Белорусский государственный университет

ПРАКТИКУМ
ПО КУРСУ «ПРОГРАММИРОВАНИЕ»
В ДВУХ ЧАСТЯХ
ЧАСТЬ 1. ЯЗЫК ПРОГРАММИРОВАНИЯ СИ

Учебное пособие

Для студентов университетов
специальностей
«Информатика»,
«Прикладная математика»,
«Компьютерная безопасность»,
«Актуарная математика» и
«Экономическая кибернетика

Минск 2007

Авторы: С. П. Бондаренко, А. П. Побегайло

Рецензенты: Л.Ф. Зимянин, Н.А. Разоренов

В практикум включены шесть лабораторных работ по языку программирования Си. Каждая из них содержит необходимые теоретические сведения, примеры и задания для самостоятельной работы.

Приведены также индивидуальные задания, которые охватывают весь учебный материал.

Лабораторный практикум предназначен для студентов ВУЗов, специальностей «Информатика», «Прикладная математика», «Компьютерная безопасность», «Актуарная математика» и «Экономическая кибернетика.

Предисловие

В практикуме изложены основы языка программирования Си, который был разработан в начале семидесятых годов американскими программистами Брайаном Керниганом и Деннисом Ритчи для кодирования операционной системы Unix, что позволило бы сделать эту операционную систему переносимой на различные аппаратные платформы. Поэтому в языке программирования Си так много средств, позволяющих программировать на «низком уровне». Можно сказать, что язык программирования Си разрабатывался как язык ассемблера высоко уровня. Эту особенность языка программирования Си нужно учитывать при его изучении.

Пособие включает шесть лабораторных работ по основным концепциям языка программирования Си. Каждая из них состоит из описания изучаемых средств языка программирования Си, набора обязательных и дополнительных задач для самостоятельного решения, а также заданий для индивидуального выполнения. В лабораторных работах рассмотрены следующие темы: работа с числовыми данными, программирование функций, обработка массивов и строк, работа со структурными типами данных и обработка файлов. Выполнение этих работ позволит освоить основные приемы программирования на языке Си.

Практикум предназначен для студентов всех специальностей факультета прикладной математики и информатики Белорусского государственного университета. Доступное изложение материала позволяет рекомендовать данное учебное пособие также для самостоятельного изучения основ программирования на языке Си.

Лабораторная работа 1

Тема. Работа с целыми и действительными числами. Форматированный ввод-вывод чисел. Управляющие инструкции. Использование стандартных математических функций.

1.1. Типы данных и переменные

Данными называется информация, хранящаяся в памяти компьютера. Единица этой информации называется значением данных или просто значением. Тип данных определяет множество значений и множество операций, допустимых над этими значениями. Данные делятся на переменные и константы. Переменными называются данные, значения которых могут быть изменены в процессе выполнения программы. Константами называются данные, значения которых изменить нельзя.

Значение переменной или константы хранится в области памяти компьютера, которая имеет свой адрес и длину. Причем длина этой области памяти зависит от типа данных переменной или константы. В языках программирования высокого уровня адрес значения переменной представляется символическим именем, которое называется именем переменной или просто переменной. Адрес значения константы может иметь символическое имя, а может и не иметь такого имени. В первом случае константа называется именованной константой, а во втором – литералом.

1.2. Числовые типы данных

К числовым типам данных относятся типы, которые предназначены для работы с числовыми данными. В языке программирования Си определены следующие числовые типы данных:

`int` – целочисленный тип данных;

`char` – символьный тип данных;

`float` – тип чисел с плавающей точкой;

`double` – тип чисел с плавающей точкой двойной точности.

Целочисленные переменные имеют тип `int` и объявляются следующим образом:

```
int x;
int i, j, k;
```

т. е. тип данных ставится перед именем переменной, которая используется для хранения этих данных. Во второй строке объявлено сразу несколько переменных целочисленного типа. В этом случае переменные разделяются запятыми.

В языке программирования Си для хранения символов используются переменные типа `char`. Объявляются переменные этого типа следующим образом:

```
char x;
```

Тип `char` также считается числовым типом данных. Над данными этого типа разрешены те же операции, что и над данными типа `int`.

В языке программирования Си для работы с действительными числами используются типы `float` или `double`. Переменные типа `double` имеют в два раза больший диапазон значений, чем переменные типа `float`. Переменные этого типа объявляются следующим образом:

```
float x;
double y, z;
```

При объявлении переменные могут инициализироваться. Для этого используется операция присваивания переменной некоторого значения. В следующих примерах показаны возможные варианты инициализации переменных.

```
int x = 2, y = 3;
int z = x + y;      /* z = 5 */
char x = 'a';
float x = 1.2f;     /* f обозначает float */
double y = 2.3;
double z = x + y;   /* z = 3.5 */
```

Здесь переменные `x` и `y` инициализируются числовыми литералами соответствующего типа.

Объявление именованной константы отличается от объявления переменной только тем, что начинается с ключевого слова `const`. Ниже приведены несколько примеров объявления числовых констант:

```
const int a = 2;
const char a = 'a';
const float a = 1.2f;
const double b = 2.3;
```

Как видим, именованные числовые константы инициализируются только числовыми литералами.

Отметим, что в программе все переменные и именованные константы должны иметь различные имена.

1.3. Арифметические операции над числами

Над целочисленными данными разрешается выполнять следующие арифметические операции:

```
+  сложение,
-  вычитание,
*  умножение,
/  деление,
%  деление по модулю.
```

В языке программирования Си символические обозначения этих операций называются операторами. В следующем примере показано различие между операторами `'/'` и `'%'`.

```
int x = 5, y = 2;
int z;
z = x / y;     /* z = 2; */
z = x % y;     /* z = 1 */
```

Над данными типа `float` и `double` разрешается выполнять следующие операции:

```
+  сложение,
-  вычитание,
*  умножение,
/  деление.
```

Операторы `'+'` и `'-'` являются как бинарными, так и унарными, т. е. могут применяться как к одному, так и к двум операндам. Кроме того, числовые переменные могут быть операндами следующих унарных операторов:

`++` инкремент, т. е. увеличение значения операнда на единицу,

`--` декремент, т. е. уменьшение значения операнда на единицу.

Эти операторы могут быть как префиксными, так и постфиксными, т. е. записываться как перед операндом, так и после него. Следующий пример демонстрирует различие между префиксными и постфиксными операциями инкремента и декремента:

```
int x = 1, y = 1, z;
z = ++x;     /* z = 2, x = 2 */
z = y++;     /* z = 1, y = 2 */
```

т. е. значение префиксной операции вычисляется перед вычислением значения выражения, а значение постфиксной операции вычисляется после вычисления значения выражения.

Если результат арифметической операции над числами необходимо поместить в первый операнд этой операции, то для этих целей используются следующие сокращенные обозначения арифметических операций:

+=, -=, *=, /=, %=. Например, увеличение значения переменной x на некоторое значение может быть записано следующим образом:

```
x += y;     /* эквивалентно x = x + y; */
```

1.4. Форматированный ввод и вывод чисел

Для форматированного ввода данных с консоли используется функция scanf. Для форматированного вывода данных на консоль используется функция printf. Прототипы этих функций описаны в заголовочном файле stdio.h. О заголовочных файлах и стандартных библиотеках будет рассказано позже. Пока же рассмотрим пример программы, в котором с консоли вводятся два целых числа, а затем на консоль выводится их сумма.

```
#include <stdio.h>

int main()
{
  int x, y;
  /* выводим текстовое сообщение */
  printf("Input two integers: ");
  /* вводим два целых числа */
  scanf("%d %d", &x, &y);
  /* выводим сумму этих чисел */
  printf("x + y = %d\n", x + y);
  return 0;
}
```

Немного поясним работу функций scanf и printf. Первым параметром этих функций является строка, которая будет форматироваться при вводе или выводе данных. Если в каком-то месте этой строки нужно вставить символьное представление целого числа, то на это место ставится спецификация ввода или вывода целого числа, которая в обоих случаях имеет вид %d. Следующие за строкой параметры функции scanf представляют собой адреса переменных, в которые нужно ввести целое число. Заметим, что операция взятия адреса переменной обозначается символом '&'. В свою очередь следующие за строкой параметры функции printf представляют собой просто целочисленные переменные или литералы, которые нужно вставить в строку для вывода на консоль.

В связи с этим примером также заметим, что управляющий символ '\n' в форматируемой строке функции printf используется для перевода курсора в первую позицию на следующей строке консоли.

Подобным образом выполняется и ввод-вывод чисел с плавающей точкой. Только для чисел типа `double` используются следующие спецификации: `%lf` – для ввода с консоли и `%f` – для вывода на консоль. Следующий пример демонстрирует форматированный ввод и вывод чисел с плавающей точкой двойной точности.

```
#include <stdio.h>

int main()
{
  double x = 0, y = 0;
  /* выводим текстовое сообщение */
  printf("Input two real numbers: ");
  /* вводим два действительных числа */
  scanf("%lf %lf", &x, &y);
  /* выводим сумму этих чисел */
  printf("x + y = %f\n", x + y);
  return 0;
}
```

Сделаем следующее замечание относительно этих программ. Каждая программа на языке программирования Си должна содержать функцию `main`, которая возвращает целочисленное значение. При запуске приложения операционной системой управление всегда передается функции `main`. Более подробно работа с функциями будет рассмотрена в одной из следующих лабораторных работ.

1.5. Логические операторы и операторы сравнения

В языке программирования Си определены следующие логические операторы:

`!` логическое отрицание,
`&&` логическая операция «и»,
`||` логическая операция «или».

Отметим, что операндами логических операций могут быть значения любого типа. При этом считается, что нулевому значению соответствует логическое значение «ложь», а любое другое значение соответствует логическому значению «истина».

Для сравнения числовых значений используются следующие операторы сравнения:

`==` равно,
`!=` не равно,
`>` больше,

`<` меньше,
`>=` больше или равно,
`<=` меньше или равно.

Если условие сравнения выполняется, то результатом оператора сравнения является целое число `1`, в противном случае – число `0`. Например, можно написать следующее выражение для сравнения числовых данных:

```
(a > b) && (b != c) || (c == d)
```

1.6. Условные инструкции if и if-else

Для управления ходом выполнения программы в языке программирования Си используются управляющие инструкции, которые рассмотрены в этом и следующих трех параграфах.

Для проверки истинности некоторого условия используется управляющая инструкция

```
if (выражение)
  инструкция
```

которая работает следующим образом. Если значение выражения истинно, то выполняется инструкция, следующая за выражением. Иначе, ничего не выполняется. Например, следующая инструкция увеличивает значение переменной `a` на единицу только в том случае, если это значение меньше нуля.

```
if (a < 0)
  ++a;
```

Если необходимо выбрать одно из действий в зависимости от значения логического выражения, то для этой цели используется управляющая инструкция

```
if (выражение)
  инструкция
else
  инструкция
```

В следующем примере значение переменной `a` увеличивается на единицу, если оно меньше нуля, и уменьшается на единицу в противном случае.

```
if (a < 0)
  ++a;
else
  --a;
```

1.7. Инструкции цикла while и do-while

Для выполнения инструкции в цикле до тех пор, пока значение некоторого логического выражения остается истинным, используются управляющие инструкции

```
while (выражение)
  инструкция
```

и

```
do
  инструкция
while (выражение)
```

В этих случаях логическое выражение после ключевого слова `while` также называется условием продолжения цикла. В первом случае условие продолжения цикла проверяется перед выполнением инструкции, а во втором случае – после выполнения инструкции. Использование инструкций цикла `while` и `do-while` показано в следующих примерах:

```
while (a < 0)
  ++a;
```

или

```
do
  ++a;
while (a < 0);
```

Для принудительного выхода из циклов `while` и do-while используется инструкция `break`. Например:

```
while (a < 0)
{
  ++a;
  /* если a > b, то выход из цикла while */
  if (a > b)
    break;
}
```

Для перехода на исполнение следующего цикла, не ожидая завершения исполнения текущего цикла, используется инструкция `continue`. Например:

```
while (a < 0)
{
  ++a;
  /* если a > b, то переход на начало цикла while */
  if (a > b)
    continue;
  --b;
}
```

Как видно из этих определений, возможно вложение управляющих инструкций друг в друга.

1.8. Инструкция цикла for

Для выполнения циклических действий с автоматическим изменением значений некоторых переменных удобно использовать управляющую инструкцию

```
for (выражение_1; выражение_2; выражение_3)
  инструкция;
```

где выражения имеют следующее назначение:

«выражение_1» описывает инициализацию цикла и вычисляется только один раз перед началом цикла;

«выражение_2» описывает условие, которое проверяется каждый раз перед выполнением инструкции; если значение этого выражения истинно, то инструкция выполняется, в противном случае цикл заканчивается;

«выражение_3» вычисляется после каждой итерации цикла.

Например, программа

```
#include <stdio.h>

int main()
{
  for (int i = 0; i < 3; ++i)
    printf("i = %d\n", i);
  printf("\n");
  return 0;
}
```

выводит на консоль последовательные значения целочисленной переменной.

Для принудительного выхода или продолжения цикла `for` используются инструкции `break` и `continue` соответственно.

1.9. Инструкция выбора switch

Инструкция выбора `switch` передает управление другой инструкции в зависимости от значения некоторого выражения. В общем случае инструкция `switch` имеет вид

```
switch (выражение)
{
case  константа_1:  инструкции
case  константа_2:  инструкции
...
default:  инструкции
}
```

и работает следующим образом. Сначала вычисляется значение выражения. Затем это значение сравнивается с константами. Управление передается на метку, для которой это сравнение дает значение «истина». Если значение выражения не совпадает ни с одной из констант, то управление передается инструкции с меткой `default`. А если этой метки нет, то происходит выход из блока `switch`. Для принудительного выхода из блока `switch` используется инструкция `break`.

В следующей программе приведен пример использования инструкции `switch`.

```
#include <stdio.h>

int main()
{
  char c;
  printf("Input any char 'a' or 'b': ");
  scanf("%c", &c);
  switch (c)
  {
    case 'a':
      printf("You input 'a'.\n");
      break;
    case 'b':
      printf("You input 'b'.\n");
      break;
    default:
      printf("You input a different letter.\n");
  }
  return 0;
}
```

Сделаем некоторые важные замечания относительно использования инструкции `switch`. Во-первых, выражение и константы в инструкции `switch` должны иметь целочисленный тип. Во-вторых, никакие две константы не могут иметь одинаковое значение.

1.10. Блоки

Часто при программировании управляющих инструкций необходимо, чтобы при выполнении некоторого условия выполнялись сразу несколько других инструкций. Для этой цели инструкции объединяются в блок инструкций или просто блок. Начало и конец блока отмечаются соответственно символами `'{'` и `'}'`. Особо отметим, что после окончания блока ставить точку с запятой не нужно. Например, блок внутри управляющей инструкции `if` может выглядеть следующим образом:

```
if (a < 0)
{
  --a;
  c = a + b;
}
```

1.11. Стандартная библиотека математических функций

В языке программирования Си имеется стандартная библиотека, которая содержит основные функции, используемые при математических расчетах. Эти функции могут быть разбиты на следующие группы:

тригонометрические функции

`double acos(double x)`	возвращает арккосинус числа `x`
`double asin(double x)`	возвращает арксинус числа `x`
`double atan(double x)`	возвращает арктангенс числа `x`
`double atan2(double y, double x)`	возвращает арктангенс числа y/x
`double cos(double x)`	возвращает косинус числа `x`
`double sin(double x)`	возвращает синус числа `x`
`double tan(double x)`	возвращает тангенс числа `x`

экспоненциальные и логарифмические функции

`double cosh(double x)`	возвращает косинус гиперболический числа `x`
`double exp(double x)`	возвращает экспоненту числа `x`
`double frexp(double x, int *a)`	находит такую мантиссу `m` и экспоненту `e` числа `x`, что `x=m*2^e` и $0.5 \le m < 1.0$; потом возвращает `m`, а по адресу `a` записывает `e`
`double ldexp(double x, int e)`	возвращает число `x *2^e`
`double log(double x)`	возвращает натуральный логарифм числа `x`
`double log10(x)`	возвращает логарифм числа `x` по основанию `10`
`double sinh(double x)`	возвращает синус гиперболический числа `x`
`double tanh(double x)`	возвращает тангенс гиперболический числа `x`

степень и корень

double pow(double x, double y)	возвращает *x* в степени *y*
double sqrt(double x)	возвращает квадратный корень из *x*

остальные функции

`double modf(double x, int *a)`	возвращает дробную часть числа `x`, а по адресу `a` записывает целую часть этого числа
`double ceil(double x)`	возвращает наименьшее целое число, которое больше `x`
`double fabs(double x)`	возвращает абсолютное значение числа `x`
`double floor(double x)`	возвращает наибольшее целое число, которое меньше `x`
`double fmod(double x, double y)`	возвращает остаток от деления `x` на `y`

Для использования математических функций в программу необходимо включить заголовочный файл `math.h`. Например, следующая программа сначала вводит число `x`, затем вычисляет синус этого числа и выводит его на консоль.

