1997. 736с.

Приложение 1. Полезные программы

1.1. Реализация меню.

Меню, как способ выбора на некотором этапе вычислительного процесса, широко используется в программах.

Простейшее меню можно реализовать следующим образом:

```
// -----------------------------------------------------
// Простейшее меню - Easy Menu
// -----------------------------------------------------
#include "mylib.cpp"

void function_1() { cout<<"Input"; }
void function_2() { cout<<"Work";  }
void function_3() { cout<<"Print"; }

int main()
 { while(1)
   {
    clrscr();
    int pos;
    cout<<"  1.Input data"<<endl;
    cout<<"  2.Work  data"<<endl;
    cout<<"  3.Print data"<<endl;
    cout<<"  4.Exit      "<<endl;

    cout<<"  -->"; cin>>pos;

    switch (pos)
     {
      case 1: function_1(); break;
      case 2: function_2(); break;
      case 3: function_3(); break;
      case 4: return 0;
     }
    getch();
   }
}
//-----------------------------------------------------
```

Вначале выводится нумерованный список функций, затем вводится номер выбранной функции, которая исполняется. Процесс повторяется до ввода номера выхода.

Более сложная реализация меню показана ниже. В этом случае выбор позиции выполняется перемещением метки по меню, активизация происходит по нажатию клавиш Enter или пробела.

Метка по меню перемещается клавишами курсора вверх (Up) и вниз (Down).

Интересно, что эти клавиши при нажатии возвращают два кода, вначале 0, затем 80 (Up) или 72 (Down). Поэтому вначале анализируется, клавиша из какой группы нажата, алфавитной или управляющей.

```
// ------------------------------------------------------
//   Простое меню. Управление клавишами курсора.
// ------------------------------------------------------

#include "mylib.cpp"

const ms=4;                 // {- количество позиций меню -}
char L[ms][32]=             // {- наименования позиций ----}
     {"1.Input data",
      "2.Work  data",
      "3.Print data",
      "4.Exit      "};

void CF(int c,int f) { textcolor(c); textbackground(f);}

void OutMenu(int j)         // { вывод МЕНЮ на экран ------}
 { int i;                   // { j-местоположение курсора -}
   for(i=0;i<ms;i++)
    { if (i==j-1)CF(0,7);else CF(7,0);
      gotoxy(6,i+2); cprintf("%s",L[i]);}
 }

int main()                  // {--------- MAIN ------------}
{
 clrscr();
 int j=1; int ch; int b1=0; int J;
 OutMenu(1);
 while (!b1)
  {gotoxy(1,1);
   ch=getch();
   if(ch==0)
    { ch=getch();
      switch (ch)
       {case 80: if(j<ms)j++;
                 else j=1;
                 OutMenu(j); break;        //-Up--
        case 72: if (j>1)  j--;
```

```
                else j=ms;
                OutMenu(j); break;        //-Down-
     }
      }
    else
      { switch (ch)
      {case 13: b1=1; break;        // {---- Enter -----}
       case 32: b1=1; break;        // {---- Space -----}
     }
    }
  }
 switch (j)
  {case 1: gotoxy(10,12); printf("Procedure - 1"); break;
   case 2: gotoxy(20,14); printf("Procedure - 2"); break;
   case 3: gotoxy(30,16); printf("Procedure - 3"); break;
   case 4: gotoxy(40,18); printf("Procedure - 4"); break;
  }
 getch();
}
// ------------------------------------------------
```