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

int main()
{
  double x = 0;
  printf("Input a real number: ");
  scanf("%lf", &x);
  printf("sin x = %f\n", sin(x));
  return 0;
}
```

1.12. Задачи для самостоятельного решения

Написать программы для решения следующих задач.

1. Найти НОД и НОК двух заданных натуральных чисел, используя алгоритм Евклида.
2. Определить, является ли заданное натуральное число простым.
3. Вывести на консоль все числа Армстронга, которые находятся в заданном интервале натуральных чисел. Напомним, что числом Армстронга называется натуральное число, которое равно сумме своих цифр,

каждая из которых возведена в степень n, где n – количество цифр в записи числа. Например, $153 = 1^3+5^3+3^3$.

4. Найти действительные корни квадратного уравнения.
5. Найти сумму ряда $\sum_{k=1}^{\infty}\frac{(-1)^k}{(2k-1)^2}$ с точностью, которая вводится с консоли и обозначает количество верных цифр после десятичной точки.

1.13. Дополнительные задачи

Написать программы для решения следующих задач.

1. Разложить заданное натуральное число на простые множители.
2. Получить, если это возможно, для заданного натурального числа x палиндром, используя следующую итерационную формулу: $y_{n+1} = y_n + z_n$, где z_n - натуральное число, запись которого совпадает с обратной записью натурального числа y_n, и $y_0 = x$.
3. Найти все точки с целочисленными координатами, которые находятся внутри круга радиуса R и с центром в точке с координатами x, y, где R, x, y – действительные числа, которые вводятся с консоли.
4. Вычислить значение функции sin$x$ с точностью ε, используя разложение в ряд Тейлора $\sin x = \sum_{n=0}^{\infty}(-1)^n\frac{x^{2n+1}}{(2n+1)!}$. Действительное число x и точность ε вводятся с консоли.
5. Вычислить квадратный корень из положительного действительного числа x с точностью ε, используя следующую итерационную формулу: $y_{n+1} = \frac{1}{2}(y_n + \frac{x}{y_n})$, $y_0 = 2^k$. Здесь k – целая часть от числа $\frac{m}{2}$, где m такое целое число, что $x = 2^m q$ и число q удовлетворяет неравенству $\frac{1}{2} \le q < 1$. Действительное число x и точность ε вводятся с консоли.

1.14. Задачи для индивидуальной работы

1. ***Нули в факториале.*** Составить программу, которая для заданного натурального числа A определяет количество последних нулей в записи числа A!, не вычисляя факториал.
2. ***Римская система счисления.*** Составить программу, которая представляет заданное натуральное число в римской системе счисления.
3. ***Объединение.*** Заданы два натуральных числа, цифры которых расположены по возрастанию. Написать программу, которая получает новое число, цифры которого также расположены по возрастанию и включены в это число по принципу:

- полного объединения цифр двух чисел;
- пересечения цифр двух чисел;
- максимального вхождения в каждое из чисел;
- вычеркивания из записи второго числа цифр, входящих в первое число.

4. ***Палиндром.*** Составить программу, которая для заданного натурального числа, не являющегося палиндромом, получает из этого числа, если это возможно:

- максимальный (минимальный) по величине палиндром путем перестановки его цифр;
- палиндром, выбросив минимальное количество ненужных цифр из записи числа и сохранив порядок следования оставшихся цифр;
- палиндром путем удаления минимального количества цифр в записи числа и перестановкой оставшихся;
- палиндром, дописывая, слева или справа, к записи числа минимальное количество цифр.

5. ***Цифра в ряду.*** Составить программу, определяющую *к*-ую цифру в натуральном числе, которое получено:

- последовательной записью натуральных чисел: 1 2 3 4 …;
- последовательной записью квадратов натуральных чисел: 1 4 9 16 25 36 …;
- последовательной записью степеней двойки, начиная с нулевой степени: 1 2 4 8 16 32 …;
- последовательной записью чисел Фибоначчи: 1 2 3 5 8 13 …;
- последовательной записью двоичных чисел, каждое последующее из которых получено инвертированием предшествующего двоичного числа: 010 101 101010 …; первое двоичное число выбирается произвольно.

6. ***Минимальное вычеркивание.*** Составить программу, которая в записи заданного натурального числа вычеркивает минимальное количество цифр так, чтобы оставшиеся цифры:

- являлись отрезком натурального ряда;
- образовали арифметическую (геометрическую) прогрессию;
- образовали возрастающую последовательность степеней двойки;
- являлись строго возрастающей последовательностью.

7. ***Дроби.*** Составить программу, которая выводит в порядке возрастания все обыкновенные несократимые дроби, со знаменателем, не превышающим заданного натурального числа. Сортировку полученных дробей не использовать.

8. ***Число π.*** Вычислить число π с заданной точностью ε, используя предложенные формулы:

$$\pi = 4(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + ...)$$

$$\pi = 3 + 4(\frac{1}{2*3*4} - \frac{1}{4*5*6} + \frac{1}{6*7*8} - ...)$$

$$\pi = \sqrt{6(1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + ...)}$$

9. ***Извлечение корня.*** Для заданных действительных чисел a и p вычислить с заданной точностью ε величину $x = \sqrt[p]{a}$, используя следующие рекуррентные формулы:

$$а) x_{n+1} = \frac{x_n}{p^2}((p^2 - 1) + \frac{1}{2}(p+1)\frac{a}{x_n^p} - \frac{1}{2}(p-1)\frac{x_n^p}{a}),\ x_0 = a;$$

$$б) x_{n+1} = \frac{1}{p}((p-1)x_n + \frac{a}{x_n^{p-1}}), x_0 = a;$$

10. ***Кубический корень.*** Для заданного действительного числа a вычислить с заданной точность ε величину $x = \sqrt[3]{a}$, используя рекуррентное соотношение Ньютона

$$x_{n+1} = \frac{1}{3}(x_n + 2\sqrt{\frac{a}{x_0}}), x_0 = a;$$

Лабораторная работа 2

Тема. Указатели. Статические массивы. Динамическое распределение памяти. Динамические массивы.

2.1. Указатели

Указатель – это переменная, которая содержит адрес другой переменной. Для объявления указателя нужно перед именем переменной поставить символ `'*'`. Например:

```
char *pch;   /* указатель на символьную переменную */
int  *pn;    /* указатель на переменную типа int */
double *pd;  /* указатель на переменную типа double */
```

Один указатель часто объявляется в следующем стиле:

```
int* pn;
```

Однако этот стиль не подходит для объявления нескольких указателей в одной инструкции, так как перед каждым из них нужно ставить символ `'*'`. Например:

```
int *pn1, *pn2;  /* два указателя */
```

Но в следующем примере

```
int *pn, n;      /* указатель и переменная */
```

только переменная `pn` является указателем.

При объявлении указателя желательно сразу же его проинициализировать. Это значительно упрощает отладку программы. Если начальное значение указателя неизвестно, то ему нужно присвоить значение NULL, которое будет явно указывать, что указатель пока не используется. Например:

```
int* pn = NULL;
```

Для получения адреса переменной используется оператор `'&'`, который так и называется – оператор получения адреса переменной. В следующем примере указателю присваивается адрес переменной:

```
int n;
int* pn = &n;  /* pn указывает на переменную n */
```

Для получения значения переменной, на которую указывает указатель, используется оператор `'*'`, который называется оператором обращения по адресу. Например:

```
int n = 2;
int* pn = &n;  /* pn указывает на переменную n */
```

```
int m = *pn;  /* m = 2 */
```

2.2. Массивы

Массив – это конечная последовательность однотипных данных. Каждый член этой последовательности называется элементом массива. Доступ к элементам массива производится по их номеру в последовательности, который в этом случае называется индексом. Количество элементов в массиве называется размерностью массива. Причем важно заметить, что если размерность массива равна `n`, то индекс этого массива изменяется от `0` до `n-1`. Например, в языке программирования Си массив из десяти целых чисел объявляется следующим образом:

```
int a[10];
```

Имя массива является константным указателем на его первый элемент, т. е. указателем, значение которого нельзя изменить. При определении массива его можно проинициализировать. Например,

```
int a[3] = {0, 1, 2};
```

Доступ к элементам массива выполняется при помощи оператора индексирования `'[]'`, результатом выполнения которого является значение элемента массива с заданным индексом. Например,

```
int n;
int a[3] = {10, 20, 30};
n = a[0];    /* n = 10 */
a[1] = 40;
```

В языке программирования Си можно определять и многомерные массивы. В этом случае граница каждого измерения массива указывается в отдельных квадратных скобках. Например,

```
int  a[2][2] = {{1, 2}, {3, 4}};
int  n = a[0][0];    /* n = 1 */
```

В заключение этого параграфа приведем программу, которая вводит, а затем выводит на консоль элементы массива из трех элементов.

```
#include <stdio.h>

int main()
{
  int  a[3];
  int i;
  printf("Input three integers: ");
  for (i = 0; i < 3; ++i)
    scanf("%d", &a[i]);
```

```
  printf("The array:\n");
  for (i = 0; i < 3; ++i)
    printf("a[%d] = %d ", i, a[i]);
  printf("\n");
  return 0;
}
```

2.3. Арифметические действия с указателями

Над указателями одного типа можно выполнять арифметическую операцию вычитания `'-'`. В результате выполнения этой операции получаем целое число, которое указывает, на сколько один адрес памяти смещен от другого в единицах, равных длине типа, на который указывают указатели. Например:

```
int a[5];
int n, m;

n = &a[5] - &a[3];  /* n = 2 */
m = &a[3] - &a[5];  /* m = -2 */
```

К указателям можно применять операторы `'++'` и `'--'`. В этом случае значение указателя соответственно увеличится или уменьшится на длину типа, на который указывает указатель. Например:

```
int* a;
++a;  /* a = &a[1] */
```

К указателю можно прибавить или вычесть целое число. В этом случае значение указателя соответственно увеличится или уменьшится на это число, умноженное на длину типа данных, на который указывает указатель. Например:

```
int a[5];
int *p = a+2;  /* p = &a[2] */
```

2.4. Динамическое распределение памяти

Под динамической памятью понимают память, которая выделяется программе во время ее работы. Динамическое распределение памяти включает выделение программе памяти по ее запросу и последующее освобождение программой этой памяти. Для выделения памяти в языке программирования Си используются следующие функции:

```
void* malloc(size_t size);
void* calloc(size_t n, size_t size);
void* realloc(void* ptr, size_t size);
```

Прототипы этих функций находятся в заголовочном файле `stdlib.h`. Параметры этих функций содержат тип `size_t`, который

является переопределением типа unsigned int. Тип void* является родовым указателем, т. е. указателем на «пустую» память. Впоследствии этот указатель приводится к указателю на нужный тип. Опишем кратко работу этих функций.

Функция malloc выделяет программе память размером в size байтов.

Функция calloc выделяет программе память размером в (n*size) байтов, при этом выделенная память обнуляется.

Функция realloc перераспределяет память, на которую указывает ptr, до размера в size байтов. Т. е. программе распределяется новый блок памяти в size байтов. При этом size байтов информации из старого блока копируются в новый блок. После этого старый блок памяти освобождается.

Все эти функции в случае успеха возвращают указатель на выделенную память, в случае неудачи – NULL.

Для освобождения памяти используется функция

```
void free(void* ptr);
```

прототип которой также находится в заголовочном файле stdlib.h. Параметр ptr этой функции является указателем на освобождаемую память.

Сделаем важное замечание. Динамическую память нужно освобождать после ее использования, иначе остаются неиспользованные блоки памяти, которые называются «мусором». Также заметим, что динамическое выделение памяти используется только в том случае, если заранее неизвестно, какой размер памяти нужно отвести под данные. В следующем примере показано, как динамически выделить память под целое число.

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

int main()
{
  char c;
  int* pn = NULL;
  printf("Press 'y' to input integer: ");
  scanf("%c", &c);
  if (c == 'y')
  {
    /* выделение памяти под целое число */
    pn = (int*)malloc(sizeof(int));
    /* если ошибка, то выходим из программы */
```

```
    if (!pn)
    {
      printf("Error: there is no memory.\n");
      return 0;
    }
    printf("Input integer: ");
    scanf("%d", pn);
    printf("You input integer: %d\n", *pn);
    /* освобождаем память */
    free(pn);
  }
  else
    printf("You don't want to input integer.\n");
  return 0;
}
```

2.5. Динамические массивы

Для того чтобы динамически создать одномерный массив, нужно, во-первых, объявить в программе указатель на тип, соответствующий типу элементов этого массива, а во-вторых, выделить память под массив, используя одну из функций `malloc` или `calloc`. Ниже приведена программа, которая динамически создает одномерный целочисленный массив.

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

int main()
{
  /* указатель на массив, размерность массива */
  int *a, n;
  printf("Input a size of an array: ");
  scanf("%d", &n);
  /* выделяем память под массив */
  a = (int*)malloc(n * sizeof(int));
  /* если ошибка, то выходим из программы */
  if (!a)
  {
    printf("Error: there is no memory.\n");
    return 0;
  }
  /* вводим элементы массива */
  printf("Input elements of the array: ");
  for (int i = 0; i < n; ++i)
    scanf("%d", &a[i]);
```

```
  /* можно ввести элементы массива и так */
  printf("Input elements of the array: ");
  for (i = 0; i < n; ++i)
    scanf("%d", a + i);

  /* что-то делаем с массивом */

  printf("You input the array: ");
  for (i = 0; i < n; ++i)
    printf("%d ", a[i]);
  printf("\n");

  /* освобождаем выделенную память */
  free(a);
  return 0;
}
```

Память под многомерные динамические массивы выделяется по строкам, начиная с первого индекса. Это делается для того, чтобы обеспечить применение операции индексирования `'[]'` столько раз, какова размерность многомерного массива. В этом случае тип динамического массива объявляется как указатель, который содержит оператор обращения по адресу `'*'` столько раз, какова размерность массива. Например, указатели на двухмерный и трехмерный целочисленные массивы могут быть объявлены следующим образом:

```
int **a;   /* указатель на двумерный массив */
int ***b;  /* указатель на трехмерный массив */
```

Приведем примеры динамического создания и уничтожения двухмерного массива.

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

int main()
{
  /* указатель на массив, размерности массива */
  int **a, n, m;
  /* индексы элементов массива */
  int i, j;
  printf("Input two sizes of an array: ");
  scanf("%d%d", &n, &m);

   /* выделяем память для указателей на строки */
  a =  (int**)malloc(n * sizeof(int*));
  /* если ошибка, то выходим из программы */
  if (!a)
  {
    printf("Error: there is no memory.");
```

```
    return 0;
  }

  /* выделяем память для строк */
  for (i = 0; i < n; ++i)
  {
    a[i] = (int*)malloc(m * sizeof(int));
    /* если ошибка, то освобождаем память*/
    /* и выходим из программы */
    if (!a[i])
    {
      for (j = 0; j < i; ++j)
        free(a[j]);
      free(a);
      printf("Error: there is no memory.\n");
      return 1;
    }
  }

  /* вводим элементы массива */
  printf("Input elements of the massiva:\n");
  for (i = 0; i < n; ++i)
    for (j = 0; j < m; ++j)
      scanf("%d", &a[i][j]);

  /* что-то делаем с массивом */

  printf("You input the massiva:\n");
  for (i = 0; i < n; ++i)
  {
    for (j = 0; j < m; ++j)
      printf("%d ", a[i][j]);
    printf("\n");
  }

  /* освобождаем выделенную память */
  for (i = 0; i < n; ++i)
    free(a[i]);
  free(a);
  return 0;
}
```

2.6. Задачи для самостоятельного решения

Написать программы для решения следующих задач.

1. Получить, если это возможно, из заданного натурального числа путем перестановки его цифр палиндром. Для решения задачи использовать статический массив размерности «десять».

2. Отсортировать целочисленный массив методами вставки, выборки или обмена. Выбор типа сортировки запрашивать с консоли. Размерность и элементы массива вводятся с консоли.
3. Используя бинарный поиск, найти элемент в отсортированном целочисленном массиве. Размерность, элементы и искомый элемент массива вводятся с консоли.
4. Определить, есть ли в целочисленной квадратной матрице 5-го порядка прямоугольник, вершинами которого являются заданные числа. Элементы матрицы и числа вводятся с консоли.
5. Найти в целочисленной квадратной матрице такие элементы, которые являются одновременно наибольшими в столбце и наименьшими в строке. Размерность и элементы матрицы вводятся с консоли.

2.7. Дополнительные задачи

1. Найти медиану в записи натурального числа, которое вводится с консоли в переменную типа `unsigned`. Под медианой будем понимать такую цифру в записи числа, которая по величине больше и меньше одинакового количества цифр в записи этого же числа.
2. Найти в целочисленном массиве за один просмотр пять наименьших элементов. Размерность и элементы массива вводятся с консоли.
3. Отсортировать целочисленный массив бинарным деревом (пирамидальная сортировка). Размерность и элементы массива вводятся с консоли.
4. В квадратной матрице 5-го порядка, элементами которой являются нули и единицы, найти наибольший квадрат, заполненный единицами.
5. Найти квадратные матрицы, которые являются параллельными проекциями кубической матрицы на координатные плоскости. Элемент кубической матрицы, который находится ближе к координатной плоскости, закрывает элементы, находящиеся за ним. Размерность и элементы кубической матрицы вводятся с консоли.

2.8. Задачи для индивидуальной работы

1. ***Слияние.*** Для двух заданных конечных упорядоченных по возрастанию последовательностей натуральных чисел получить за один просмотр новую последовательность, элементы которой также упорядочены по возрастанию и включены в последовательность по принципу:

- пересечения элементов двух исходных последовательностей;
- полного объединения элементов двух исходных последовательностей;

- исключения из первой последовательности элементов второй последовательности.

2. *Фрагмент.* В заданной последовательности *N* натуральных чисел выбрать подпоследовательность максимальной длины, являющуюся:

- возрастающей подпоследовательностью;
- подпоследовательностью-палиндромом;
- арифметической прогрессией.

3. *Цепочки.* Выбрать из заданной последовательности *N* натуральных чисел возрастающие подпоследовательности.

4. *Периодическая последовательность.* Определить, является ли заданная последовательность *N* натуральных чисел периодической.

5. *Минимальная разность.* В заданной последовательности *N* целых чисел найти такое число, для которого разность между суммами предшествующих ему и следующих за ним чисел минимальна.

6. *Наилучшее приближение.* Заданы две конечные неубывающие последовательности целых чисел. За один просмотр этих последовательностей найти сумму двух элементов, взятых по одному из каждой последовательности, наиболее близкую к заданному числу *q*.

7. *Общие элементы.* Заданы три конечные неубывающие последовательности. За один просмотр этих последовательностей найти элементы, принадлежащие каждой из последовательностей.

8. *Прогрессии.* Составить программу, которая в заданной последовательности *N* натуральных чисел находит группы рядом стоящих чисел, образующих:

- геометрическую прогрессию с заданным множителем;
- геометрическую прогрессию с произвольным множителем;
- арифметическую прогрессию с заданным коэффициентом;
- арифметическую прогрессию с произвольным коэффициентом.

9. *Группы чисел.* В заданной последовательности *N* натуральных чисел заменить за один просмотр группу рядом стоящих:

- совершенных чисел на максимальный элемент этой группы;
- сверхпростых чисел на последний элемент этой группы;
- автоморфных чисел на сумму всех элементов этой группы.

10. *Две группы чисел.* Составить программу, которая в заданной последовательности *N* натуральных чисел находит за один просмотр две рядом стоящие группы чисел, из которых:

- первая содержит простые числа, а вторая числа-палиндромы;
- первая является обратной записью второй;

- элементы первой группы больше, а второй – меньше некоторого заданного числа *M*;
- элементы первой группы образуют арифметическую прогрессию, а элементы второй группы образуют геометрическую прогрессию.

11.*Посетители супермаркета.* Каждый входящий в супермаркет посетитель с помощью специальной карточки отмечает в компьютере время прихода и ухода из супермаркета. Найти интервал времени, в течение которого в супермаркете находилось максимальное количество посетителей.

12. *Минимальное удаление.* В заданной конечной последовательности натуральных чисел удалить минимальное количество элементов так, чтобы оставшиеся элементы образовали:

- возрастающую последовательность;
- последовательность-палиндром;
- арифметическую прогрессию;
- пилообразную последовательность.

13. *Квадрат.* В заданной квадратной матрице 6-го порядка, состоящей из 0 и 1, переставить строки и столбцы таким образом, чтобы в центре матрицы сформировался квадрат из четырех единиц.

14. *Два квадрата.* В заданной квадратной матрице 6-го порядка, состоящей из 0 и 1, переставить строки и столбцы таким образом, чтобы в левом верхнем углу и правом нижнем углу матрицы сформировались квадраты из четырех единиц.

15. *Максимальные последовательности из единиц.* Для заданной матрицы порядка *N***N*, элементы которой нули и единицы, найти самую длинную последовательность подряд стоящих единиц по горизонтали, вертикали или диагонали.

16. *Циклический сдвиг.* Для заданной матрицы порядка *N***M*, элементы которой натуральные числа, найти номера строк, совпадающих полностью или в результате циклического сдвига одной из них на некоторое количество элементов.

17. *Замена.* Заменить каждый элемент заданной матрицы порядка *N***M* минимальным элементом, выбираемым среди элементов, стоящих не ниже и не правее этого элемента, включая его значение.

18. *Упакованная матрица.* Умножить упакованную в одномерный массив симметричную матрицу размером *N***N* на вектор размером *N*, не распаковывая ее.

19. *Удаление нулевых строк.* Сжать упакованную в одномерный массив симметричную матрицу размером *N***N*, удалив полностью нулевые строки и столбцы, не распаковывая ее.

20. ***Кладоискатель.*** Лабиринт задается квадратной матрицей 10-го порядка, состоящей из 1 и 0. Клетка считается проходимой, если она содержит 0, и непроходимой, если она содержит 1. Начальное положение кладоискателя задается координатами одной из крайних клеток. Кладоискатель может перемещаться из одной проходимой клетки в другую, если они имеют общую сторону. Местонахождение клада задается координатами одной из проходимых клеток. Найти любой и кратчайший пути кладоискателя.

21. ***Караван.*** План местности, разбитой на квадраты, представлен квадратной матрицей *N*-го порядка. Каждый квадрат имеет высоту относительно уровня моря, значение которой определяется целым числом. Необходимо определить маршрут каравана из позиции (X_n, Y_n) в позицию (X_k, Y_k) с минимальной суммой перепадов высот. Перемещаться из одной клетки в другую можно только при наличии общей стороны.

22. ***Шахматный конь.*** Для заданной позиции шахматной доски, на которой стоит шахматный конь, найти количество тех клеток, которые этот шахматный конь может достичь через указанное число ходов.

23. ***Два шахматных коня.*** Для двух заданных позиций шахматной доски, на которых стоят два шахматных коня, найти такую позицию, которую и один и второй конь могут достичь за одно и тоже минимальное число ходов.

24. ***Жизнь.*** На клетчатом поле расположены несколько «особей»; «особь» представляет собой одну клетку; клетки считаются 8-ми связными, т. е. каждая клетка имеет 8 соседних клеток; шаг – это смена поколения, при котором действуют следующие правила:

- *выживание*: особь, у которой есть 2 или 3 соседа, выживает;
- *гибель:* особь, имеющая более 3 соседей, погибает от перенаселения; особь, имеющая менее 2 соседей, гибнет от одиночества;
- *рождение:* если у пустой клетки имеется ровно 3 соседних особи, то в ней рождается особь.

Определить состояние системы через указанное число шагов; при этом обеспечить возможность пошагового наблюдения за поведением системы.

Лабораторная работа 3

Тема. Программирование функций. Рекурсивные функции. Функции с переменным количеством параметров. Стандартные функции сортировки и поиска.

3.1. Определение функций

Функция представляет собой поименованную последовательность инструкций, которая выполняет определенные действия. Функции используются для того, чтобы в тексте программы не повторять одни и те же блоки инструкций несколько раз. Например, определим функцию, которая вычисляет сумму двух целых чисел:

```
int add(int x, int y)
{
  int z;
  z = x + y;
  return z;
}
```

Как видим, определение функции состоит из двух частей: заголовка и тела, которое представляет собой блок инструкций. Заголовок функции включает тип возвращаемого функцией значения, имя функции и список параметров. В нашем случае функция возвращает значение типа `int`, имеет имя `add` и два формальных параметра `x` и `y`. Для вычисления суммы внутри функции объявляется переменная `z`, которая не видна вне функции и поэтому называется локальной переменной. Полученное значение суммы возвращается в вызывающую программу инструкцией `return`.

Эта же функция может быть определена более кратко:

```
int add(int x, int y)
{
  return x + y;
}
```

т. е. после ключевого слова `return` можно использовать выражение.

Если функция не возвращает никакого значения, то в языке программирования Си считается, что эта функция возвращает значение типа `void`, т. е. пустое значение. В этом случае для выхода из функции используется инструкция `return` без выражения. Если выход из функции, не возвращающей значения, происходит как выход из блока, то в этом случае инструкцию `return` можно опустить. Например, следующая функция напечатает сумму двух чисел:

```
void print(int x, int y)
{
  printf("x + y = %d\n", x + y);
}
```

Если функция не имеет параметров, то вместо списка параметров пишется ключевое слово `void`. Например, следующая функция печатает сообщение «Hello»:

```
void hello(void)
{
  printf("Hello.\n");
}
```

Пустой список параметров функции обозначает, что функция имеет неопределенное значение параметров.

Параметры функции можно рассматривать как локальные переменные, которые видны только в теле функции. Отсюда следует, что значения параметров можно изменять в теле функции. Например, можно написать следующую функцию:

```
int inc(int n)
{
  return ++n;
}
```

которая вернет значение `n+1`. Заметим, что функция

```
int inc(int n)
{
  return n++;
}
```

вернет значение `n`.

3.2. Прототипы функций

Для предварительного описания функции может использоваться ее объявление или прототип. В отличие от определения функции, прототип содержит только заголовок функции и не включает ее тело. Прототипы функций используются в следующих случаях:

- функция определена в другом файле;
- функция определена после обращения к ней;
- функция определена в библиотеке функций, которая подключается к программе на этапе ее компоновки или выполнения.

Например, функция сложения двух чисел имеет следующий прототип:

```
int add(int x, int y);
```

При объявлении функции в списке параметров можно указывать только типы параметров, опуская их названия. Например, функцию `add` можно объявить следующим образом:

```
int add(int, int);
```

Заметим, что объявление функции заканчивается символом `';'`.

3.3. Вызов функции

Для вызова функции в программе пишется имя функции, за которым в скобках следует список аргументов, которые являются переменными и константами, определенными в программе. При этом происходит выполнение тела функции, в котором параметры заменяются значениями аргументов. Важно отметить, что тип аргумента должен соответствовать типу параметра или допускать приведение к типу параметра. Кроме того, заметим, что функция должна быть объявлена или определена до своего вызова.

Ниже приведена программа, в которой вызывается функция сложения двух чисел.

```
#include <stdio.h>
int add(int x, int y)
{
  return x + y;
}
int main()
{
  int z;
  /* вызываем функцию *.
  z = add(1, 2);
  /* печатаем результат */
  printf("z = %d\n", z);
  /* можно вызвать функцию так */
  printf("1 + 2 = %d\n", add(1, 2));
  /* можно также и так,              */
  /* но в этом случае непонятно зачем */
  add(1, 2);
  return 0;
}
```

Если функция не имеет параметров, то при ее вызове аргументы не указываются, но круглые скобки остаются. В следующей программе показан вызов функции `hello`:

```
#include <stdio.h>
void hello(void)
```

```
{
  printf("Hello.\n");
}
int main()
{
  hello();
  return 0;
}
```

В заключение этого параграфа сделаем два очень важных замечания. Во-первых, аргументы передаются в функцию по значению. Т. е. значения аргументов переписываются в параметры функции. Так как значение аргумента копируется в тело функции, то сам аргумент изменить в теле функции невозможно. Во-вторых, функция возвращает результат также по значению. Т. е. значение, вычисленное в инструкции `return`, копируется в переменную, которая определена в теле вызывающей программы и которой присваивается результат вычисления функции.

3.4. Рекурсивные функции

В языке программирования Си возможно определение рекурсивных функций. Например, определим рекурсивную функцию, которая вычисляет факториал натурального числа:

```
unsigned factorial(unsigned n)
{
  if (!n)
    return 1;
  else
    return n * factorial(n - 1);
}
```

3.5. Передача аргументов через указатели

Для того чтобы функция могла изменить значения переменных, которые определены в программе, нужно передать в функцию указатели на эти переменные. Например, в следующей программе определена функция, которая увеличивает на единицу значение переменной, определенной в программе.

```
#include <stdio.h>
void inc(int* x)
{
  ++(*x);
}
int main()
{
```

```
    int x = 1;
    /* вызываем функцию */
    inc(&x);
    /* печатаем результат */
    printf("x = %d\n", x);
    return 0;
}
```

Отметим, что в этом случае в функцию должен передаваться адрес переменной, а не сама переменная.

3.6. Функции с переменным количеством параметров

В языке программирования Си допускается использование функций с переменным количеством параметров. В этом случае в конце списка параметров ставится многоточие «…». При этом предполагается, что в списке параметров присутствует хотя бы один параметр. Примерами функций с переменным количеством параметров являются стандартные функции `scanf` и `printf`. Например, можно определить функцию для сложения произвольного количества целых чисел следующим образом:

```
int add_num(int n, ...);  /* n - количество слагаемых */
```

Для обработки функций с переменным количеством параметров используется переменная типа `va_list` и три макрокоманды:

```
va_start(va_list ap, last_arg);
va_arg(va_list ap, type);
va_end(va_list ap);
```

Макрокоманда `va_start` инициализирует переменную `ap` для начала извлечения аргументов функции, а параметр `last_arg` должен быть именем последнего заданного параметра функции.

Макрокоманда `va_arg` возвращает следующий аргумент функции, причем параметр `ap` должен совпадать с соответствующим параметром из макрокоманды `va_start`, а параметр `type` должен указывать тип следующего аргумента.

Макрокоманда `va_end` после завершения работы с аргументами функции обеспечивает правильную работу инструкции `return` в функции с переменным количеством параметров.

Все эти макрокоманды и тип `va_list` определены в заголовочном файле `stdarg.h`.

Например, определим функцию `print`, которая распечатывает переменное количество целых чисел, которые передаются ей как параметры.

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

```
#include <stdarg.h>

int print(int n, ...)
{
  va_list  a;
  /* проверяем, есть ли параметры */
  if (n < 1)
  {
    printf("There are no numbers.\n");
    return 0;
  }
  printf("Number of parameters = %d\n", n);

  /* инициализируем ‘a’ последним заданным аргументом */
  va_start(a, n);

  while (n)    /* распечатываем аргументы */
  {
    int x;

    /* получаем следующий аргумент */
    x = va_arg(a, int);
    /* распечатываем аргумент */
    printf("%d\n", x);
    --n;
  }

  /* завершение продвижения по аргументам */
  va_end(a);
  return  0;
}

/* проверяем работу функции print */
int main()
{
  print(0);
  print(1, 1);
  print(2, 10, 20);
  print(3, 100, 200, 300);
  return 0;
}
```

3.7. Указатели на функции

Имя функции является адресом этой функции в памяти. Язык программирования Си позволяет определять указатели на функции. Например, инструкция

```
int (*fp)(int x, int y);
```

объявляет переменную `fp`, которая является указателем на функцию. Причем эта функция должна иметь два параметра типа `int` и возвращать

значение также типа int. В следующей программе функции сложения и вычитания двух чисел вызываются через указатель.

```
#include <stdio.h>

int add(int x, int y)
{
  return x + y;
}

int sub(int x, int y)
{
  return x - y;
}

int main()
{
  int (*p)(int, int);

  p = add;
  /* вызываем функцию через указатель */
  printf("1 + 2 = %d\n", (*p)(1, 2));

  p = sub;
  /* можно вызвать функцию и так */
  printf("1 - 2 = %d\n", p(1, 2));
  return 0;
}
```

Так как указателю на функцию могут присваиваться адреса различных функций при условии совпадения типа параметров и типа возвращаемого значения, то указатели на функции часто называют функторами.

3.8. Вызов стандартных функций сортировки и поиска

На практике при работе с данными часто встречаются задачи сортировки массива и поиска элементов в отсортированном массиве. Причем массивы могут иметь разные типы. Чтобы облегчить решение таких задач, в стандартной библиотеке языка программирования Си есть специальные функции qsort и bsearch, которые предназначены для решения этих задач.

Функция qsort выполняют сортировку массива, элементы которого имеют произвольный тип. Эта функция реализует «быстрый алгоритм» сортировки массивов и имеет следующий прототип:

```
void qsort(void *base, size_t n, size_t size,
    int (*cmp)(const void *e1, const void *e2));
```

который описан в заголовочном файле stdlib.h. Кратко опишем назначение параметров этой функции:

base – адрес массива,
n – количество элементов в массиве,
size – длина элемента массива,
cmp – указатель на функцию сравнения, которая возвращает:

- отрицательное число, если элемент e1 меньше элемента e2;
- 0, если элемент e1 равен элементу e2;
- положительное число, если элемент e1 больше элемента e2.

Функция bsearch выполняет бинарный поиск элемента в отсортированном массиве. Эта функция имеет следующий прототип:

```
void* bsearch(const void *key, const void *base, size_t n,
  size_t size, int (*cmp)(const void *ck, const void *ce);
```

который также описан в заголовочном файле stdlib.h. Первый параметр key этой функции является указателем на элемент, который нужно найти. Остальные параметры повторяют параметры функции qsort. В случае успешного завершения поиска функция bsearch возвращает адрес найденного элемента, а в случае неудачи – NULL.

В следующей программе показан пример использования функций qsort и bsearch для сортировки целочисленного массива и дальнейшего поиска элементов в этом отсортированном массиве.

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

/* функция для сравнения элементов массива */
int comp_int(const int* e1, const int* e2)
{
  return (*e1 < *e2) ? -1 : ((*e1 == *e2) ? 0 : 1);
}
/* программа сортировки элементов массива и поиска целого*/
/*  числа в отсортированном массиве */
int main()
{
  int  n;    /* размер массива *.
  int* a;    /* массив */
  int  i;    /* индекс */
  int  k;    /* число для поиска */
  int* s;    /* адрес найденного числа */

  printf("Input an array size: ");
  scanf("%d", &n);
  a = (int*)malloc(n*sizeof(int));

  /* вводим массив */
  printf("Input elements: ");
```

```
  for (i = 0; i < n; ++i)
    scanf("%d", &a[i]);

  /* сортируем массив */
  qsort(a, n, sizeof(int),
    (int (*)(const void*, const void*))comp_int);

  /* выводим отсортированный массив */
  printf("The sorted array: ");
  for (i = 0; i < n; ++i)
    printf("%d ", a[i]);
  printf("\n");

  /* вводим число для поиска */
  printf("Input a number to search.\n>");
  scanf("%d", &k);

  /* ищем это число в отсортированном массиве */
  if(!(s = (int*) bsearch(&k, a, n, sizeof(int),
      (int (*)(const void*, const void*))comp_int)))
    printf("There is no such an integer.\n");
  else
    printf("The integer index = %d.\n", s-a);
  free(a);

  return 0;
}
```

3.9. Задачи для самостоятельного решения

Написать программы, реализующие следующие функции. Продемонстрировать работу функций путем их вызова из функции `main`.

1. Получить все числа Мерсена в заданном интервале натурального ряда. Числом Мерсена называется простое число n, которое представимо в виде $n = 2^p - 1$, где p – также простое число. Например, $31 = 2^5 - 1$, $127 = 2^7 - 1$. При решении задачи использовать функцию для определения, является ли заданное натуральное число простым.

2. Написать функцию для нахождения минимального и максимального элементов массива, т. е. минимальный и максимальный элементы находятся одним вызовом функции.

3. Решить систему линейных уравнений второго порядка методом Крамера. Для вычисления определителей использовать функцию, параметром которой является матрица второго порядка.

4. Написать функцию для вычисления суммы или произведения переменного количества целочисленных параметров.

5. Разработать универсальную функцию сортировки массива, которая для сравнения элементов массива вызывает функцию сравнения, адрес

которой передается как параметр через указатель. Функция сортировки имеет следующий прототип:

```
void sort(void* a, int num, int len,
  int (*cmp)(const void*, const  void*));
```

где

`a` – адрес массива,

`num` – количество элементов в массиве,

`len` – длина элемента в массиве,

`cmp` – указатель на функцию сравнения, которая возвращает следующие значения:

`-1`, если первый параметр меньше второго;

`0`, если параметры равны;

`1`, если первый параметр больше второго.

В функции сортировки использовать любой алгоритм сортировки.

3.10. Дополнительные задачи

Написать программы, реализующие следующие функции. Продемонстрировать работу функций путем их вызова из функции `main`.

1. Вывести на консоль все сверхпростые числа в заданном интервале натурального ряда. Натуральное число называется сверхпростым, если оно остается простым при любой перестановке своих цифр. Для решения задачи использовать функции генерации перестановок и определения, является ли заданное натуральное число простым.

2. Написать функцию для определения НОД множества заданных натуральных чисел. Количество натуральных чисел и сами числа вводятся с консоли. При решении задачи использовать функцию для нахождения НОД двух чисел.

3. Найти все пары дружественных натуральных чисел в заданном интервале. Два натуральных числа называются дружественными, если сумма делителей одного из них равна другому и наоборот. При решении задачи использовать функцию для нахождения суммы делителей натурального числа, исключая само число.

4. Реализовать бинарный поиск в отсортированном массиве, используя рекурсивную функцию.

5. Написать рекурсивную функцию для нахождения корня нелинейного уравнения, используя метод деления отрезка пополам. Для вычисления значения функции использовать универсальную функцию. Продемонстрировать работу программы, найдя корень уравнения $x^3 - 1 - \sin x = 0$ на отрезке [1;3].

3.11. Задачи для индивидуальной работы

1. ***Перестановки цифр.*** Известно, что элементы заданной последовательности *N* натуральных чисел являются перестановками цифр *M* чисел (*M*<*N*). Переставить элементы последовательности так, чтобы элементы, являющиеся перестановками цифр одного числа стояли подряд. *Замечание*: используется функция, возвращающая число, цифры которого упорядочены по убыванию или возрастанию.

2. ***Фрагмент.*** Составить программу, которая получает новое число, выделив из записи заданного натурального числа фрагмент максимальной длины:

- состоящий из одинаковых цифр;
- являющийся отрезком натурального ряда;
- расположенный между двумя одинаковыми цифрами;
- расположенный между максимальной и минимальной цифрами, включая и сами эти цифры;
- образующий арифметическую (геометрическую) прогрессию;
- образующий возрастающую последовательность степеней двойки;
- являющийся неубывающей последовательностью.

3. ***Определитель.*** Написать рекурсивную функцию для вычисления определителя квадратной матрицы порядка *n*.

4. ***Симметрия.*** Написать рекурсивную функцию для определения, является ли заданный фрагмент целочисленного массива симметричным.

5. ***Максимальный элемент.*** Написать рекурсивную функцию для поиска в заданном целочисленном массиве максимального элемента.

6. ***Многоугольник.*** Написать рекурсивную функцию для вычисления площади выпуклого многоугольника, используя функцию для вычисления площади треугольника.

7. ***Числа-близнецы.*** Найти все числа-близнецы в заданном интервале натуральных чисел. Близнецами называются два простых числа, разность между которыми равна двум. Использовать функцию, которая проверяет, является ли заданное натуральное число простым.

Лабораторная работа 4

Тема. Строки. Ввод-вывод строк. Форматированный ввод-вывод. Обработка строк с использованием стандартных функций языка Си. Работа с памятью.

4.1. Объявление и инициализация строк

Строкой называется массив символов, который заканчивается пустым символом `'\0'`. Строка объявляется как обычный символьный массив, например:

```
char s1[] = "This is a string.";
char *s2 = "This is a string.";
```

Различие между указателями `s1` и `s2` заключается в том, что указатель `s1` является именованной константой, а указатель `s2` – переменной.

Строковые константы заключаются в двойные кавычки в отличие от символов, которые заключаются в одинарные кавычки. Например,

```
"This is a string."
```

Длина строковой константы не может превышать 509 символов по стандарту. Однако многие реализации допускают строки большей длины.

При инициализации строк размерность массива лучше не указывать, это выполнит компилятор, подсчитав длину строки и добавив к ней единицу.

В языке программирования Си для работы со строками существует большое количество функций, прототипы которых описаны в заголовочных файлах `stdlib.h` и `string.h`. Работа с этими функциями будет рассмотрена в следующих параграфах.

4.2. Ввод-вывод строк

Для ввода строки с консоли служит функция