1.2. Программа с использованием текстового режима

В качестве примера приводим небольшую программу анимации "Tank", в которой используется работа с цветом:

```
// ------------------------------------------------
//  Animation of text.  Танковая атака
// ------------------------------------------------
#include "mylib.cpp"
void Tank(int x, int y, int c)  // Отрисовка танка
                                   по координатам
 { textcolor(c);
   gotoxy(x,y);   cprintf("%s","_(#)==");
   gotoxy(x,y+1); cprintf("%s","<oooo>");
 }
void BaBax(int x, int y, int c) // Отрисовка снаряда
 { gotoxy(x,y);   textcolor(c); cprintf("%s","*");}

void Wzriv(int x, int y, int c) // Взрыв снаряда
 { textcolor(c);
   gotoxy(x,y-1); cprintf("%s","\\|/");
   gotoxy(x,y);   cprintf("%s"," 0");
   gotoxy(1,1);   delay(300); textcolor(0);
   gotoxy(x,y-1); cprintf("%s","\\|/");
   gotoxy(x,y);   cprintf("%s"," 0");
```

```
 }
//-------------------------------------------------
void main ()
 { int i,j; int CTank=2; int Tropa=10;
   clrscr();
   for (i=2;i<74;i++)// Основной цикл движения танка
    if (i%20==0)     // Через 20 шагов остановка и
                        выстрел!
     {
      Tank(i,Tropa,CTank);
      gotoxy(1,1);
      for (j=i+7;j<74;j++)  // Цикл движения снаряда
     { BaBax(j,Tropa,12);
       gotoxy(1,1); delay(50);
       BaBax(j,Tropa,0);
     }
       Wzriv(j+2,Tropa,14);    // Взрыв снаряда
       Tank(i,Tropa,0);
      }
     else
      {
       Tank(i,Tropa,CTank);    // Движение танка
       gotoxy(1,1); delay(200);
       Tank(i,Tropa,0);
      };
   Tank(i,Tropa,CTank);
   gotoxy(1,1);
   getch();
}
```

1.3. Собственная библиотека функций

Часто используемые функции полезно собрать в один файл, который оформляется в виде библиотеки. Эта библиотека подключается к программе директивой компилятора:

```
#include "mylib.cpp"
```

В качестве примера приведем библиотеку некоторых функций.

```
// -------------------------------------------------
// MyLib   Моя библиотека функций
// -------------------------------------------------
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#include <math.h>
#include <stdlib.h>
#include <dos.h>
```

```
// ------------------------------------------------
// Рекурсивная функция kcc возвращает количество цифр
// числа n. Пример вызова функции:
// printf("Kol Cifr v Chisle = %i.\n",kcc(123456789));
int kcc(long n) {if (n<10) return 1; return kcc(n/10)+1;}

// ------------------------------------------------------
// Рекурсивная функция scc возвращает сумму цифр числа n.
// Пример вызова функции:
// printf("Summa Cifr v Chisle = %i.\n",scc(123456789));
int scc(long n)
  {if (n<10) return 1; return scc(n/10)+n%10;}

//------------------------------------------------------
// Рекурсивная функция fact возвращает факториал числа n.
// Пример вызова функции:
// printf("Factorial = %li.\n",fact(10));
long fact(int n){ if (!n) return 1; return n*fact(n-1);}

// ------------------------------------------------------
// Рекурсивная функция p10to2 возвращает двоичное изо-
бра// жение десятичного числа n. Пример вызова функции:
// printf("Perevod 10 to 2 = %li.\n",p10to2(87));
long p10to2(long n)
  { if (!n) return 0; return 10*p10to2(n/2)+n%2;}

// ------------------------------------------------------
// Функция BitPrint печатает двоичный код числа n.

void BitPrint(int A)
  { int B=16384; int C;
    while (B)
     { if(A&B) C=1; else C=0; B>>=1; cout<<C;};
    cout<<endl;
  };

// ------------------------------------------------------
// Перегруженная функция print() для печати целых,
// вещественных и строковых данных.
void print(int i)   {printf("%i",i);}
void print(float x) {printf("%8.3f",x);}
void print(char *s) {printf("%s",s);}
```

Приложение 2. Практическое программирование

2.1. Тема 01: Линейные программы и программ с ветвлением

Задание. Разработать программу на языке программирования C++. В программе реализовать интерфейс диалога, математическое решение задачи и вывод результатов. Программа должна сообщать о невозможности решения задачи в случае некорректных данных. Вывод данных реализовать в форматированном виде.

Подобие. Даны два треугольника, один со сторонами A, B, C, другой со сторонами D, E, F. (Числа A,B,C,D,E,F целые). Определить, являются ли они подобными.

Круги. Даны два круга, один радиусом R1 и с центром в точке X1, Y1, другой радиусом R2 и с центром X2, Y2. Определить, находится ли один из них внутри другого.

Ромб и шар. Может ли шар радиуса R пройти сквозь ромбовидное отверстие с диагоналями P и Q?

Горячо! Даны три резистора R1, R2, R3, соединенные параллельно, к которым подключен источник напряжения U. Не сгорят ли они, если максимальная мощность рассеивания каждого резистора составляет W ватт?

Точка. Определить, находится ли точка M(x,y) внутри верхней части круга с центром в начале координат и радиусом R.