```
char* gets(char *str);
```

которая записывает строку по адресу `str` и возвращает адрес введенной строки. Функция прекращает ввод, если встретит символ `'\n'` или `EOF` (конец файла). Символ перехода на новую строку не копируется. В конец прочитанной строки помещается нулевой байт. В случае успеха функция возвращает указатель на прочитанную строку, а в случае неудачи — `NULL`.

Для вывода строки на консоль служит стандартная функция

```
int puts(const char *s);
```

которая в случае удачи возвращает неотрицательное число, а в случае неудачи – EOF.

Прототипы функций gets и puts описаны в заголовочном файле stdio.h. Например,

```
#include <stdio.h>

int main()
{
  char str[80];
  printf("Input String: ");
  gets(str);
  puts(str);
  return 0;
}
```

4.3. Форматированный ввод-вывод

Для форматированного ввода данных с консоли используется функция

```
int scanf(const char *format, ...);
```

которая в случае успешного завершения возвращает количество единиц прочитанных данных, а в случае неудачи – EOF. Параметр format должен указывать на форматируемую строку, которая содержит спецификации форматов ввода. Количество и типы аргументов, которые следуют после строки форматирования, должны соответствовать количеству и типам форматов ввода, заданным в строке форматирования. Если это условие не выполняется, то результат работы функции непредсказуем.

Пробел, символы '\t' или '\n' в форматной строке описывают один или более пустых символов во входном потоке, к которым относятся символы: пробел, '\t', '\n', '\v', '\f'. Функция scanf пропускает пустые символы во входном потоке.

Литеральные символы в форматной строке, за исключением символа '%', требуют, чтобы во входном потоке появились точно такие же символы. Если такого символа нет, то функция scanf прекращает ввод. Функция scanf пропускает литеральные символы.

В общем случае спецификация формата ввода имеет вид:

```
%[*] [ширина] [модификаторы] тип
```

где

- символ '*' обозначает пропуск при вводе поля, определенного данной спецификацией;
- «ширина» определяет максимальное число символов, вводимых по данной спецификации;

- «модификаторы» уточняют тип аргументов;
- «тип» определяет тип аргумента.

Тип может принимать следующие значения:

- `c` – символьный массив;
- `s` – строка символов, строки разделяются пустыми символами;
- `d` – целое число со знаком в 10 с/с;
- `i` – целое число со знаком, система счисления завит от двух первых цифр;
- `u` – целое число без знака в 10 с/с;
- – целое число без знака в 8 с/с;
- `x`, `X` – целое число без знака в 16 с/с;
- `e`, `E`, `f`, `g`, `G` – плавающее число;
- `p` – указатель на указатель;
- `n` – указатель на целое;
- `[…]` – массив сканируемых символов, например `[A321]`.

В последнем случае из входного потока будут вводиться только символы, заключенные в квадратные скобки. Если первый символ внутри квадратных скобок равен `'^'`, то вводятся только те символы, которые не входят в массив. Диапазон символов в массиве задается через символ `'-'`. При вводе символов ведущие пустые символы и завершающий нулевой байт строки также вводятся.

Модификаторы могут принимать следующие значения:

- `h` – короткое целое,
- `l`, `L` – длинное целое или плавающее,

и используются только для целых или плавающих чисел.

В следующем примере показаны варианты использования функции scanf. Обратите внимание, что перед спецификатором формата, начиная с ввода плавающего числа, стоит символ «пробел».

```
#include <stdio.h>

int main()
{
  int  n;
  double  d = 0.0;
  char  c;
  char*  s;

  printf("Input an integer: ");
```

```
    scanf("%d", &n);

    printf("Input a double: ");
    scanf(" %lf", &d);

    printf("Input a char: ");
    scanf(" %c", &c);

    printf("Input a word: ");
    scanf(" %s", &s);
    return 0;
}
```

Обратите внимание, что в этой программе число с плавающей точкой проинициализировано. Это сделано для того, чтобы компилятор подключил библиотеку для поддержки работы с плавающими числами. Если этого не сделать, то на этапе выполнения при вводе числа с плавающей точкой произойдет ошибка.

Для форматированного вывода данных на консоль используется функция

```
int printf(const char *format, ...);
```

которая в случае успешного завершения возвращает количество единиц выведенных данных, а в случае неудачи – EOF. Параметр `format` представляет собой форматируемую строку, которая содержит спецификации форматов вывода. Количество и типы аргументов, которые следуют после строки форматирования, должны соответствовать количеству и типам спецификаций формата вывода, заданным в строке форматирования. В общем случае спецификация формата вывода имеет следующий вид:

```
%[флаги] [ширина] [.точность] [модификаторы] тип
```

где

- «флаги» – это различные символы, уточняющие формат вывода;
- «ширина» определяет минимальное количество символов, выводимых по данной спецификации;
- «точность» определяет максимальное число выводимых символов;
- «модификаторы» уточняют тип аргументов;
- «тип» определяет тип аргумента.

Для вывода целых чисел со знаком используется следующий формат вывода:

```
%[-] [+ | пробел] [ширина] [l] d
```

где

- – выравнивание влево, по умолчанию – вправо;
- + – выводится знак '+', заметим, что для отрицательных чисел всегда выводится знак '-';
- «пробел» – в позиции знака выводится пробел;
- l – модификатор типа данных long;
- d – тип данных int.

Для вывода целых чисел без знака используется следующий формат вывода:

```
%[-] [#] [ширина] [l] [u|o|x|X]
```

где

- # – выводится начальный 0 для чисел в 8 с/с или начальные 0x или 0X для чисел в 16 с/с,
- l – модификатор типа данных long;
- u – целое число в 10с/с,
- o – целое число в 8 с/с,
- x, X – целое число в 16 с/с.

Для вывода чисел с плавающей точкой используется следующий формат вывода:

```
%[-] [+ | пробел] [ширина] [.точность] [f|e|E|g|G]
```

где

- «точность» – обозначает число цифр после десятичной точки для форматов f, e и E или число значащих цифр для форматов g и G. Числа округляются отбрасыванием. По умолчанию принимается точность в шесть десятичных цифр;
- f – число с фиксированной точкой,
- e – число в экспоненциальной форме, экспонента обозначается буквой 'e',
- E – число в экспоненциальной форме, экспонента обозначается буквой 'E';
- g – наиболее короткий из форматов f или g;
- G – наиболее короткий из форматов f или G.

Например, программа:

```
#include <stdio.h>

int main()
{
```

```
  printf("n = %d\n", -123);
  printf("f = %f\n", 12.34);
  printf("e = %e\n", 12.34);
  printf("E = %E\n", 12.34);
  printf("f = %.2f", 12.34);
  return 0;
}
```

выведет на консоль следующие числа:

```
n = 123
f = 12.340000
e = 1.234000e+001
E = 1.234000E+001
f = 12.34
```

4.4. Форматирование строк

Существуют варианты функций `scanf` и `printf`, которые предназначены для форматирования строк и называются соответственно `sscanf` и `sprintf`.

Функция

```
int  sscanf(const char *str, const char *format, ...);
```

читает данные из строки, заданной параметром `str`, в соответствии с форматной строкой, заданной параметром `format`. В случае удачи возвращает количество прочитанных данных, а в случае неудачи – `EOF`. Например,

```
#include <stdio.h>

int main()
{
  char str[] = "a 10 1.2 String No input";
  char c;
  int  n;
  double d;
  char s[80];
  sscanf(str, "%c %d %lf %s", &c, &n, &d, s);
  printf("%c\n", c);    /* печатает: a */
  printf("%d\n", n);    /* печатает: 10 */
  printf("%f\n", d);    /* печатает: 1.200000 */
  printf("%s\n", s);    /* печатает: String */
  return 0;
}
```

Функция

```
int sprintf(char *buffer, const char *format, ...);
```

форматирует строку в соответствии с форматом, который задан параметром format и записывает полученный результат в символьный массив buffer. Возвращает функция количество символов, записанных в символьный массив buffer, исключая завершающий нулевой байт. Например:

```
#include <stdio.h>

int main()
{
  char buffer[80];
  char str[] = "c = %c, n = %d, d = %f, s = %s";
  char c = 'c';
  int  n = 10;
  double d = 1.2;
  char s[] = "This is a string.";
  sprintf(buffer, str, c, n, d, s);
  printf("%s\n", buffer);
  /* печатает: /*
  /*c = c, n = 10, d = 1.200000, s = This is a string */
  return 0;
}
```

4.5. Преобразование строк в числовые данные

Прототипы функций преобразования строк в числовые данные приведены в заголовочном файле stdlib.h, который нужно включить в программу.

Для преобразования строки в целое число используется функция

```
int atoi(const char *str);
```

которая в случае успешного завершения возвращает целое число, в которое преобразована строка str, а в случае неудачи – 0. Например:

```
int n;
char *str = “-123”;
n = atoi(str);   /* n = -123 */
```

Для преобразования строки в длинное целое число используется функция

```
long int atol(const char *str);
```

которая в случае успешного завершения возвращает целое число, в которое преобразована строка str, а в случае неудачи – 0. Например:

```
long int n;
char *str = “-123”;
n = atol(str);  /* n = -123 */
```

Для преобразования строки в число типа double используется функция

```
double atof(const char  *str);
```

которая в случае успешного завершения возвращает плавающее число типа double, в которое преобразована строка str, а в случае неудачи – 0. Например:

```
double n;
char *str = "-123.321";
n = atof(str);  /* n = -123.321 */
```

Следующие функции выполняют действия, аналогичные функциям atoi, atol, atof, но предоставляют более широкие возможности.

Функция

```
long int strtol(const char *str, char **endptr, int base);
```

преобразует строку str в число типа long int, которое и возвращает. Параметры этой функции имеют следующее назначение.

Если аргумент base равен 0, то преобразование зависит от первых двух символов строки str:

- если первый символ – цифра от 1 до 9, то предполагается, что число представлено в 10 с/с;
- если первый символ – цифра 0, а второй – цифра от 1 до 7, то предполагается, что число представлено в 8 с/с;
- если первый символ 0, а второй – 'X' или 'x', то предполагается, что число представлено в 16 с/с.

Если аргумент base равен числу от 2 до 36, то это значение принимается за основание системы счисления и любой символ, выходящий за рамки этой системы, прекращает преобразование. В системах счисления с основанием от 11 до 36 для обозначения цифр используются символы от 'A' до 'Z' или от 'a' до 'z'.

Значение аргумента endptr устанавливается функцией strtol. Это значение содержит указатель на символ, который остановил преобразование строки str. В случае успешного завершения функция strtol возвращает преобразованное число, а в случае неудачи – 0. Например:

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

int main()
{
  long int  n;
```

```
  char  *p;
  n = strtol ("12a", &p, 0);
  printf(" n = %ld, stop = %c\n", n, *p );
      /* печатает:  n = 12, stop = a */
  n = strtol("012b", &p, 0);
  printf(" n = %ld, stop = %c\n", n, *p );
      /* печатает: n = 10, stop = b */
  n = strtol("0x12z", &p, 0);
  printf(" n = %ld, stop = %c\n", n, *p );
      /* печатает: n = 18, stop = z */
  n = strtol("01119", &p, 0);
  printf(" n = %ld, stop = %c\n", n, *p );
      /* печатает: n = 7, stop = 9 */
  return 0;
}
```

Функция

```
unsigned long int  strtol(const char *str, char **endptr,
  int base);
```

работает аналогично функции `strtol`, но преобразует символьное представление числа в число типа `unsigned long int`.

Функция

```
double strtod(const char *str, char **endptr);
```

преобразует символьное представление числа в число типа `double`.

Все функции, перечисленные в этом параграфе, прекращают свою работу при встрече первого символа, который не подходит под формат рассматриваемого числа.

Кроме того, в случае если символьное значение числа превосходит диапазон допустимых значений для соответствующего типа данных, то функции `atof`, `strtol`, `strtoul`, `strtod` устанавливают значение переменной `errno` в `ERANGE`. Переменная `errno` и константа `ERANGE` определены в заголовочном файле `math.h`. При этом функции `atof` и `strtod` возвращают значение `HUGE_VAL`, функция `strtol` возвращает значение `LONG_MAX` или `LONG_MIN`, а функция `strtoul` – значение `ULONG_MAX`.

Для преобразования числовых данных в символьные строки могут использоваться нестандартные функции `itoa`, `ltoa`, `utoa`, `ecvt`, `fcvt` и `gcvt`. Но лучше для этих целей использовать стандартную функцию `sprintf`.

4.6. Стандартные функции для работы со строками

В этом параграфе рассмотрены функции для работы со строками, прототипы которых описаны в заголовочном файле `string.h`.

1. ***Сравнение строк***. Для сравнения строк используются функции `strcmp` и `strncmp`.

Функция

```
int strcmp(const char *str1, const char *str2);
```

лексикографически сравнивает строки `str1` и `str2`. Функция возвращает `−1`, `0` или `1`, если строка `str1` соответственно меньше, равна или больше строки `str2`.

Функция

```
int strncmp(const char *str1, const char *str2, size_t n);
```

лексикографически сравнивает не более чем n первых символов из строк `str1` и `str2`. Функция возвращает `−1`, `0` или `1`, если первые `n` символов из строки `str1` соответственно меньше, равны или больше первых `n` символов из строки `str2`. Например:

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

int main()
{
  char str1[] = "aa bb";
  char str2[] = "aa aa";
  char str3[] = "aa bb cc";
  printf("%d\n", strcmp(str1, str3));
              /* печатает: -1 */
  printf("%d\n", strcmp(str1, str1));
              /* печатает: 0 */
  printf("%d\n", strcmp(str1, str2));
              /* печатает: 1 */
  printf("%d\n", strncmp(str1, str3, 5));
              /* печатает: 0 */
  return 0;
}
```

2. ***Копирование строк.*** Для копирования строк используются функции `strcpy` и `strncpy`.

Функция

```
char *strcpy(char *strDst, const char *strSrc);
```

копирует строку `strSrc` в строку `strDst`. Строка `strSrc` копируется полностью, включая завершающий нулевой байт. Функция возвращает

указатель на strDst. Если строки перекрываются, то результат непредсказуем.

Функция

```
char *strncpy(char *strDst, const char *strSrc, size_t n );
```

копирует n символов из строки strSrc в строку strDst. Если строка strSrc содержит меньше чем n символов, то последний нулевой байт копируется столько раз, сколько нужно для расширения строки strSrc до n символов. Функция возвращает указатель на строку str1. Например:

```
char str1[80];
char str2[] = "Copy string.";

strcpy(str1, str2);
printf(str1);  /* печатает: Copy string. */
```

3. ***Соединение строк***. Для соединения строк в одну строку используются функции strcat и strncat.

Функция

```
char* strcat(char *strDst, const char *strSrc);
```

присоединяет строку strSrc к строке strDst, причем завершающий нулевой байт строки strDst стирается. Функция возвращает указатель на строку strDst.

Функция

```
char* strncat(char *strDst, const char *strSrc, size_t n );
```

присоединяет n символов из строки strSrc к строке strDst, причем завершающий нулевой байт строки strDst стирается. Если длина строки strSrc меньше n, то присоединяются только символы, входящие в строку strSrc. После соединения строк к строке strDst всегда добавляется нулевой байт. Функция возвращает указатель на строку strDst. Например:

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

int main ( )
{
  char str1[80] = "String ";
  char str2[] = "catenation ";
  char str3[] = "Yes No";
  strcat(str1, str2 );
  printf("%s\n", str1 );  /* String catenation */
```

```
  strncat( str1, str3, 3 );
  printf("%s\n", str1 );/* String catenation Yes */
  return 0;
}
```

4. *Поиск символа в строке.* Для поиска символа в строке используются функции `strchr`, `strrchr`, `strspn`, `strcspn` и `strpbrk`.

Функция

```
char* strchr(const char *str, int c);
```

ищет первое вхождение символа, заданного параметром `c`, в строку `str`. В случае успеха функция возвращает указатель на первый найденный символ, а в случае неудачи – `NULL`.

Функция

```
char* strrchr(const char *str, int c);
```

ищет последнее вхождение символа, заданного параметром `c`, в строку `str`. В случае успеха функция возвращает указатель на последний найденный символ, а в случае неудачи – `NULL`. Например,

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

int  main()
{
  char str[80] = "Char search";

  printf("%s\n", strchr ( str, 'r' ));
            /* печатает: r search */
  printf("%s\n", strrchr ( str, 'r' ));
            /* печатает: rch */
  return 0;
}
```

Функция

```
size_t strspn(const char *str1, const char *str2);
```

возвращает индекс первого символа из строки `str1`, который не входит в строку `str2`.

Функция

```
size_t strcspn(const char *str1, const char *str2);
```

возвращает индекс первого символа из строки `str1`, который входит в строку `str2`. Например:

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

```
int  main()
  {
    char str[80] = "123 abc";
    printf("n = %d\n", strspn ( str, "321" ));
              /* печатает: n = 3 */
    printf("n = %d\n", strcspn ( str, "cba" ));
              /* печатает: n = 4 */
    return 0;
  }
```

Функция

```
char* strpbrk(const char *str1, const char *str2);
```

находит в строке `str1` первый символ, который равен одному из символов в строке `str2`. В случае успеха функция возвращает указатель на этот символ, а в случае неудачи – `NULL`. Например:

```
char str[80] = "123 abc";
printf("%s\n", strpbrk(str, "bca"));  /* печатает: abc */
```

5. ***Сравнение строк.*** Для сравнения строк используются функция `strstr`.

Функция

```
char* strstr(const char *str1, const char *str2);
```

находит первое вхождение строки `str2` (без конечного нулевого байта) в строку `str1`. В случае успеха функция возвращает указатель на найденную подстроку, а в случае неудачи – `NULL`. Если указатель `str1` указывает на строку нулевой длины, то функция возвращает указатель `str1`. Например:

```
char str[80] = "123 abc 456”;
printf("%s\n", strstr(str, "abc"));  /* abc 456 */
```

6. ***Разбор строки на лексемы.*** Для разбора строки на лексемы используется функция `strtok`.

Функция

```
char* strtok(char *str1, const char *str2);
```

возвращает указатель на следующую лексему (слово) в строке `str1`, в которой разделителями лексем являются символы из строки `str2`. В случае если лексемы закончились, то функция возвращает значение `NULL`. При первом вызове функции `strtok` параметр `str1` должен указывать на строку, которая разбирается на лексемы, а при последующих вызовах этот параметр должен быть установлен в `NULL`. После нахожде-

ния лексемы функция strtok записывает после этой лексемы на место разделителя нулевой байт.

Например:

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

int main()
{
  char str[ ] = "12 34 ab cd";
  char *p;
  p = strtok(str, " ");
  while (p)
  {
    printf("%s\n", p );
      /* печатает в столбик значения: 12 34 ab cd */
    p = strtok(NULL, " ");
  }
  return 0;
}
```

7. *Определение длины строки.* Для определения длины строки используется функция strlen.

Функция

```
size_t strlen(const char *str);
```

возвращает длину строки, не учитывая последний нулевой байт. Например:

```
char str[] = "123";
printf("len = %d\n", strlen(str));  /* печатает: len = 3 */
```

4.7. Функции для работы с памятью

В заголовочном файле string.h описаны также функции для работы с блоками памяти, которые аналогичны соответствующим функциям для работы со строками.