Пифагор. Треугольник задан длинами сторон A, B, C. Является ли он прямоугольным? Распечатать вершину прямого угла, если он прямоугольный.

Туристы. За A часов поездки на автомашине и P часов на поезде туристы проехали S км. Какова скорость поезда, если она на N км/час больше скорости автомашины?

Стройка. На стройке работает N бригад, причем на объекте Dom1 их в K раз больше, чем на объекте Dom2. Сколько их работает на каждом объекте? (K -целое число).

Автопарк. В автопарке находилось N автобусов, причем неисправных было в K раз меньше, чем исправных (K – целое число). Сколько автобусов вышло на линию?

Станки. Завод закупил N станков, причем токарных в K раз больше, чем сверлильных. Сколько токарных и сверлильных станков установили в цеха (K – целое число)?

Треугольник. Основание равнобедренного треугольника на K см больше его боковой стороны (K – целое число). Какова длина боковой стороны, если периметр треугольника равен P см.

Конфеты. За K кг конфет и P кг печенья заплатили Z рублей. Сколько стоит 1 кг печенья, если он дешевле 1 кг конфет на N рублей.

На Волге. Теплоход проходит за P1 часа по течению и R1 часа против течения S км. Он же за R2 ч против течения проходит на N км больше, чем за P2 ч по течению. Определить скорость теплохода по течению и против течения.

Пешеходы. Два пешехода вышли одновременно из двух городов (между ними S км), и встретились через T часов. Какова скорость пешеходов, если известно, что один из них прошел на N км больше.

Сладости. В пакете смешали 2 сорта конфет по цене C1 руб. и C2 руб. за кг, получили при этом S кг смеси по C3 руб. за кг. Сколько конфет каждого сорта в пакете?

На заводе. Всего первым станком за T1 часа и вторым за T2 часа сделано N деталей. Четвертая часть деталей, сделанных обоими станками за T3 часа, составила K шт. Сколько деталей делал каждый станок за час ?

Модернизация цеха позволила в апреле выпустить на K изделий больше, чем в марте, причем за эти месяцы произведено N изделий. Сколько их выпущено в каждом месяце?

На почте. Комплект из Kn1 конвертов и Ot1 открыток стоит C руб. Сколько стоит один конверт, если Kn2 конверта дешевле Ot2 открыток на K руб.?

Равенство. Определить, имеется ли среди чисел A, B, C, D хотя бы одна пара равных между собой чисел.

Максимальное число. Даны шесть целых чисел. Найти среди них максимальное нечетное число. Если такого числа нет, вывести сообщение.

Отрезки. Определить, можно ли из отрезков длиной A, B, C, D построить хотя бы один треугольник.

Кирпич. Определить, пройдет ли кирпич с ребрами A, B, C в прямоугольное отверстие размером D, E.

Карандаши. Какова стоимость покупки K карандашей по цене Pk копеек и T тетрадей по цене, в N раз дороже цены карандаша, а также общее количество купленных предметов.

GPS. Определить, находятся ли две точки A(x1, y1) и B(x2, y2) в одном квадранте или находятся в разных квадрантах. Напечатать, в каких квадрантах они находятся.

2.2. Тема 02: Операторы цикла.

Задание. Разработать программу на языке программирования C++.

В диалоговом режиме ввести число N (В диапазоне от 1 до 1000). Программа должна вывести искомые числа в виде нескольких колонок, выровненных по правому краю. Все числа от 1 до N натуральные.

1. Распечатать все числа от 1 до N, у которых остатки от деления на число Z не превышают числа M.
2. Распечатать квадраты только 3-значных чисел от 1 до N.
3. Распечатать все числа от 1 до N, не оканчивающиеся на цифру 3.
4. Распечатать корни всех чисел от 1 до N, не имеющих однозначных делителей (не равных числу).