Функция

```
void* memchr(const void *str, int c, size_t n);
```

ищет первое вхождение символа, заданного параметром c, в n байтах строки str.

Функция

```
int memcmp(const void *str1, const void *str2, size_t n);
```

сравнивает первые n байтов строк str1 и str2.

Функция

```
void* memcpy(const void *dst, const void *src, size_t n);
```

копирует первые n байтов из строки src в строку dst.

Функция

```
void* memmove(const void *dst, const void *src, size_t n);
```

копирует первые n байтов из строки src в строку dst, обеспечивая корректную обработку перекрывающихся строк.

Функция

```
void* memset(const void *str, int c, size_t n);
```

копирует символ, заданный параметром c, в первые n байтов строки str.

4.8. Задачи для самостоятельного решения

Реализовать следующие функции. Проверить работоспособность функций на строках, которые вводятся с консоли в функции main.

1. Функция для сортировки массива строк. Строки вводятся с консоли. Для сортировки используется стандартная функция qsort.

2. Функция для исключения слова из строки. Параметрами функции являются строка и слово, которое исключается из строки. Функция возвращает модифицированную строку.

3. Функция для вставки слова в строку после заданного слова. Параметрами функции являются исходная строка, два слова и новая строка, в которую нужно записать результат. Функция возвращает новую строку.

4. Функция для замены слова в строке на новое слово. Параметрами функции являются исходная строка, два слова и новая строка, в которую нужно записать результат. Функция возвращает новую строку.

5. Функция для замены числа в строке на новое число. Параметрами функции являются строка, два числа типа int и новая строка, в которую нужно записать результат. Функция возвращает модифицированную строку.

4.9. Дополнительные задачи

1. Написать программу разбора строки на слова с подсчетом количества разных слов и вывода их на консоль. Строка вводится с консоли.

2. Написать программу для вычисления значения арифметического выражения, которое может содержать операции '+', '-', '*' и '/', операндами которых могут быть целые и действительные числа, а для приоритета операций могут использоваться круглые скобки. Требование к реализации: при вычислении выражения должно выполняться неявное

преобразование типов, аналогичное преобразованию типов при вычислении значения выражения в языке программирования Си.

3. В заданной строке символов выделить фрагмент максимальной длины, который:

- является палиндромом;
- является повторяющейся подстрокой;

4. В заданной строке символов удалить (добавить) минимальное количество символов так, чтобы оставшиеся символы образовали:

- строку-палиндром;
- возрастающую в алфавитном порядке последовательность символов.

4.10. Задачи для индивидуальной работы

Задана строка со стандартным набором символов-разделителей между словами. Необходимо:

- проверить, является ли она палиндромом по словам;
- определить количество повторений каждого слова;
- найти слово, повторяющееся в строке максимальное количество раз;
- удалить повторения слов, сформировать строку из различных слов;
- подсчитать и выдать в порядке убывания количество слов, начинающихся с одинаковой буквы;
- переставить слова по убыванию количества гласных букв;
- переставить слова в алфавитном порядке;
- выделить фрагмент наибольшей длины, состоящий из одинаковых слов;
- выделить фрагмент наибольшей длины, состоящий из слов, упорядоченных по возрастанию длин;
- найти пары рядом стоящих слов, каждое из которых состоит из неповторяющихся букв;
- найти пары рядом стоящих слов, длины которых кратны;
- найти пары рядом стоящих слов, у которых совпадают множества букв, стоящих на нечетных позициях;
- найти пары рядом стоящих слов, из которых одно является зеркальным отображением другого.

Лабораторная работа 5

Тема. Перечисления. Структуры. Объединения. Битовые поля.

5.1. Перечисления

Перечислением называется тип данных, который включает множество именованных целочисленных констант. Именованные константы, принадлежащие перечислению, называются перечислимыми константами.

Объявляются перечисления следующим образом:

```
enum color {r, g, b};     /* объявление типа */
```

где `enum` – ключевое слово; `color` – имя типа перечисления; `r`, `g`, `b` – сами перечислимые константы. При объявлении типа перечисления его элементы могут инициализироваться произвольными целочисленными константами или константным выражением. Например:

```
enum color {r = 2, g = r = 2, b = 6};
```

Если инициализация отсутствует, то перечислимым константам присваиваются последовательные значения: `0`, `1`, `2` и т. д. Например:

```
enum color {r, g, b};  /* r = 0, g = 1, g = 2 */
```

Переменная типа перечисления объявляется следующим образом:

```
enum color c;
```

и также называется перечислением. Объявление типа перечисления и переменной, которая имеет этот тип, может быть объединено в одну инструкцию. Например:

```
enum turn {off, on} a;  /* a – переменная */
```

Переменным перечислимого типа можно присваивать только именованные значения перечислимых констант. Например,

```
color c = r;  /* правильно */
color c = 0;  /* ошибка */
```

Целочисленным переменным можно присваивать значения перечислимых констант. Например,

```
int  e = r;  /* правильно */
```

5.2. Структуры

Тип «структура» описывает упорядоченный набор данных, которые называются полями или членами структуры. Каждое поле структуры имеет имя и тип, который должен отличаться от типов `void` и функция. При этом следует учитывать, что структура может включать только та-

кие поля, длина которых известна компилятору в момент определения структуры.

Объявляются структуры следующим образом:

```
struct  emp
{
  int  empno;
  char  name[20];
  double  salary;
};
```

где `struct` – ключевое слово; `emp` – имя типа структуры; `empno`, `name`, `salary` – члены структуры.

Переменная типа структуры объявляется следующим образом:

```
struct emp e;
```

и также называется структурой. Объявление типа структуры и переменной, которая имеет этот тип, может быть объединено в одну инструкцию. Например:

```
struct  emp
{
  int  empno;
  char  name[20];
  double  salary;
}  director;      /* director – переменная */
```

Инициализируются структуры так же как и массивы. Например,

```
struct emp a = {10, "Paul", 2000};
```

Так как члены структуры описываются в блоке, то их имена принадлежат локальной области видимости внутри этого блока. Для доступа к элементу структуры вне блока определения структуры используется оператор точка (`'.'`). Например:

```
director.empno = 20;
strcpy(director.name, "John");
      /* нельзя выполнить присваивание,    */
      /* так как director.name – константа */
director.salary = 3000;
```

Членами структуры могут быть массивы и другие структуры. Для доступа к членам вложенной структуры оператор `'.'` используется столько раз, какова вложенность структуры. Например:

```
#include <stdio.h>

struct demo
{
```

```
    int a;
    struct inside
    {
      int b;
      int c;
    } b;
};

int main()
{
    struct demo c = {1, {2, 3}};
    printf("%d %d %d\n", c.a, c.b.b, c.b.c );  /* 1 2 3 */
    return 0;
}
```

При объявлении структура может содержать поля, тип которых является указателем на тип объявляемой структуры. Например:

```
struct node               /* вершина двоичного дерева */
{
  struct node *left;      /* левый потомок */
  struct node *right;     /* правый потомок */
};
```

Структуры одного типа можно присваивать друг другу. В этом случае оператор присваивания выполняет почленное копирование структур. Например:

```
struct emp boss;
boss = director;
```

Отметим, что структуры даже одного типа нельзя сравнивать между собой.

5.3. Объединения

Тип «объединение» описывает набор данных, которые называются элементами или членами объединения. Объединение позволяет хранить в одной и той же области памяти значения различных типов, которые соответствуют типам элементов объединения. Но в каждый данный момент времени в этой области памяти может храниться только одно значение. Отсюда следует, что длина памяти, распределяемой компилятором под объединение, равна наибольшей из длин членов этого объединения. Как и в случае структуры, члены объединения могут иметь любой тип, за исключением типов `void` и функция.

Объявляются объединения следующим образом:

```
union  num
{
  int n;
  double f;
};
```

где union – это ключевое слово; num – имя типа объединения; n, f – члены объединения.

Переменная типа объединение объявляется следующим образом:

```
union num d;
```

и также называется объединением. Объявление типа объединения и переменной, которая имеет этот тип, может быть объединено в одну инструкцию. Например:

```
union  num
{
  int  n;
  double  f;
} d;         /* d – переменная */
```

При объявлении переменной типа объединения ее можно инициализировать значением, которое должно иметь тип первого члена объединения. Например:

```
union num  d = {1};    /* правильно, d = 1 */
union num d = {1.0};  /* ошибка */
```

Как и в случае со структурами, для доступа к элементу объединения используется оператор '.'. Например:

```
union num e, g;
e.n = 1;
e.f = 1.1;
```

Объединения одного типа можно присваивать друг другу. В этом случае оператор присваивания выполняет почленное копирование объединений. Например,

```
union num e, g;
e = g;
```

Так же как и структуры, объединения нельзя сравнивать.

5.4. Битовые поля

Битовым полем называется член структуры или объединения, который определяет последовательность бит. Битовое поле может иметь один из следующих типов: int, unsigned или signed. При объявлении битового поля после его имени указывается длина поля в битах. Например:

```
struct  demo
{
  unsigned a1: 1;  /* 1 бит */
  signed b1: 3;    /* 3 бита */
  int c1: 6;       /* 6 бит */
};
```

Длина битового поля должна быть неотрицательным целым числом и не должна превышать длины базового типа данных битового поля.

Инициализируются битовые поля так же, как и обычные элементы структуры. Например:

```
struct  demo
{
  unsigned a1: 1;
  unsigned  a2: 2;
}  s = { 0, 1 };      /* a1 = 0, a2 = 1 */
```

Доступ к элементам битового поля осуществляется так же, как и доступ к обычным членам структуры. Например:

```
#include <stdio.h>

struct demo
{
  unsigned a: 1;
  signed   b: 3;
  int      c: 6;
};

int main ()
{
  struct demo s;
  s.a = 1;
  s.b = -1;
  s.c = -4;
  printf("s.a = %u\n", s.a);  /* печать: s.a = 1 */
  printf("s.b = %d\n", s.b);  /* печать: s.b = -1 */
  printf("s.c = %d\n", s.c);  /* печать: s.c = -4 */
  return 0;
}
```

Битовые поля могут использоваться в выражениях точно так же, как и переменные базового типа битового поля. Обычно битовые поля используются для установки различных флагов.

5.5. Передача структур в функции

Когда структура используется как параметр функции, то она передается в функцию по значению, как и принято в языке программирования Си. Например:

```
struct emp
{
  int empno;
  char name[20];
  double salary;
};

void print(struct emp s)
{
  printf("%d %s %f\n", s.empno, s.name, s.salary);
}
```

Для того чтобы функция могла изменить значения членов структуры, она должна получить указатель на эту структуру. Для доступа к членам структуры через указатели используется оператор `'->'`. Например:

```
void init_emp(emp *ps, int en, char *nm, double sal)
{
  ps->empno = en;
  strcpy(ps->name, nm);
  ps->salary = sal;
}
```

Аналогично передаются в функции и объединения. Например, передадим в функцию указатель на объединение.

```
union  num
{
  int  n;
  double  f;
} d;

void  print(char  c, num  *n)
{
  switch(c)
  {
  case 'i':
    printf("n = %d\n", n->n);
    break;
  case'f':
    printf("f = %f\n", n->f);
    break;
  default:
    printf("Unknown type.\n");
```

```
}
```

5.6. Задачи для самостоятельного решения

1. Для трехзначной логики, значения которой задаются следующим перечислением

```
enum triple = {lie, truth, unknown}
```

реализовать логические функции:

```
triple not(triple);           /* отрицание */
triple and(triple, triple);   /* логическое И */
triple or(triplet, triple);   /* логическое ИЛИ */
```

заданные следующими таблицами истинности:

x	not x
lie	truth
truth	lie
unknown	unknown

x	y	x and y	x or y
lie	lie	lie	lie
lie	truth	lie	truth
lie	unknown	lie	unknown
truth	lie	lie	truth
truth	truth	truth	truth
truth	unknown	unknown	truth
unknown	lie	lie	unknown
unknown	truth	unknown	truth
unknown	unknown	unknown	unknown

Проверить работоспособность реализованных логических функций путем построения в функции `main` их таблиц истинности.

2. Написать для работы со структурой типа:

```
struct array_stack      /* стек на массиве */
{
  int size;  /* размерность массива */
  int* p;    /* указатель на массив */
  int top;   /* верхушка стека */
};
```

следующие функции:

```
  /* инициализация стека */
void init(struct array_stack* s, int size);
  /* разрушение стека */
```

```
void destruct(struct array_stack* s);
  /* втолкнуть элемент в стек */
void push(struct array_stack* s, int n);
  /* вытолкнуть элемент из стека */
int pop(struct array_stack* s);
int is_empty(struct array_stack* s);  /* пустой стек? */
int is_fool(struct array_stack* s);   /* полный стек? */
```

Реализацию всех функций для работы со стеком поместить в один файл `stack.c`, а их прототипы в заголовочный файл `stack.h`.

3. Написать для работы со структурой типа:

```
struct list          /* список */
{
  struct node        /* элемент списка */
  {
    int item;        /* данные */
    struct node* p;  /* указатель на следующий элемент */
  };
    /* указатель на первый элемент в списке */
  struct node  *head;
};
```

следующие функции:

```
void destruct(struct list* l);     /* разрушить список */
    /* включить элемент в список */
void ins(struct list* l, int n);
int del(struct list* l);  /* исключить элемент из списка */
int is_empty(struct list* l);      /* пустой список? */
```

Реализацию всех функций для работы со списком поместить в один файл `list.c`, а их прототипы в заголовочный файл `list.h`.

4. Написать программу, которая создает массив структур типа:

```
struct student
{
  char  name[10];   /* фамилия студента */
  int  num;         /* номер группы */
  double  grade;    /* средний балл */
};
```

Размерность массива и данные, хранящиеся в массиве, вводятся с консоли. После создания массива программа позволяет пользователю выполнить над массивом структур следующие действия:

а) Отсортировать массив по фамилиям студентов. Вывести отсортированный массив на консоль.

b) Найти в массиве запись о студенте по его фамилии. Фамилия студента вводится с консоли, результат поиска также вывести на консоль.
c) Отсортировать массив по группам, а внутри одной группы по фамилиям студентов. Вывести отсортированный массив на консоль.
d) Вывести на консоль отчет о среднем балле студентов в каждой группе.

Для выполнения запросов разработать меню, которое выводится на консоль после обработки каждого запроса. Выход из программы выполняется по требованию пользователя. Для сортировки массива и поиска требуемой записи в массиве использовать стандартные функции сортировки и поиска.

5.7. Дополнительные задачи

1. Написать для работы со структурой типа:

```
struct array_queue    /* кольцевая очередь на массиве */
{
  int  size;     /* размерность массива */
  int*  p;       /* указатель на массив */
  int  head;     /* индекс первого занятого элемента */
  int  n;        /* количество элементов в очереди */
};
```

следующие функции:

```
  /* инициализация очереди */
void init(struct array_queue* q, int size);
  /* разрушить очередь */
void destruct(struct array_queue* q);
  /* втолкнуть элемент в очередь */
void push(struct array_queue* q, int n);
  /* удалить элемент из очереди */
int del(struct array_queue* q);
int is_empty(struct array_queue* q);  /* пустая очередь? */
int is_fool(struct array_queue* q);   /* полная очередь? */
```

Реализацию всех функций для работы с очередью поместить в один файл `queue.c`, а их прототипы в заголовочный файл `queue.h`.

2. Написать для работы со структурой типа:

```
struct deque    /* дек – двустороння очередь */
{
  struct node    /* элемент дека */
  {
  int item;          /* данные */
```

```
  struct node* next; /* указатель на следующий элемент */
  struct node* prev; /* указатель на предыдущий элемент */
  };
  /* указатели на первый и последний элементы в списке */
 struct node  *head, *tail;
};
```

следующие функции:

```
void  destruct(struct list* l);    /* разрушить дек */
  /* включить элемент в голову дека */
void  ins_head(struct list* l, int n);
  /* включить элемент в хвост дека */
void  ins_tail(struct list* l, int n);
  /* исключить элемент из головы дека */
int  del_head(struct list* l);
  /* исключить элемент из хвоста дека */
int  del_tail(struct list* l);
int  is_empty(struct list* l);    /* пустой дек? */
```

Реализацию всех функций для работы с деком поместить в один файл `deque.c`, а их прототипы в заголовочный файл `deque.h`.

5.8. Задачи для индивидуальной работы

1. ***Железнодорожный состав.*** На железнодорожном пути находится состав из некоторого количества белых и черных вагонов. Используя тупик, сформировать новый состав, в котором черные и белые вагоны строго чередуются. Для решения использовать структуру данных «стек».

2. ***Корректировка текста.*** Задан набор предложений текста. Признаком конца предложения служит один из следующих символов: `'.'`, `'!'`, `'?'`. Слова в предложении разделены одним из следующих символов: пробел, `','`, `';'`, `':'`. Скорректировать текст, используя следующие правила. Если в тексте встретился символ `'*'`, то предшествующий ему символ должен быть удален. Если в тексте встретился символ `'@'`, то предшествующее ему слово должно быть удалено. Если в тексте встретился символ `'&'`, то предшествующее ему предложение в тексте должно быть удалено. Для осуществления корректировки текста использовать структуру данных «стек».

3. ***Лабиринт.*** Лабиринт задается квадратной матрицей 10-го порядка, состоящей из 1 и 0. Клетка считается проходимой, если она содержит 0, и непроходимой, если она содержит 1. Начальное положение путника задается координатами одной из проходимых клеток. Путник может перемещаться из одной проходимой клетки в другую, если они имеют общую сторону. Выход считается найденным, если путник попадает в одну из

граничных клеток. Найти выход из лабиринта, используя стратегию «держусь за стенку».

4. ***Болото.*** План местности, разбитой на квадраты, задан матрицей размером *M*N*. Местность представляет собой болото с имеющимися на нем островками твердой земли. Островки определяются наличием единиц в соответствующих клетках матрицы и представляют собой конфигурации, не соприкасающиеся ни по вертикали, ни по горизонтали, ни по диагонали. Остальные клетки матрицы заполнены нулями. Островки расположены друг от друга на различных расстояниях. Расстояние между островками определяется количеством клеток с нулями между ними по горизонтали или по вертикали. Путник может пересечь болото, перебрасывая между островками доски. При расстоянии между островками в одну клетку путнику нужна одна доска, при расстоянии в две клетки – 2 доски и т.д. Определить, какое минимальное количество досок путнику необходимо взять с собой, чтобы пересечь болото, учитывая, что одна доска может использоваться только один раз. Начальной клеткой является любая островная клетка первой строки матрицы, конечная должна быть любой островной клеткой последней строки матрицы. В решении использовать структуру данных «стек».

5. ***Грядки.*** Садовый участок разбит на квадратные клетки со стороной 1 м и имеет размер M*N $м^2$. На участке вскопаны грядки, представляющие собой прямоугольные конфигурации, не соприкасающиеся ни по вертикали, ни по горизонтали. Грядки, совпадающие по диагонали, считаются разными грядками. Определить общее количество грядок и количество квадратных грядок. В решении использовать структуру данных «очередь».