5. Распечатать все числа от 1 до N, у которых младшая цифра кратна текущему числу.
6. Распечатать все числа от 1 до N, имеющие делители 3,4 и 7.
7. Распечатать квадраты всех нечетных чисел от 1 до N, кратных порядковому номеру текущего числа.
8. Распечатать все числа от 1 до N, у которых младшая цифра является делителем числа N.
9. Распечатать все числа от 1 до N, у которых нет двухзначных делителей (не равных числу).
10. Распечатать кубы всех нечетных чисел от 1 до N.
11. Распечатать все числа от 1 до N, у которых самый большой делитель (не равный числу) есть однозначное число.
12. Распечатать все числа от 1 до N, у которых первая и вторая цифра справа равны числу M.
13. Распечатать все числа от 1 до N, у которых есть хотя бы один двухзначный делитель (не равный числу).
14. Распечатать квадраты всех нечетных чисел от 1 до N, вторая цифра справа которых четна.
15. Распечатать все числа от 1 до N, у которых имеется делитель (не равный числу), кратный числу.
16. Распечатать все числа от 1 до N, у которых первая и вторая цифра справа не равны.

2.3. Тема 03: Целочисленная арифметика.

Задание. Разработать программу на языке программирования C++.

В диалоговом режиме задается целое длинное число A. Для решения каждой из трех задач обязательно использовать функции.

1. Определить, есть ли равные цифры в числе.
2. Определить, есть ли в числе рядом стоящие одинаковые цифры.
3. Определить, состоит ли число из возрастающих по значению цифр, начиная с младшей.
4. Определить, является ли первая и последняя цифра числа одинаковой.
5. Найти наибольшую нечетную цифру. Если ее нет, возвратить 0.
6. Определить номер максимальной по величине цифры.
7. Определить, сколько цифр числа делится на Z.
8. Определить, состоит ли число только из четных цифр.
9. Определить, является ли число состоящим только из одинаковых цифр.
10. Определить, состоит ли число из четных и нечетных цифр.
11. Найти количество наибольших цифр.
12. Найти количество простых цифр.

13.Найти номер наименьшей цифры.
14.Определить, является ли число палиндромом.
15.Определить, состоит ли правая половина числа из нечетных цифр.
16.Определить, являются ли первая и последняя цифра числа одинаковой четности.
17.Найти наименьшую четную цифру. Если ее нет, возвратить 0.
18.Определить, сколько цифр числа имеют меньших соседей справа.
19.Определить, является ли сумма цифр числа кратной количеству цифр.

2.4. Тема 04: Суммирование числовых рядов.

Задание. Разработать программу на языке программирования C++.

Вычислить сумму ряда, состоящего из N слагаемых, и вывести результат с M знаками после запятой. Предложите набор из 5 тестовых заданий для проверки правильности вычислений.

1. Вычислить: $y = 1 + \frac{x \ln a}{1!} + \frac{(x \ln a)^2}{2!} + \frac{(x \ln a)^3}{3!} + \ldots;$ (N слагаемых).

2. Вычислить: $y = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots;$ (N слагаемых).

3. Вычислить: $y = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots;$ (N слагаемых).

4. Вычислить: $y = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots;$ (N слагаемых).

5. Вычислить: $S = \frac{1}{(x)^2} + \frac{2}{(x)^3} + \frac{3}{(x)^4} + \frac{5}{(x)^5} + \frac{8}{(x)^6} + \ldots;$(N слагаемых).

6. Вычислить: $S = \frac{1!}{13} - \frac{2!}{15} + \frac{3!}{28} - \frac{4!}{43} + \ldots;$ (N слагаемых).

7. Вычислить: $S = \frac{1!}{(x+1)^3} - \frac{2!}{(x+2)^3} + \frac{3!}{(x+3)^3} \ldots;$ (N слагаемых.)

8. Вычислить: $S = \frac{1}{(2x-1)^2} + \frac{1}{(3x+2)^2} + \frac{1}{(4x-3)^2} + \ldots;$(N слагаемых).

9. Вычислить: $S = \frac{\sqrt{x}}{\ln(x)} + \frac{\sqrt{x^2}}{\ln(2 \cdot x)} + \frac{\sqrt{x^3}}{\ln(3 \cdot x)} + \ldots;$ (N слагаемых).

10.Вычислить: $S = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} - \ldots;$ (N слагаемых).

11. Вычислить: $y = x - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots;$ (N слагаемых).

12. Вычислить: $y = \frac{x}{a} - \frac{x^3}{\sqrt{a}} + \frac{x^5}{\sqrt{\sqrt{a}}} - \frac{x^7}{\sqrt{\sqrt{\sqrt{a}}}} + \ldots;$ (N слагаемых).

13. Вычислить: $y = 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots;$ (N слагаемых).

14. Вычислить: $y = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots;$ (N слагаемых).