6. ***Жидкокристаллический робот.*** В результате применения жидкого азота киборг рассыпался на *K* осколков различной массы. Считая, что площадь, где рассыпались осколки, представляется матрицей размером *N*M*, а вес осколка и его местоположение определяются элементом матрицы, найти, за какое количество единиц времени произойдет восстановление робота. Процесс восстановления робота осуществляется следующим образом. В каждую единицу времени осколки наименьшей массы исчезают, передавая свою массу ближайшему к нему осколку. Ближайшие осколки ищутся в полном окаймлении данного осколка. Если их будет несколько, то выбирается осколок наименьшей массы. В решении использовать структуру данных «список».

7. ***Сапер.*** Имеется план местности, разбитой на квадраты, заданный матрицей размером *M*N*. Местность представляет собой минное поле, где в некоторых местах находятся мины. Наличие мины определяется

наличием единицы в соответствующей клетке матрицы, отсутствие – нулем. Саперу для разминирования необходимо предварительно рассчитать количество и массу используемых подрывных зарядов. Один подрывной заряд способен воздействовать на группу мин, расположенных только в смежных клетках (участок) по вертикали, горизонтали и диагонали, при этом его масса определяется по следующему правилу: масса = (количество мин на участке + 3)/4. В решении использовать структуру данных «очередь».

8. ***Кладоискатель.*** Имеется план местности, разбитой на квадраты, заданный матрицей размером *M*N*. Местность представляет собой густой лес с протоптанными тропинками. Тропинки определяются наличием нуля в соответствующей клетке матрицы, участок леса определяется наличием единицы. В клетке с номером ($X_к$,$Y_к$) зарыт клад. В одной из граничных клеток, где начинается тропинка, находится кладоискатель. Необходимо так проложить путь к кладу, чтобы последовательно взять ряд необходимых предметов, которые в клетках матрицы обозначены как 2, 3, 4, 5. Клетка, соответствующая тропинке, всегда соседствует с одной из таких клеток. Клеток, содержащих значения 2, 3, 4, 5, в матрице может быть несколько. В решении использовать структуру данных «очередь».

9. ***Абоненты.*** Составить программу обслуживания системой заявок абонентов в течение длительного периода времени. Каждый абонент имеет возможность послать в систему любое количество заявок на обслуживание, которые помещаются в очередь по мере их поступления. Структура заявки содержит следующие поля: код абонента, номер заявки, содержание заявки. Первым обслуживается абонент с наибольшим количеством заявок в очереди. Для хранения заявок использовать структуру данных «линейный список».

10. ***Нарушители.*** Составить программу, осуществляющую учет нарушителей правил дорожного движения в ГАИ. Для каждого водителя в системе хранится список нарушений. Для каждого нарушения указывается дата, вид нарушения и размер штрафа. При оплате всех штрафов информация о водителе удаляется из системы. Программа осуществляет внесение информации о совершенном нарушении в систему, удаление его при поступлении сведений об оплате штрафа, выдает сведения обо всех нарушениях, совершенных указанным водителем, выдает список нарушений за указанный промежуток времени. Для хранения информации в системе используется структура данных «линейный список», при этом формируется список водителей, а для каждого водителя создается свой список нарушений.

11. ***Автосервис.*** Написать программу для автосервиса, выполняющую учет ремонтных работ автомобилей. При записи на обслуживание заполняется заявка, в которой указываются фамилия владельца, марка автомобиля, вид работы, стоимость ремонта. Обслуживание выполняется в порядке поступления заявок. После выполнения работы распечатывается квитанция. Программа осуществляет внесение заявок в систему, удаление сведений после выполнения ремонта, получение квитанции о выполненном ремонте, просмотр списка заявок, поиск нужных заявок по всем полям. Для хранения информации в системе используется структура данных «линейный двунаправленный список».

12. ***Файлы.*** Создать структуру данных «кольцевой список», каждый элемент которой содержит два поля: одно информационное и одно ссылочное, указывающее соответственно на последующий элемент списка. Информационное поле содержит сведения: имя файла, дату создания, количество обращений к файлу. Написать процедуры инициализации списка, пошагового просмотра элементов списка, добавления нового элемента в список, изменение величины «количества обращений» при поступлении соответствующей информации, удаления файлов из списка, дата создания которых меньше заданной, выборку сведений о файлах с наибольшим количеством обращений и наиболее ранней датой создания.

13. ***Книги.*** Создать структуру данных «кольцевой двунаправленный список». Каждый элемент списка содержит следующие сведения: номер УДК, фамилию и инициалы автора, название и количество экземпляров данной книги в библиотеке. Написать процедуры инициализации списка, просмотра элементов списка в обоих направлениях, добавления нового элемента в список перед указанным элементом и после указанного элемента, корректировки количества экземпляров указанной книги, удаления элемента из списка перед указанным и после указанного элементов.

Лабораторная работа 6

Тема. Файлы и потоки. Работа с бинарными файлами. Работа с текстовыми файлами.

6.1. Файлы и потоки

Файлом называют последовательность байтов, хранящихся на внешнем носителе информации. Под доступом к файлу понимают запись и чтение данных из файла. Потоком называется логический интерфейс (программа), который обеспечивает доступ к файлу. Прежде чем использовать поток для доступа к файлу, его необходимо соединить с этим файлом, т. е. обеспечить поток информацией о файле. Эта информация хранится в структуре типа `FILE`. Потому считается, что поток имеет тип `FILE*`, т. е., является указателем на файл. Когда поток соединяют с файлом, то говорят, что файл открывают. Когда поток отсоединяют от файла, то говорят, что файл закрывают.

Каждый поток может работать в двух режимах: текстовом и бинарном. Режим работы потока задается при его соединении с файлом.

В текстовом режиме поток записывает и читает из файла текстовые строки, которые заканчиваются символом `'\n'` и могут содержать символ `'\t'`. По стандарту поток должен обеспечивать обработку строк длиной не менее 254 символа, включая символ `'\n'`. Стандартом допускается, что при чтении и записи данных текстовым потоком может происходить их преобразование.

В бинарном режиме поток записывает и читает данные из файла в том виде, в котором они хранятся в оперативной памяти.

В следующих параграфах будут описаны стандартные функции для работы с файлами, используемые в языке Си. Прототипы этих функций находятся в заголовочном файле `stdio.h`.

6.2. Соединение и отсоединение потока от файла. Перенаправление потока

Для соединения потока с файлом используется функция

```
FILE* fopen(const char* filename, const char* mode);
```

которая открывает файл, имя которого задано параметром `filename`, в режиме, заданном параметром `mode`. В случае успешного завершения функция `fopen` возвращает указатель на поток, а в случае неудачи – `NULL`.

Параметр `mode` может принимать следующие значения:

"r" – чтение в текстовом режиме;

"w" – запись в текстовом режиме;

"a" – присоединение в текстовом режиме;

"rb" – чтение в бинарном режиме;

"wb" – запись в бинарном режиме;

"ab" – присоединение в бинарном режиме;

"r+" или "w+", или "a+" – чтение и запись в текстовом режиме;

"r+b" или "w+b", или "a+b" – чтение и запись в бинарном режиме;

"rb+" или "wb+", или "ab+" – чтение и запись в бинарном режиме.

При открытии файла в режимах "r", "rb", "r+", "r+b" его индикатор позиции устанавливается на начало файла. В случае, если открывается несуществующий файл, то функция fopen заканчивается неудачей.

При открытии файла в режимах "w", "wb", "w+", "w+b" создается новый файл. Если файл с заданным именем существует, то его содержимое стирается, а индикатор позиции устанавливается на начало файла.

При открытии файла в режимах "a", "ab", "a+", "a+b" создается новый файл. Если файл с заданным именем существует, то он открывается, и индикатор позиции устанавливается на конец файла.

Следует учитывать, что если текстовый файл открывается в режиме чтения и записи, то базовая операционная система может открыть его в бинарном режиме. Максимальное количество файлов, которые можно открыть одновременно, задается переменной FOPEN_MAX. Максимальная длина имени файла задается переменной FILENAME_MAX.

Для отсоединения потока от файла используется функция

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

которая закрывает файл, при этом освобождая все буферы потока. При успешном завершении функция возвращает 0, а в случае неудачи – EOF.

Для перенаправления потока используется функция

```
FILE* freopen(const char* filename, const char* mode,
  FILE* stream);
```

которая закрывает файл, соединенный с потоком stream, и соединяет с этим потоком файл filename в режиме mode. В случае успеха функция возвращает указатель на поток, а в случае неудачи – NULL. Параметр mode принимает те же значения, что и в функции fopen.

Поясним подробнее разницу между текстовым и бинарным режимами работы потока. Как в текстовом, так и в бинарном режиме можно ис-

пользовать все функции для доступа к файлу. При работе в бинарном режиме поток записывает на диск и считывает с диска точные копии данных, переданные функциями записи данных на диск и требуемые функциями чтения данных с диска соответственно. Работа потока в текстовом режиме отличается от работы потока в бинарном режиме тремя моментами. Во-первых, в этом случае символ `CTRL+Z` интерпретируется как конец файла. Во-вторых, при записи в текстовый поток из комбинации символов `'\r'` (carriage return, возврат каретки) и `'\n'` (new line, новая строка) в файл записывается только символ `'\n'`, а при чтении из текстового потока символ `'\n'` преобразуется в комбинацию символов `"\r\n"`. В-третьих, так как при записи в текстовый поток может происходить преобразование количества и представления символов, то для получения требуемой позиции в файле нужно использовать только функции `fgetpos` и `ftell`.

6.3. Работа с индикаторами ошибки, позиции и конца файла

1. ***Работа с индикатором ошибки.*** С каждым потоком связан индикатор ошибки, который находится в установленном положении, если в потоке, связанном с файлом, произошла ошибка. В противном случае индикатор ошибки находится в сброшенном состоянии. Для работы с индикатором ошибки используются функции `ferror` и `clearerr`.

Функция

```
int ferror(FILE* stream);
```

возвращает ненулевое значение, если индикатор ошибки установлен, в противном случае функция возвращает `0`.

Функция

```
void clearerr(FILE* strem);
```

сбрасывает индикаторы ошибки и конца файла для потока `stream`.

2. ***Работа с индикатором конца файла.*** Структура `FILE` содержит индикатор конца файла, который устанавливается в ненулевое значение функцией чтения из файла при достижении этой функцией конца файла. Состояние конца файла читается функцией

```
int feof(FILE* stream);
```

которая возвращает ненулевое значение, если индикатор конца файла установлен, в противном случае функция возвращает `0`.

3. ***Работа с индикатором позиции.*** Для каждого файла после его открытия определяется индикатор позиции, который указывает на смещение от начала файла в байтах. Тип индикатора позиции определяется как

```
typedef long fpos_t;
```

Для работы с индикатором позиции предназначены следующие функции.

Функция

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

устанавливает индикатор позиции на начало файла, связанного с потоком `stream`. При этом сбрасывается индикатор ошибки и конца файла.

Функция

```
int fseek(FILE* stream, long offset, int mode);
```

сдвигает индикатор позиции файла на `offset` байтов. В случае успешного завершения функция возвращает `0`, в противном случае – ненулевое значение. Параметр `mode` указывает на режим сдвига и может принимать следующие значения:

- `SEEK_SET` – смещение от начала файла;
- `SEEK_CUR` – смещение от текущей позиции;
- `SEEK_END` – смещение от конца файла.

При работе с текстовым потоком должны использоваться только следующие комбинации значений параметров:

- `mode = SEEK_SET`, `offset = 0` или `offset` равно значению, возвращаемому функцией `ftell`;
- `mode = SEEK_CUR`, `offset = 0`;
- `mode = SEEK_END`, `offset = 0`.

Функция

```
int fsetpos(FILE* stream, const fops_t *pos);
```

устанавливает индикатор позиции файла `stream` в позицию, на которую указывает параметр `pos`. Индикатор конца файла сбрасывается. В случае успеха функция возвращает `0`, а в случае неудачи возвращает ненулевое значение и устанавливает переменную `errno`.

Функция

```
long ftell(FILE* stream);
```

в случае успешного завершения возвращает текущую позицию файла `stream`, а в случае неудачи – возвращает значение `1L` и устанавливает значение переменной `errno`. Для бинарного потока позиция равна сме-

щению в байтах от начала файла, а в случае текстового потока – значению, которое может использоваться функцией `fseek`.

Функция

```
int fgetpos(FILE* stream, fops_t* pos);
```

записывает текущую позицию файла `stream` по адресу `pos`. В случае успеха функция возвращает `0`, а в случае неудачи – ненулевое значение и устанавливает значение переменной `errno`.

6.4. Блочный ввод-вывод

Блоком называется область оперативной памяти, содержимое которой записывается в файл. Ввод-вывод блоками используется бинарными потоками. Для записи блока в файл используется функция

```
size_t fwrite(const void* ptr, size_t size, size_t nitems,
  FILE* stream);
```

которая записывает содержимое блока памяти, на который указывает `ptr`, в файл `stream`. Длина записываемого блока определяется как произведение `size*nitems`. Функция возвращает число записанных единиц памяти. В случае удачи это число должно быть равно `nitems`. Для чтения блока из файла используется функция

```
size_t fread(const void* ptr, size_t size, size_t nitems,
  FILE* stream);
```

параметры которой имеют тот же смысл, что и в функции `fwrite`.

После работы функций `fwrite` и `fread` индикатор позиции сдвигается на количество записанных или прочитанных байтов соответственно.

В следующей программе создается бинарный файл.

```
/* создание бинарного файла */
#include <stdio.h>

struct emp
{
  int code;
  char name[20];
  double salary;
};

int main()
{
  FILE* out;        /* выходной поток */
  struct emp s;     /* для записей файла */
  s.salary = 0.0;   /* для обработки плавающих чисел */

  /* открываем выходной поток в бинарном режиме */
```

```
  if(!(out = fopen("C:\\employee.bin", "wb")))
  {
    printf("Open file failed.\n");
    return 1;
  }

  printf("Input code, name and salary.\n");
  printf("Press Ctrl+z to exit.\n");
  printf(">");

  /* вводим первую запись с консоли */
  scanf("%d%s%lf", &s.code, &s.name, &s.salary);
  while (!feof(stdin))
  {
    /* пишем запись в файл */
    fwrite(&s, sizeof(struct emp), 1, out);
    printf(">");
    /* вводим следующие записи с консоли */
    scanf("%d%s%lf", &s.code, &s.name, &s.salary);
  }
  /* закрываем выходной поток */
  fclose(out);
  return 0;
}
```

В следующей программе выполняется чтение записей из бинарного файла.

```
/* чтение бинарного файла */
#include <stdio.h>

struct emp
{
  int code;
  char name[20];
  double salary;
};

int main()
{
  FILE* in;      /* выходной поток */
  struct emp s;  /* для записей файла */
  unsigned i;    /* номер записи */

  /* открываем входной поток в бинарном режиме */
  if(!(in = fopen("C:\\employee.bin", "rb")))
  {
    printf("Open file failed.\n");
    return 1;
  }
```

```
  printf("Press Ctrl+z to exit.\n");

  /* читаем индекс */
  printf("Input an index: ");
  scanf("%u", &i);
  while (!feof(stdin))
  {

    /* устанавливает указатель на нужную запись */
    fseek(in, i*sizeof(struct emp), SEEK_SET);
    /* читаем запись из файла */
    if(!fread(&s, sizeof(struct emp), 1, in))
    {
      printf("The wrong index.\n");
      printf("Input an index: ");
      scanf("%u", &i);
      continue;
    }

    /* выводим запись на консоль */
    printf("\tcode = %d name = %s sal = %f\n",
      s.code, s.name, s.salary);
    /* читаем индекс */
    printf("Input an index: ");
    scanf("%u", &i);
  }

  /* закрываем входной поток */
  fclose(in);
  return 0;
}
```

6.5. Символьный ввод-вывод

Символьный ввод-вывод используется с текстовыми потоками.

1. ***Ввод-вывод символов.*** Для записи и чтения символов из текстового файла используются функции `fputc`, `putc`, `fgetc`, `getc`.

Функция

```
int fputc(int c, FILE* stream);
```

записывает символ, заданный параметром `c`, в поток `stream` и продвигает индикатор позиции на следующий символ. В случае успеха функция возвращает символ `c`, а в случае неудачи – `EOF` и устанавливает индикатор ошибки.

Функция

```
int putc(int c, FILE* stream);
```

работает так же, как и функция fputc, но может быть реализована как макрокоманда.

Функция

```
int fgetc(FILE* stream);
```

читает символ из потока stream и продвигает индикатор позиции на следующий символ. В случае успешного завершения функция возвращает прочитанный символ. В случае достижения конца файла функция возвращает EOF и устанавливает индикатор конца файла. В случае ошибки функция возвращает EOF и устанавливает индикатор ошибки.

Функция

```
int getc(FILE* stream);
```

работает так же, как и функция fgetc, но может быть реализована как макрокоманда.

Функция

```
int ungetc(int c, FILE* stream);
```

записывает символ c в поток stream. Функции fseek, fsetpos и rewind игнорируют такие символы. Доступ к записанным символам выполняется по правилу FIFO. В случае успеха функция возвращает записанный символ, а в случае неудачи – EOF.

2. *Ввод-вывод строк.* Для записи и чтения строк из символьного потока используются символы fpus и fgets.

Функция

```
int fputs(const char* str, FILE* stream);
```

записывает строку str в файл stream, не включая завершающий нулевой байт. В случае успешного завершения функция возвращает ненулевое число, а в случае неудачи – EOF.

Функция

```
int fgets(char* str, int n, FILE* stream);
```

читает строку из потока stream в строку str. Останавливается функция в случае, если прочитан (n-1) символ, или встретился символ '\n', или обнаружен конец файла. В любом из этих случаев в конец строки помещается символ '\n'. В случае успеха функция возвращает указатель str, а в случае неудачи – NULL. Строка str не изменяется, если не прочитан ни один символ и обнаружен конец файла.

3. *Форматированный ввод-вывод.* Для форматированного ввода-вывода в текстовые файлы используются функции fscanf и fprintf.

Функция

```
int fprintf(FILE* stream, const char* format, ...);
```

выполняет вывод в файл `stream` в соответствии с форматной строкой `format`. Работает эта функция так же, как и функция форматирования строк sprintf, которая была рассмотрена в лабораторной работе 4.

Функция

```
int fscanf(FILE* stream, const char* format, ...);
```

выполняет ввод из файла `stream` текста в соответствии с форматной строкой `format`. Работает эта функция так же, как и функция форматирования строк `sscanf`, которая была рассмотрена в лабораторной работе 4.