15. Вычислить: $y = 1 - \frac{3}{2} + \frac{3 \cdot 5}{2 \cdot 4} \cdot x^2 - \frac{3 \cdot 5 \cdot 7}{2 \cdot 4 \cdot 6} \cdot x^3 + \ldots;$ (N слагаемых).

16. Вычислить: $y = -(1+x)^2 + \frac{(1+x)^4}{2} - \frac{(1+x)^6}{4} - \ldots;$(N слагаемых).

2.5. Тема 05: Обработка элементов последовательности.

Задание. Разработать программу на языке программирования C++.

Дана последовательность целых чисел, в диапазоне от –32000 до 32000, индикатор окончания – число 99999. Предложите набор из 2 тестовых заданий для проверки правильности вычислений.

1. Определить количество чисел последовательности, которые являются полным квадратом некоторого другого числа.
2. Определить количество чисел последовательности, у которых сумма цифр является простым числом.
3. Найти наибольшее количество подряд идущих нулей в последовательности.
4. Определить количество чисел последовательности, у которых цифры образуют возрастающую последовательность.
5. Найти число последовательности, у которого количество одинаковых цифр максимально.
6. Найти числа последовательности, у которого сумма делителей есть простое число.
7. Определить количество чисел последовательности, у которых сумма цифр является полным квадратом.
8. В последовательности имеются два простых числа. Определить количество тех четных чисел последовательности, которые находятся между простыми числами.
9. Определить, сколько в последовательности троек рядом стоящих чисел, которые могут выражать длины некоторого треугольника.
10. Определить, состоит ли последовательность из чередующихся простых и непростых чисел.
11. Определить количество чисел последовательности, значения которых кратны их номеру в этой последовательности.
12. Определить, сколько в последовательности пар взаимно-простых соседних чисел, т.е. не имеющих общих делителей, кроме 1. (Например: 16, 35).

13. Определить количество чисел-палиндромов последовательности, т.е. чисел, которые читаются одинаково и справа и слева (12321).
14. Определить количество чисел последовательности, состоящих только из разных цифр.
15. Определить номера автоморфных чисел, т.е. чисел, которые совпадают с младшими разрядами своих квадратов (25*25 = 625).
16. Определить номера совершенных чисел, т.е. чисел, совпадающих с суммой своих делителей, кроме себя (6 = 1+2+3).
17. Определить, сколько в последовательности пар простых соседних чисел-близнецов, т.е. двух простых чисел, отличающихся на 2. (17, 19).
18. Определить, есть ли в последовательности число, у которого чередуются четные и нечетные цифры (0 – четная цифра).
19. Определить количество наибольших нечетных чисел. Если их нет, тогда выдать сообщение.
20. Определить, каких чисел больше, положительных четных или отрицательных нечетных.
21. Определить, сколько в последовательности пар соседних простых чисел.
22. Определить, содержит ли каждый палиндром последовательности нечетное количество цифр.
23. Определить, имеет ли наименьшее число последовательности четных соседей слева и справа.
24. Найти число и его номер в последовательности, у которого количество делителей максимально.
25. Определить количество симметричных чисел, т.е. чисел с четным количеством цифр, у которых левая половина совпадает с правой половиной (245245).
26. Найти число последовательности, у которого сумма делителей максимальна

2.6. Тема 06: Одномерные массивы.

Задание. Разработать программу на языке программирования C++.

Дан массив из N целых чисел, где N<=16, каждое число в диапазоне от –32000 до 32000. Создать программу с обязательным использованием функций, реализующую задание согласно Вашему варианту.

Примечание:

- Массив перед обработкой и после обработки распечатать в виде строки чисел.
- Числа массива, подлежащие обработке, при распечатке исходного массива отметить отличающимся цветом.
- Массив может быть введен в диалоговом режиме, либо задан списком констант.

1. Удалить из массива числа, которые являются полным квадратом и имеют нечетную сумму цифр.
2. Вместо чисел, имеющих в составе цифру P, вставить сумму цифр этого числа.
3. Удалить из массива числа - полные квадраты, не имеющие цифры R.
4. Удалить из массива числа, имеющие непростую сумму цифр.
5. Удалить из массива автоморфные числа, т.е. числа, которые совпадают с младшими разрядами своих квадратов (25*25 = 625).
6. Вставить после каждого непростого числа его наибольший делитель.
7. Вставить после числа, являющегося полным квадратом, квадратный корень этого числа.
8. Вставить перед непростым числом сумму его делителей (все делители, кроме 1 и самого числа).
9. Удалить из массива числа – палиндромы, в которых есть хотя бы одна нечетная цифра.
10. Вместо двух рядом стоящих простых числа в массив вставить одно число – произведение удаленных простых чисел.
11. Удалить из массива все простые числа, состоящие из двух цифр.
12. Вставить после чисел, состоящих из одинаковых цифр, сумму их делителей.
13. Удалить из массива те числа, у которых старшая цифра больше младшей.
14. Удалить из массива числа-палиндромы с нечетным количеством цифр.
15. В начало массива перенести все простые числа.
16. Вставить между нечетными числами сумму их цифр.
17. Заменять каждую пару нечетных чисел суммой их цифр. После замены массив считается новым.
18. Удалить из массива числа, у которых количество делителей точно равно Z.
19. Определить, есть ли в массиве такие числа a, b и c, для которых выполняется равенство: a^:2 + b^2 = c^2. Эти тройки чисел напечатать.
20. Удалить из массива совершенные числа, т.е. числа, совпадающих с суммой своих делителей, кроме себя (6 = 1+2+3).
21. Удалить из массива числа – палиндромы, сумма цифр которых нечетна.
22. Удалить из массива все простые числа, состоящие только из нечетных цифр.

2.7. Тема 07: Обработка текстовых строк.

Задание. Разработать программу на языке программирования C++.

Дана строка символов **`St`** длиной не более 72, которая может состоять из цифр, малых латинских букв и знаков **`'+','-','*','/','.','(', ')'`** и пробела.

Строка задается либо в диалоговом режиме, либо в виде константы.

Перед обработкой строку следует вывести на экран.

1. Имеется ли в строке St изображение двухзначного числа?
   Например: в строке '**mite37sim**' имеется, а в '**wer673i**' - нет.
2. В строке St находится изображение целого числа, состоящее из цифр, например: '**abc12347654387de**'. Напечатать это число, делая пробелы между триадами цифр, отделяя таким образом тысячи, миллионы и т.д.
   Пример: **12 347 654 387.**
3. Определить, имеется ли в строке St хотя бы две цифры и один из знаков: **'+', '-', '*', '/';** Пример: **a7+b14=c8.**
4. Дана строка St. Напечатать сначала цифры, а потом малые латинские буквы, имеющиеся в этой строке.
   Пример: **AppLe 17-9=8 WhILe K19 Print(c485) End**
5. Определить, есть ли в строке St изображения нечетных чисел.
   Пример: **c234ap+a883ae-2784ymp**
6. Дана строка St. Определить, расположены ли латинские буквы по алфавиту. Пример: **abd+e56-klm16*pq21.**
7. Определить количество цифр и малых латинских букв в строке St, а также общую сумму всех цифр. Пример: **bA17d+BUf56-klM16*pQ21.**
8. Дана строка St, состоящая из латинских букв, цифр и знаков.
   Определить, что больше, цифр, знаков или букв.
   Пример: **a12bd+e/56-kl/m16*pq+21.**
9. Дана строка St, состоящая из латинских букв.
   Определить, чередуются ли в этой строке гласные и согласные буквы.
   Пример: **beginapenalanalog.**
10. Найти количество символов в самой длинной цепочке из одинаковых символов. Например, для **St='abcccdtteeeeekkksdgfhaa' K=5.**
11. Определить, сколько раз в строке St встречается
    двухбуквенное сочетание XY, где X и Y - заданные буквы.
    Пример: пусть **X=’n’,Y=’a’.** Тогда для **St=’begin-penal-analog’ K=2;**
12. Дана строка St, состоящая из латинских букв.
    Определить, сколько раз в строке встречается пара одинаковых чисел.
    Пример: **a12bd731+e/56-kl/m12*pq+56.**
13. Дана строка St, состоящая из латинских букв и цифр.
    Определить, имеются ли в строке изображения двоичных чисел, т.е. со-