В следующей программе создается текстовый файл.

```
/* создание текстового файла */
#include <stdio.h>

int main()
{
  int code;
  char name[80];
  double salary;

  FILE* out;  /* выходной поток */
  /* открываем выходной поток в текстовом режиме */
  if(!(out = fopen("C:\\employee.txt", "w")))
  {
    printf("Open file failed.\n");
    return 1;
  }
  printf("Input code, name and salary.\n");
  printf("Press Ctrl+z to exit.\n");
  printf(">");

  /* вводим первую запись с консоли */
  scanf("%d%s%lf", &code, &name, &salary);
  while (!feof(stdin))
  {
    /* пишем запись в файл */
    fprintf(out, "%d %s %f ", code, name, salary);
    printf(">");
    /* вводим следующие записи с консоли */
    scanf("%d%s%lf", &code, &name, &salary);
  }

  /* закрываем выходной поток */
```

```
  fclose(out);
  return 0;
}
```

В следующей программе читается текстовый файл.

```
/* чтение текстового файла */
#include <stdio.h>

int main()
{
  int code;
  char name[80];
  double salary;

  FILE* in;    /* входной поток */
  /* открываем входной поток в текстовом режиме */
  if(!(in = fopen("C:\\employee.txt", "r")))
  {
    printf("Open file failed.\n");
    return 1;
  }

  /* читаем первую запись */
  fscanf(in, "%d%s%lf", &code, name, &salary);
  while (!feof(in))
  {
    /* выводим запись на консоль */
    printf("code = %d name = %s sal = %f\n",
           code, name, salary);
    /* читаем следующие записи */
    fscanf(in, "%d%s%lf", &code, name, &salary);
  }

  /* закрываем входной поток */
  fclose(in);
  return 0;
}
```

6.6. Работа с буферами

Буфером называется область оперативной памяти, используемая потоком для временного хранения данных из файла. Для работы с буферами используются функции `setvbuf`, `setbuf`, `fflush`.

Функция

```
int setvbuf(FILE* stream, char* buffer, int mode,
  size_t size);
```

определяет буфер ввода-вывода и режим работы с ним для потока stream. Вызывается эта функция после открытия файла, но перед доступом к нему. При успешном завершении функция возвращает значение 0, а в случае неудачи – ненулевое значение.

Параметр buffer указывает на блок памяти для буфера. Если этот параметр равен NULL, то функция setvbuf использует функцию malloc для захвата памяти под буфер.

Параметр mode определяет режим работы с буфером и может принимать следующие значения:

- _IOFBF – вывод данных из буфера во внешнюю память выполняется только при полной загрузке буфера или при закрытии файла;
- _IOLBF – вывод данных из буфера во внешнюю память выполняется при записи в буфер символа '\n';
- _IONBF – нет буферизации, в этом случае параметры size и buffer игнорируются.

Параметр size определяет длину буфера в байтах.

Функция

```
int setbuf(FILE* stream, char* buffer);
```

вызывает функцию setvbuf. Причем если значение параметра buffer не равно NULL, то функция setvbuf вызывается следующим образом:

```
setvbuf(stream, buffer, _IOFBF, BUFSIZE);
```

где константа BUFSIZE задает длину буфера по умолчанию. Эта константа описана в заголовочном файле stdio.h. В противном случае функция setvbuf вызывается следующим образом:

```
setvbuf(stream, 0, _IOFBF, BUFSIZE);
```

Т.е. в этом случае буферизация не используется.

Функция

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

записывает данные из буфера потока stream в соединенный с этим потоком файл. В случае успешного завершения функция возвращает значение 0, а в случае неудачи – EOF. Если значение параметра stream равно NULL, то освобождаются буферы всех потоков, которые работают в режиме вывода.

В следующей программе создается текстовый файл с буфером.

```
/* создание файла с буфером */
```

```
#include <stdio.h>

int main()
{
  int code;
  char name[80];
  double salary = 0.0;

  FILE* out;  /* выходной поток */
  const unsigned size = 1024;  /* размер буфера */
  char buffer[size];      /* буфер потока */
  /* открываем выходной поток в текстовом режиме */
  if(!(out = fopen("C:\\employee.txt", "w")))
  {
    printf("Open file failed.\n");
    return 1;
  }

  /* устанавливаем буфер для потока */
  if(setvbuf(out, buffer, _IOFBF, size))
  {
    printf("Set buffer failed.\n");
    return 0;
  }
  printf("Input code, name and salary.\n");
  printf("Press Ctrl+z to exit.\n");
  printf(">");

  /* вводим первую запись с консоли */
  scanf("%d%s%lf", &code, &name, &salary);
  while (!feof(stdin))
  {
    /* пишем запись в файл */
    fprintf(out, "%d %s %f ", code, name, salary);
    printf(">");
    /* вводим следующие записи с консоли */
    scanf("%d%s%lf", &code, &name, &salary);
  }

  /* закрываем выходной поток */
  fclose(out);
  return 0;
}
```

В следующей программе создается текстовый файл без буфера.

```
/* создание файла без буфера */
#include <stdio.h>

int main()
```

```
{
  int code;
  char name[80];
  double salary;

  FILE* out;  /* выходной поток */
  /* открываем выходной поток в текстовом режиме */
  if(!(out = fopen("C:\\employee.txt", "w")))
  {
    printf("Open file failed.\n");
    return 1;
  }
  /* нет буферизации */
  setbuf(out, NULL);
  printf("Input code, name and salary.\n");
  printf("Press Ctrl+z to exit.\n");
  printf(">");

   /* вводим первую запись с консоли */
  scanf("%d%s%lf", &code, &name, &salary);
  while (!feof(stdin))
  {
    /* пишем запись в файл */
    fprintf(out, "%d %s %f ", code, name, salary);
    printf(">");
    /* вводим следующие записи с консоли */
    scanf("%d%s%lf", &code, &name, &salary);
  }

  /* закрываем выходной поток */
  fclose(out);
  return 0;
}
```

6.7. Стандартные потоки

Каждой программе предоставляются три стандартных потока, которые по умолчанию соединены с консолью. Указатели на эти потоки возвращают макрокоманды `stdin`, `stdout`, `stderr`. Для работы со стандартными потоками предназначены рассмотренные ранее функции `scanf`, `printf`, а также следующие функции:

```
int putchar(int  c);    /* вывод символа в stdout */
int getchar(void);      /* ввод символа из stdin */
int puts(const char* str);  /* вывод строки в stdout */
char* gets(char*  str);     /* ввод строки из stdin */
void perror(const char* str);  /* вывод сообщения об */
                                /* ошибке в stderr */
```

Работают эти функции так же, как и аналогичные функции для работы с файлами.

6.8. Служебные функции для работы с файлами

В этом параграфе перечислим служебные функции для работы с файлами, которые не входят ни в одну из вышеперечисленных категорий. К ним относятся функции `remove`, `rename`, `tmpfile`, `tmpname`.

Функция

```
int remove(const char* filename);
```

удаляет файл с именем `filename`. Если файл открыт, то работа функции зависит от реализации. В случае успешного завершения функция возвращает 0, а в случае неудачи – ненулевое значение.

В следующей программе показывается, как удалить файл.

```
/* удаление файла */
#include <stdio.h>

int main()
{
  if(remove("C:\\employee.bin"))
  {
    printf("There is no such a file.\n");
    return 1;
  }
  printf("The file was deleted.\n");
  return 0;
}
```

Функция

```
int rename(const char* old_filename, const char*
  new_filename);
```

переименовывает файл с именем `old_filename` в файл с именем `new_filename`. Если файл с именем `new_filename` уже существует, то работа функции зависит от реализации. В случае успешного завершения функция возвращает 0, а в случае неудачи – ненулевое значение.

В следующей программе показывается, как переименовать файл.

```
/* переименование файла */
#include <stdio.h>

int main()
{
  if(rename("C:\\employee.txt", "C:\\emp.txt"))
  {
    printf("There is no such a file.\n");
```

```
    return 1;
  }
  printf("The file was renamed.\n");
  return 0;
}
```

Функция

```
FILE* tmpfile(void);
```

создает временный файл в режиме "w+b". После закрытия потока файл удаляется. В случае успешного завершения функция возвращает указатель на файл, а в случае неудачи – NULL.

Функция

```
char* tmpnam(char* str);
```

возвращает имя для временного файла. Максимальное количество имен равно TMP_MAX, а максимальная длина имени равна L_tmpnam. Если значение параметра str равно NULL, то функция возвращает указатель на свою строку, в противном случае возвращается указатель str.

Следующая программа показывает пример использования временного файла.

```
/* использование временного файла */
#include <stdio.h>

int main()
{
  int code;
  char name[80];
  double salary = 0.0;

  FILE* temp;  /* временный файл */
  /* открываем временный файл */
  if(!(temp = tmpfile()))
  {
    printf("Create temp file failed.\n");
    return 1;
  }
  printf("Input code, name and salary.\n");
  printf("Press Ctrl+z to exit.\n");
  printf(">");

  /* вводим первую запись с консоли */
  scanf("%d%s%lf", &code, &name, &salary);
  while (!feof(stdin))
  {
    /* пишем запись в файл во временный файл */
```

```
    fprintf(temp, "%d %s %f ", code, name, salary);
    printf(">");
    /* вводим следующие записи с консоли */
    scanf("%d%s%lf", &code, &name, &salary);
  }

  /* устанавливаем индикатор позиции на начало файла */
  rewind(temp);
  printf("\nRead records from the temporary file.\n");
  /* читаем первую запись из временного файла */
  fscanf(temp, "%d%s%lf", &code, name, &salary);
  while (!feof(temp))
  {
    /* выводим запись на консоль */
    printf("code = %d name = %s sal = %f\n",
           code, name, salary);
    /* читаем следующие записи из временного файла */
    fscanf(temp, "%d%s%lf", &code, name, &salary);
  }

  /* закрываем временный файл */
  fclose(temp);
  return 0;
}
```

6.9. Задачи для самостоятельного решения

1. Написать программу, которая корректирует текстовый файл. Программа позволяет выполнить над текстовым файлом следующие функции:

a) Исключает из текста заданное слово. Слово, которое нужно исключить, вводится с консоли. Откорректированный текст выводится на консоль.
b) Вставляет в текст новое слово после заданного слова. Новое и заданное слова вводятся с консоли. Откорректированный текст выводится на консоль.
c) Заменяет в тексте старое слово на новое. Старое и новое слова вводятся с консоли. Откорректированный текст выводится на консоль.

Работу с функциями оформить в виде меню, которое выводится на консоль каждый раз после выполнения функции. Имя исходного текстового файла вводится с консоли. Выход из программы по требованию пользователя.

2. Заданы два бинарных файла с записями следующей структуры:

```
struct  Student
{
```

```
  long  num;        /* номер зачетки */
  char  name[10];   /* имя студента */
  int  group;       /* номер группы */
  double  grade;    /* средний балл */
};
```

Разработать программу, которая выполняет над заданными файлами алгебраические операции объединения ($\cup$), пересечения ($\cap$) и разности ($-$). Результат выполнения этих операций записывается в третий файл. Имена исходных и выходного файлов вводятся с консоли. Операции над файлами удовлетворяют следующим требованиям:

- операция объединения включает в результирующий файл только такие записи, которые находятся хотя бы в одном из двух входных файлах;
- операция пересечения включает в результирующий файл только такие записи, которые находятся одновременно в двух входных файлах;
- операция разности включает в результирующий файл только такие записи, которые находятся в первом файле и отсутствуют во втором файле.

Запросы на выполнение операций оформить в виде меню.

3. Разработать программу для сортировки бинарного файла, записи которого имеют структуру, описанную в задаче 2. Записи файла могут сортироваться в следующих порядках:

a) по возрастанию номеров зачеток;

b) по номерам групп, а внутри групп по фамилиям студентов.

Имя файла вводится с консоли. Запрос на вид сортировки оформить в виде меню. Перед выполнением сортировки вывести на консоль исходный файл. После завершения сортировки вывести на консоль отсортированный файл.

6.10. Дополнительные задачи

1. В заданном текстовом файле найти номер позиции, начиная с которого располагается заданный фрагмент текста.
2. В заданном текстовом файле найти и заменить по всему тексту один указанный фрагмент текста на другой.
3. Из заданного текстового файла скопировать в другой файл, начиная с заданной позиции, блок символов заданной длины.

4. Для заданного текстового файла найти повторяющийся фрагмент текста в этом файле.
5. Для заданного текстового файла выполнить шифрование хранящейся в нем информации путем выполнения операции “исключающее или” с 16(32)-разрядным ключом. Обеспечить возможность расшифровки файла.

6.11. Задачи для индивидуальной работы

1. ***Результаты соревнований.*** Задан текстовый файл, содержащий информацию о результатах соревнований спортсменов в троеборье: фамилия участника, результат в жиме, результат в прыжке, результат в беге на 100 м. За один просмотр файла получить фамилии трех лучших спортсменов по каждому из видов спорта и с указанием достигнутых ими результатов занести в текстовый файл.

2. ***Осадки.*** Задан текстовый файл, содержащий информацию о количестве осадков, выпавших в каждом месяце соответствующего года. Необходимо получить информацию:

- об общей сумме выпавших осадков в каждом году;
- о трех самых засушливых месяцах в каждом году;
- о выпавших поквартально количествах осадков в каждом году;
- о трех самых засушливых годах;
- о годе с самым засушливым летом.

3. ***Страны света.*** По имеющейся информации: название страны, часть света, занимаемая площадь, численность населения – создать бинарный файл и выполнить следующие действия:

- обновить сведения о количестве проживающего населения в указанной стране;
- по указанной части света вывести суммарное количество занимаемой площади и проживающего населения;
- определить страну с наименьшей плотностью населения;
- вывести список стран по указанной части света;
- вывести процентное распределение площадей, занимаемых странами;
- определить часть света, в которой проживает наибольшее количество человек;
- определить количество стран, входящих в каждую из частей света;
- определить часть света, в которой плотность населения максимальна;

- поменять в файле местами записи о странах с минимальным и максимальным населением.

Информация файла представлена в следующей таблице:

Название страны	Часть света	Площадь (в тыс. кв. км)	Население (в тыс. человек)
Китай	Азия	8128	1400
Япония	Азия	672	500
Англия	Европа	327	85
Франция	Европа	173	115
Канада	Америка	5123	130
Германия	Европа	180	121
США	Америка	4677	270

Все действия оформить в виде отдельных функций.

4. ***Зоопарк.*** По имеющейся информации: название животного, страна обитания, год поступления, стоимость – создать бинарный файл и выполнить следующие действия:

- обновить сведения о стоимости указанного животного;
- для указанного года определить, на какую сумму зоопарк приобрел животных;
- вывести информацию о самом дорогостоящем животном;
- получить процентное распределение затраченных сумм на покупку животных в разрезе стран обитания;
- определить, в каком году поступило наибольшее количество животных;
- определить страну, из которой поступило животных на самую большую сумму;
- определить, в каком году начал функционировать зоопарк;
- поменять местами в файле записи о животных из Африки;
- получить процентное распределение затраченных сумм на покупку животных по годам.

Информация файла представлена в следующей таблице:

Название животного	Страна обитания	Год поступления в зоопарк	Стоимость, у.е.
Волк	Россия	1996	100
Слон	Индия	1998	1500
Рысь	Россия	1996	200
Жираф	Африка	1998	1000
Медведь	Россия	1997	300

Тигр	Индия	1998	800
Носорог	Африка	1997	1200

Все действия оформить в виде отдельных функций.

5. *Компьютеры.* Создать бинарный файл, содержащий сведения об ассортименте имеющихся в продаже процессоров различных моделей и различных фирм и вывести его на экран. Структура записей: модель, фирма, частота, цена в у. е.

Написать программу, в результате выполнения которой:

- выдается список моделей процессоров со стоимостью от 80 до 100 у. е.;
- обновляется стоимость процессоров фирмы Intel на заданную величину;
- выдается список моделей процессоров с частотой > 1000 МГц и их общая стоимость;
- выдаются сведения о самой дешевой модели процессора;
- выдаются сведения о фирме, имеющей наименьшую суммарную цену выпускаемых ею процессоров;
- вычисляется процентное распределение общей стоимости процессоров в разрезе выпускающих их фирм;
- осуществляется упорядочение записей в бинарном файле по возрастанию стоимости процессоров;
- осуществляется удаление из файла информации о процессорах указанной модели.

Информация файла представлена в следующей таблице:

Название модели	Название фирмы	Частота, МГц	Цена, у.е.
Duron	AMD	700	40
Duron	AMD	750	45
Athlon	AMD	1000	80
Athlon	AMD	1200	90
Athlon	AMD	1500	100
Celeron	Intel	1000	90
P-4	Intel	1500	110
P-4	Intel	2000	150

Все действия оформить в виде отдельных функций.

6. *Соревнования.* При проведении соревнований каждому спортсмену предоставляется несколько попыток для выполнения упражнений. Четы-

рехзначный код спортсмена, начинающийся с цифры 9, и количество баллов, полученных за каждую попытку, заносятся в бинарный файл с целочисленным типом записи. Необходимо написать следующие процедуры: процедуру создания бинарного файла, записи которого формируются на основе аналогичной информации, хранящейся в текстовом файле; процедуру вывода информации, хранящейся в бинарном файле, причем код спортсмена и полученные им баллы должны быть выведены в одну строчку; процедуру формирования и вывода в новый текстовый файл среднего значения полученного спортсменом балла вместе с его кодом (при расчете среднего балла не учитываются максимальное и минимальное значения баллов, полученные спортсменом); процедуру, выполняющую обмен в файле минимального и максимального значений баллов для каждого спортсмена; процедуру, выводящую по указанному спортсмену его средний балл.

7. ***Банковская система.*** Реализовать систему, осуществляющую моделирование работы банковской системы по следующей схеме. При проведении банковской операции открытия нового счета для клиента заводится карточка, где указывается фамилия клиента, номер счета, заносимая сумма. При выполнении операций добавления или снятия денег со счета клиент указывает свою фамилию, после чего ему показывают список его счетов с имеющимся количеством денег на каждом счете и общую сумму денег. Клиент может снять или добавить деньги на указанный счет, а также снять сумму, не превышающую общей суммы. В последнем случае деньги снимаются последовательно, начиная со счета с максимальным количеством денег, затем со счета с максимальным количеством денег из оставшихся счетов и т. д. При попытке снятия суммы, превышающей общий итог, выдается соответствующее сообщение. Счет с нулевой суммой удаляется. Работа системы реализуется процедурами, выполняющими: добавление нового счета; снятие клиентом денег; добавление клиентом денег на указанный счет; вывод информации о наличии общей суммы денег у каждого из клиентов; закрытие счета. Информация хранится в бинарном файле.

8. ***Склад.*** Реализовать систему, осуществляющую моделирование работы склада по следующей схеме. Склад имеет определенное количество ячеек для хранения деталей различного типа. Емкость всех ячеек одинакова и задается в начале моделирования. Первоначальное состояние склада считывается из текстового файла. При поступлении новой детали на склад на нее заводится новая запись, в которой указывается название детали, количество поступивших деталей, номер ячейки хранения. Де-

таль такого же наименования может быть добавлена в уже имеющуюся ячейку или изъята из нее в указанных количествах. Если при добавлении детали будет превышена емкость ячейки, то превышаемое количество должно быть размещено в новой ячейке. Изъятие детали может осуществляться двумя способами: пропорциональным и последовательным. При пропорциональном изъятии заданное количество изымаемых деталей распределяется между ячейками, пропорционально хранящимся там количествам деталей. При последовательном снятии деталь сначала забирается из ячейки, хранящей максимальное количество деталей данного наименования, затем из ячейки с максимальным количеством деталей из оставшихся и т. д. При попытке изъятия количества деталей, превышающего общий итог, выдается соответствующее сообщение. При заполнении всех ячеек склада выдается соответствующее сообщение. Работа системы реализуется процедурами, выполняющими: создание модели склада, добавление новой детали; изъятие указанной детали по указанному способу; добавление детали имеющегося наименования; вывод информации о наличии общего количества хранящихся на складе деталей в разрезе наименований. Информация хранится в бинарном файле.