стоящие только из цифр 0 и 1. Пример: **a1012bd+e/010011-kl/m106*pq+21.**

14. Дана строка St, состоящая из латинских букв, цифр и знаков. Определить, имеются ли в строке изображения чисел с плавающей запятой.
Пример: **a12bd73.1+e/56-k1/m1.2*pq+56.**
15. Дана строка St, состоящая из латинских букв, цифр и круглых скобок. Определить, является ли эта строка правильным скобочным выражением.
Пример: **((A+B)*(C -(D/F))).**
16. Дана строка St, состоящая из латинских букв, цифр и знаков. Определить наибольшую длину цепочки из пробелов.
Пример: **a12b (d731+56 kl/m12) (p + q*19).**
17. Дана строка St, состоящая из латинских букв, цифр и знаков. Определить количество пар рядом стоящих одинаковых букв.
Пример: **c234append+a83ee17-2784office.**
18. Дана строка St, состоящая из латинских букв, цифр и знаков. Найти количество букв, которые находятся внутри скобочных выражений.
Пример: **234*((a+12)*(b-d)+p2)/(27+84).**
19. Дана строка St, состоящая из латинских букв, цифр и знаков. Найти количество букв самого длинного слова, ограниченного пробелами или концами строки.
Пример: **a12b d731+561 kl/m12 p1+q*19.**
20. Дана строка St, состоящая из латинских букв, цифр и знаков. Определить, расположены ли слова в строке по алфавиту, определяя порядок по первой букве слова. Пример: **begin com double repeat time topic.**
21. Дана строка St, состоящая из латинских букв, цифр и знаков. Напечатать текст, состоящий из последних букв каждого слова строки St.
Пример: **begin com double repeat time topic.**
22. Дана строка St, состоящая из латинских букв, цифр и знаков. Найти количество промежутков между словами, имеющие более одного пробела.
Пример: **begin com double repeat time topic.**
23. Дана строка St, состоящая из латинских букв, цифр и знаков. Определить количество вхождений той буквы в строке St, которая встречается первой в этой же строке. Пример: **c234append+coffee 83+c17-2784office.**
24. Дана строка St, состоящая из латинских букв, цифр и знаков. Определить, сколько раз буква и цифра разделены знаком. Пример: **abc+6w7-vba*pascal-12.**
25. Дана строка St, состоящая из латинских букв, цифр и знаков. Определить количество слов-палиндромов в строке. Напечатать эти слова.
Пример: **begin mom doubuod repeat reper time topic.**

2.8. Тема 08: Двумерные массивы.

Задание: Разработать программу на языке программирования C++.

Дана квадратная матрица целых чисел размером N<=12. Заполнить матрицу случайными числами от 0 до 100.

Для четных вариантов использовать негативный метод.

Для нечетных – позитивный метод выборки чисел из матрицы.

Найти для вариантов 1,2,3,4,5,6,19,22 максимум, для 7,8,9,10,11,12,20,23 среднее значение, для 13,14,15,16,17,18,21,24 количество, тех чисел, которые расположены:

1, 9,17: Выше главной и обратной диагонали.

2,10,18: Выше главной и ниже обратной диагонали.

3,11,19: Ниже главной и выше обратной диагонали.

4,12,20: Ниже главной и ниже обратной диагонали.

5,13,21: Левая верхняя четверть матрицы.

6,14,22: Правая верхняя четверть матрицы.

7,15,23: Левая нижняя четверть матрицы.

8,16,24: Правая нижняя четверть матрицы.

Матрицу вывести в центр экрана, окружить одинарной рамкой, причем искомые числа раскрасить отличающимся цветом.

Результаты вычислений также вывести в двойной рамке.

2.9. Тема 09: Текстовые файлы.

Задание: Разработать программу на языке программирования C++.

Текстовый файл содержит изображения целых знаковых чисел.

После их чтения и обработки результаты также помещаются в выходной текстовый файл в виде символьных изображений чисел.

При открытии входного файла следует выполнить проверку существования файла с заданным именем. Содержимое входного и выходного файлов выводить на экран, причем предусмотреть отключение этой функции. При распечатке входного файла искомые числа выделить отличающимся цветом.

1. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить те из них, которые имеют только четные цифры. Пример: 2462 8204 44 и т.д.
   На экран вывести количество чисел во входном файле, количество чисел в выходном файле.
2. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить те из них, у которых старшая цифра больше младшей. Пример: 3462 804 41 и т.д.
   На экран вывести количество чисел во входном файле, количество чисел в выходном файле.
3. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить те из них, которые состоят из возрастающих цифр. Пример:

246 12478 45 и т.д.
На экран вывести количество чисел во входном файле, количество чисел в выходном файле.

4. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить те из них, которые имеют нечетные делители. Пример: 36 896 44 и т.д.
На экран вывести количество чисел во входном файле, количество чисел в выходном файле.
5. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить количество цифр каждого входного числа. Пример: для 12345 – 5, для 846 – 3 и т.д.
На экран вывести количество чисел во входном файле, количество чисел в выходном файле.
6. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить те из них, которые являются полными квадратами. Пример: 15129 625 49 и т.д.
На экран вывести количество чисел во входном файле, количество чисел в выходном файле.
7. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить средние арифметические цифр каждого числа. Пример: для 2462 – 3.50, для 16754 – 4.60, т.д.
На экран вывести количество чисел во входном файле, количество чисел в выходном файле.
8. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить те из них, значения которых кратны их порядковому номеру.
На экран вывести количество чисел во входном файле, количество чисел в выходном файле.
9. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить те из них, которые являются простыми числами. Пример: 17 47 и т.д.
На экран вывести количество чисел во входном файле, количество чисел в выходном файле.
10. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить те из них, которые имеют только разные цифры. Пример: 2467 8204 41 и т.д.
На экран вывести количество чисел во входном файле, количество чисел в выходном файле.
11. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить те из них, которые являются палиндромами. Пример: 124421 828 44 и т.д.
На экран вывести количество чисел во входном файле, количество чисел в выходном файле.

12. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить те из них, у которые имеется чередование четных и нечетных цифр. Пример: 2361 527 38 и т.д.
    На экран вывести количество чисел во входном файле, количество чисел в выходном файле.
13. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить те из них, которые имеют четные младшие цифры. Пример: 2462 8204 44 и т.д.
    На экран вывести количество чисел во входном файле, количество чисел в выходном файле.
14. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить только двухзначные числа. Пример: 34 84 41 и т.д.
    На экран вывести количество чисел во входном файле, количество чисел в выходном файле.
15. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить только трехзначные нечетные числа. Пример: 243 127 451 и т.д.
    На экран вывести количество чисел во входном файле, количество чисел в выходном файле.
16. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить те из них, которые делятся на число «M». Пример: 35 777 49 и т.д., если M=7.
    На экран вывести количество чисел во входном файле, количество чисел в выходном файле.
17. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить двухзначные нечетные числа. Пример: 13, 87, 91 и т.д.
    На экран вывести количество чисел во входном файле, количество чисел в выходном файле.
18. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить трехзначные отрицательные числа. Пример: -151 -625 -49 и т.д.
    На экран вывести количество чисел во входном файле, количество чисел в выходном файле.
19. Дан текстовый файл с изображениями целых чисел. В выходной файл квадратные корни трехзначных чисел. Пример: для 246 – 15.68, для 754 – 27.46, и т.д.
    На экран вывести количество чисел во входном файле, количество чисел в выходном файле.
20. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить обратные величины двухзначных чисел. Пример: для 26 – 0.038, для 87 – 0.011, и т.д.
    На экран вывести количество чисел во входном файле, количество чисел в выходном файле.

21. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить логарифмы трехзначных чисел. Пример: для 247 – 5.51, для 759 – 6.63, и т.д.
    На экран вывести количество чисел во входном файле, количество чисел в выходном файле.
22. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить квадраты двухзначных чисел. Пример: для 26 – 676, для 75 – 5625, и т.д.
    На экран вывести количество чисел во входном файле, количество чисел в выходном файле.
23. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить логарифмы трехзначных чисел. Пример: для 247 – 5.51, для 759 – 6.63, и т.д.
    На экран вывести количество чисел во входном файле, количество чисел в выходном файле.
24. Дан текстовый файл с изображениями целых чисел. В выходной файл поместить квадраты двухзначных чисел. Пример: для 26 – 676, для 75 – 5625, и т.д.
    На экран вывести количество чисел во входном файле, количество чисел в выходном файле.

Учебное издание

Угаров В.В.
Технология программирования. Часть 1.
Учебно-методическое пособие

Директор издательского центра
Редактирование и подготовка оригинал-макета

Подписано в печать
Формат
Усл. печ. л.
Уч.-изд. л.
Тираж 100 экз.
Заказ

Оригинал-макет подготовлен в Издательском центре
Ульяновского государственного университета

Отпечатано в издательском центре
Ульяновского государственного университета
432000, г.Ульяновск, ул. Л.Толстого, 42