9. ***Комплекты.*** Реализовать систему, осуществляющую моделирование работы участка ОТК по следующей схеме. Участок ОТК хранит детали различных наименований, из которых осуществляется формирование комплектов. При поступлении новой детали на нее заводится карточка, где указывается наименование и количество поступления. При поступлении детали с уже имеющимся названием новое количество добавляется к уже имеющемуся количеству. На каждый комплект имеется карточка, в которой указывается название комплекта, название входящей в него детали, количество входящих в комплект деталей данного наименования. Отгрузка всегда осуществляется комплектами. Максимальное количество комплектов, которые можно отгрузить, определяется минимальным значением составляющей его детали, имеющейся в данный момент в секции ОТК. При отгрузке некоторого количества комплектов значения количеств хранящихся деталей должны быть уменьшены в пропорциональном к количеству входящих в комплект деталей отношении. Например, если в комплект К1 входит две детали Д1, то при отгрузке трех комплектов К1 количество деталей Д1 должно быть уменьшено на шесть деталей. Первоначальное состояние участка ОТК считывается из текстового файла. Работа системы реализуется процедурами, выполняющими: создание модели участка ОТК, добавление новой детали; добавление детали имеющегося наименования; отгрузка комплектов; вывод информации о наличии общего количества хранящихся на участке дета-

лей в разрезе наименований, упорядоченной по убыванию; вывод информации об имеющихся на участке комплектах с указанием количества. Информация хранится в бинарном файле.

10. ***Серии значений.*** Дан бинарный файл, содержащий серии значений показаний приборов – результатов проводимых экспериментов. Каждая серия значений отделяется от другой серии нулевым значением.

а) Определить:

- количество имеющихся значений показаний приборов в каждой серии;
- сумму значений показаний приборов в каждой серии;
- среднее значение показаний приборов в каждой серии;
- максимальное и минимальное значения показаний приборов для каждой серии;
- порядковый номер серии с максимальной суммой значений показаний приборов;
- среднее значение показаний приборов для каждой серии без учета максимального и минимального значений.

б) Отсортировать по возрастанию значения показаний приборов

- в каждой серии;
- в серии с указанным порядковым номером.

в) Поменять в каждой серии максимальное и минимальное значения показаний приборов местами.

г) Для серии с указанным порядковым номером осуществить пропорциональное уменьшение значений на указанную величину.

д) Найти выраженное в процентах распределение сумм значений по каждой серии показаний приборов к общей сумме всех значений.

11. ***Автопарк.*** Автопарк осуществляет обслуживание заказов на перевозку грузов, используя для этой цели свой парк автомашин и своих водителей. Водитель, выполнивший заказ, получает 20 % от стоимости перевозки. Для получения сведений о результатах работы автопарка создать бинарный файл, содержащий следующую информацию: дата перевозки, номер машины, фамилия водителя, масса перевезенного груза, километраж, стоимость перевозки.

Написать программу, в результате выполнения которой:

- осуществляется обновление, добавление и удаление информации в файле;
- по указанному водителю выводится информация о выполненных заказах за указанный период;

- по указанной машине определяется общий пробег и общая масса перевезенных грузов;
- по каждому водителю выводится общее количество поездок, общая масса перевезенных грузов, сумма заработанных денег;
- по водителю, выполнившему наименьшее количество поездок, рассчитывается процент заработанных им денег по отношению к общему количеству денег;
- по автомашине с наибольшим общим пробегом выводятся фамилии водителей, выполнявших на ней поездки.

Все действия оформить в виде отдельных функций.

12. ***Рыболовная флотилия.*** Флотилия рыболовных траулеров осуществляет поиск косяков рыбы и его отлов. С этой целью каждый траулер отправляется в рейс на некоторое количество суток и посещает ряд мелководных участков моря, называемых банками. Каждая банка имеет свое название. Выловленная рыба классифицируется своим названием, количеством и качеством. Для получения сведений о результатах работы рыболовной флотилии создать бинарный файл, содержащий следующую информацию: название траулера, дата выхода в море, дата возвращения, название посещенной банки, название, качество и количество выловленной рыбы.

Написать программу, в результате выполнения которой:

- осуществляется обновление, добавление и удаление информации в файле;
- по указанному траулеру выводится список выполненных рейсов за указанный период с выдачей сведений об общем количестве выловленной рыбы;
- по указанной банке определяется перечень названий и количество выловленной рыбы;
- по банке, на которой было выловлено максимальное количество рыбы низкого качества, выдаются сведения о датах рейсов и траулерах, ее посещавших;
- по траулеру, выловившему наибольшее количество рыбы, выдаются сведения о посещенных траулером банках с указанием дат выхода и возвращения.

Все действия оформить в виде отдельных функций.

13. ***Воздушный извозчик.*** Отряд грузовых вертолетов осуществляет доставку грузов и людей в высокогорном районе. Каждый вертолет обслуживается экипажем из трех пилотов, постоянно закрепленных за ним. Летчики получают по 5 % от стоимости обычного рейса (код – 0) и 10 %

от стоимости спецрейса (код – 1). Для получения сведений о результатах работы отряда грузовых вертолетов создать бинарный файл, содержащий следующую информацию: дата рейса, номер вертолета, код рейса, масса груза, количество перевезенных людей, длительность полета, стоимость рейса, дата последнего капитального ремонта, ресурс летного времени до следующего капитального ремонта.

Написать программу, в результате выполнения которой:

- осуществляется обновление, добавление и удаление информации в файле;
- по каждому вертолету выводится общее количество часов, которые они налетали после капитального ремонта, и ресурс летного времени до следующего капитального ремонта;
- по каждому вертолету формируется и выводится перечень выполненных рейсов с указанием общей массы перевезенных грузов и количества человек за указанный период;
- по всем вертолетам, выполнявшим спецрейсы, определяется общее количество рейсов, общая масса перевезенных грузов, общая сумма стоимости рейсов;
- по вертолету, выполнившему максимальное количество рейсов, выводятся сведения о количестве выполненных рейсов и количестве заработанных экипажем денег;
- по экипажу, заработавшему максимальное количество денег, определяется средняя продолжительность рейса и средний заработок за один рейс.

Все действия оформить в виде отдельных функций.

14. ***Музыкальный салон.*** Постоянно работающий музыкальный салон продает компакт-диски с записями определенных исполнителей, поступающие от различных компаний-производителей. Для получения сведений о результатах работы музыкального салона создать бинарный файл, содержащий следующую информацию: дата операции, код операции (поступление или продажа), код компакта, количество экземпляров, цена одного компакта, название музыкального произведения, фамилия автора, фамилия исполнителя.

Написать программу, в результате выполнения которой:

- осуществляется обновление, добавление и удаление информации в файле;
- по всем компактам формируются и выводятся сведения о количестве проданных и оставшихся компактов одного вида по убыванию разницы;

- по указанному компакт-диску выдаются сведения об общем количестве и общей стоимости компактов, проданных за указанный период;
- по компакт-диску, купленному максимальное количество раз, выводятся все сведения о нем и о содержащихся на нем музыкальных произведениях;
- по наиболее популярному исполнителю выводятся сведения о количестве проданных компактов с исполняемыми им произведениями;
- по каждому автору определяются сведения о количестве проданных компактов с его записями и сумме полученных денег.

Все действия оформить в виде отдельных функций.

15. ***Охотничье хозяйство.*** Система охотничьего хозяйства включает сеть пунктов приема шкурок пушных зверей (соболь, куница, песец, росомаха и т. д.). Пункты обслуживают свободных охотников, принимая от них шкурки зверей по разной цене в зависимости от вида и сорта меха. Впоследствии шкурки поставляются на различные меховые фабрики согласно заявке. Для получения сведений о результатах работы пунктов приема шкурок создать бинарный файл, содержащий следующую информацию: номер пункта, дата приема шкурок, фамилия охотника, год рождения, вид меха, сорт меха, количество принятых единиц меха, цена одной шкурки, номер меховой фабрики, дата отгрузки, количество отгруженных единиц.

Написать программу, в результате выполнения которой:

- осуществляется обновление, добавление и удаление информации в файле;
- по фабрике, получившей пушнины на максимальную сумму, выводится перечень номеров пунктов и список охотников;
- по каждой меховой фабрике формируется общий перечень и количество полученной пушнины по видам и сортам;
- по каждому пункту определяются общее количество принятой пушнины, в том числе отдельно первого сорта, и сумма заработанных денег за указанный период;
- по пункту, на котором было принято максимальное количество шкурок низкого качества, выводятся все сведения о датах приема и фамилиях охотников;
- по самому молодому охотнику выдается перечень и количество сданной им пушнины по видам и сортам.

Все действия оформить в виде отдельных функций.

16. ***Пушной аукцион.*** Управление звероводческими совхозами регулярно проводит пушные аукционы, куда звероводческие совхозы пред-

ставляют свою продукцию. Пушнина выставляется по определенной цене, но в результате аукциона она может быть продана по более высокой или низкой цене. Для получения сведений о результатах работы пушного аукциона создать бинарный файл, содержащий следующую информацию: номер звероферм, адрес звероферм, название меха, количество выставленных единиц, заявленная цена, количество проданных единиц, продажная цена, категория покупателя меха (меховая фабрика, ателье, частное лицо).

Написать программу, в результате выполнения которой:

- осуществляется обновление, добавление и удаление информации в файле;
- по звероферме, чей мех получил самую высокую цену, выводятся все сведения о ней;
- по каждой категории покупателей формируется общее количество купленных единиц меха, общая сумма уплаченных денег;
- по каждой звероферме рассчитывается прибыль от продажи пушнины;
- по всем звероферм, которые продали шкурки по цене выше средней аукционной цены, выдаются все сведения о них;
- по звероферме, получившей максимальную прибыль, формируются все сведения о ней, количестве проданной пушнины и величине прибыли.

Все действия оформить в виде отдельных функций.

17. ***Цветочная оранжерея*****.** Цветочная оранжерея выращивает различные виды цветов и продает на заказ составленные из них композиции. Каждая композиция имеет свое название и может состоять как из цветов одного вида, так и из цветов разного вида. Заказ обычно выполняется в течение нескольких дней. При выполнении заказа в течение суток дополнительно взимается плата в размере 25 %. При выполнении заказа в течение двух суток дополнительно взимается плата в размере 15 %. Для получения сведений о результатах работы цветочной оранжереи создать бинарный файл, содержащий следующую информацию: дата принятия заказа, дата выполнения заказа, название композиции, количество заказанных единиц, название входящего в композицию цветка, количество единиц, стоимость одного цветка.

Написать программу, в результате выполнения которой:

- осуществляется обновление, добавление и удаление информации в файле;

- по всем заказам рассчитывается сумма полученных денег за указанный период;
- по композиции, пользующейся максимальным спросом, выводятся все сведения о ней;
- по всем заказам формируются сведения о количестве выполненных заказов по срочности;
- по всем заказам определяются сведения о количестве использованных цветов по видам за указанный период;
- по всем заказам выдаются сведения о количестве проданных композиций и сумме полученных денег по видам композиций.

Все действия оформить в виде отдельных функций.

18. ***Парфюмерный базар.*** Постоянно работающий парфюмерный базар характеризуется участием в ней оптовых фирм-поставщиков и оптовых фирм-покупателей, при этом все сделки заключаются через специальных маклеров, имеющих доступ ко всем товарам. Для получения сведений о результатах работы парфюмерного базара создать бинарный файл, содержащий следующую информацию: дата сделки, вид сделки (поставка или продажа), название товара, цена единицы товара, количество единиц, фамилия маклера, оптовая фирма (поставщик или покупатель).

Написать программу, в результате выполнения которой:

- осуществляется обновление, добавление и удаление информации в файле;
- по каждому названию товара формируются сведения о проданном количестве и общей стоимости за указанный период;
- по каждому названию товара выдается перечень фирм-покупателей с указанием сведений о количестве единиц и стоимости купленного ими товара;
- по товару, пользующемуся наибольшим спросом, рассчитываются данные о количестве и стоимости проданного товара по каждой фирме-покупателю;
- по маклеру, совершившему максимальное количество сделок, выводятся сведения о нем и фирмах-поставщиках;
- по каждой фирме-поставщику вывадается список маклеров с указанием сведений о количестве и стоимости проданного ими товара по каждому маклеру.

Все действия оформить в виде отдельных функций.

19. ***Туристическое бюро.*** Бюро путешествий осуществляет обслуживание экскурсионных маршрутов, используя для этой цели свои автобу-

сы и экипажи, состоящие из трех человек, которые строго закреплены за своим автобусом. Экипаж, выполнивший заказ на обслуживание, получает 20 % от стоимости обслуживания. Стоимость билета договорная и зависит от длины туристической трассы и числа участников экскурсии. Для получения сведений о результатах работы туристического бюро создать бинарный файл, содержащий следующую информацию: номер автобуса, дата отбытия, дата прибытия, название маршрута, протяженность пути, количество перевезенных пассажиров, стоимость билета.

Написать программу, в результате выполнения которой:

- осуществляется обновление, добавление и удаление информации в файле;
- по указанному автобусу выводится перечень выполненных рейсов с указанием названий маршрутов за указанный период;
- по указанному автобусу рассчитывается общее количество поездок, количество перевезенных пассажиров и полученных денег;
- по каждому экипажу определяется количество начисленных денег за указанный период;
- по автобусу с наибольшим суммарным пробегом выводятся сведения о количестве перевезенных пассажиров и величине пробега.

Все действия оформить в виде отдельных функций.

20. ***Праздничная феерия.*** Магазин по продаже фруктов в предпраздничные дни формирует праздничные наборы из фруктов и орехов. Каждый набор имеет название. В наборы могут входить свежие фрукты, сухофрукты, засахаренные фрукты и орехи: миндальные, фундук, кешью и арахис. Заказ обычно выполняется в течение нескольких дней. При выполнении заказа в течение суток дополнительно взимается плата в размере 20 %. При выполнении заказа в течение двух суток дополнительно взимается плата в размере 10 %. Для получения сведений о результатах работы магазина создать бинарный файл, содержащий следующую информацию: дата принятия заказа, дата выполнения заказа, название набора, количество единиц, название ингредиента (входящего фрукта или орехов), вид, вес ингредиента, стоимость одного килограмма, заказчик;

Написать программу, в результате выполнения которой:

- осуществляется обновление, добавление и удаление информации в файле;
- по всем заказам определяется общее количество, общая стоимость и размер скидки за указанный период;
- по наборам, пользующимся максимальным спросом, выводятся все сведения о них;

- по всем заказам формируются и выводятся все сведения о количестве выполненных заказов по срочности;
- по всем заказам расситываются данные о количестве использованных ингредиентов по каждому виду;
- по всем заказам выдаются сведения о количестве проданных наборов и сумме полученных денег по видам наборов.

Все действия оформить в виде отдельных функций.

21. ***Реклама на ТВ.*** Фирмы рекламируют свои товары на телевидении в различных передачах и на различных каналах. Известна цена минуты рекламного времени в той или иной передаче. Для получения сведений о результатах рекламирования товаров на телевидении создать бинарный файл, содержащий следующую информацию: дата, название передачи, время выхода в эфир, название фирмы, рекламируемый товар, длительность рекламы (мин), стоимость единицы рекламного времени.

Написать программу, в результате выполнения которой:

- осуществляется обновление, добавление и удаление информации в файле;
- по рекламируемым товарам определяется общее количество минут рекламного времени по каждой передаче;
- по каждой фирме выдается перечень рекламируемых товаров с указанием общего количества рекламного времени и затраченных денег;
- по каждой передаче формируется перечень фирм с указанием общего количества минут и общей стоимости рекламы по каждой фирме за указанный период;
- по наиболее рекламируемому товару выводится перечень передач с указанием суммарного времени и стоимости рекламы по каждой передаче;
- по передаче, пользующейся среди рекламодателей максимальной популярностью, выдаются сведения о ней, общем количестве рекламного времени и сумме полученных денег.

Все действия оформить в виде отдельных функций.

22. ***Каменная сказка.*** Управление Ювелирторгом регулярно проводит ярмарки-продажи «Каменная сказка», где выставляются на продажу изделия ювелирных фирм. Изделия представляются по определенной цене, но в результате ярмарки они могут быть проданы по более высокой или низкой цене различным категориям покупателей. Для получения сведений о результатах работы ярмарки-продажи «Каменная сказка» создать бинарный файл, содержащий следующую информацию: название фирмы, название изделия, количество выставленных единиц, заявленная

цена, количество проданных единиц, аукционная цена, категория покупателя (учреждение, оптовик, частное лицо).

Написать программу, в результате выполнения которой:

- осуществляется обновление, добавление и удаление информации в файле;
- по каждой категории покупателей выводится общее количество купленных единиц, общая сумма уплаченных денег;
- по каждой фирме формируются сведения о выставленных и проданных изделиях;
- по всем фирмам, которые продали изделия по цене выше заявленной, выдаются все сведения о них;
- по фирме, чье изделие получило самую высокую цену, выводятся все сведения о ней и об изделии;
- по фирме, получившей максимальную прибыль, выдаются все сведения о ней, количестве проданных изделий и величине прибыли.

Все действия оформить в виде отдельных функций.

23. ***Морская полиция*****.** Отряд морской полиции осуществляет патрулирование участков морской акватории. Ежедневно патруль, состоящий из двух человек, на катере должен посетить несколько участков, пользующихся особым вниманием браконьеров, и задержать их, если они будут заниматься незаконной добычей рыбы. Для получения сведений о результатах работы отряда морской полиции создать бинарный файл, содержащий следующую информацию: дата патрулирования, номер катера, фамилии патрульных, номер участка акватории, количество задержанных нарушителей.

Написать программу, в результате выполнения которой:

- осуществляется обновление, добавление и удаление информации в файле;
- по указанному катеру выводится перечень выполненных патрулирований за указанный период с указанием сведений об общем количестве задержанных браконьеров;
- по указанному участку акватории формируется и выводится перечень дат, номеров катеров, осуществлявших патрулирование, и количестве задержанных браконьеров;
- по участку, пользующемуся наибольшим вниманием браконьеров, выдаются сведения о количестве задержанных там нарушителей и общем количестве патрулирований за указанный период;

- по экипажу, задержавшему наибольшее количество нарушителей, выводятся сведения об экипаже и количестве выполненных патрулирований;
- по каждому катеру определяется количество выходов на дежурство с указанием количества проверенных участков.

Все действия оформить в виде отдельных функций.

Литература

1. Абрамов С.А., Гнездилова Г.Г., Капустина Е.Н., Селюн М.И. Задачи по программированию. – М.: Наука, 1988. – 224с.
2. Гуденко Д.А., Петроченко Д.В. Сборник задач по программированию. – С-Пб.: Питер, 2003. – 188с.
3. Керниган Б., Ритчи Д. Язык программирования Си. – С-Пб.: Невский диалект, 2000. – 352с.
4. Котов В.М., Волков И.А., Лапо А.И. Информатика. Методы алгоритмизации. – Мн.: Народная Асвета, 2000. –302с.
5. Пильщиков В.Н. Сборник упражнений по языку Паскаль. – М.: Наука, 1989. –156с.
6. Прищепов М.А., Степанцов В.П., Севернева Е.В. Экзамен по информатике. Основы алгоритмизации и программирования. – Мн.: Тетра-Системс, 2001. – 192с.
7. Шилдт Г. Полный справочник по Си. – М.: Вильямс, 2002. – 704с.

Содержание