Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

Л. А. АРТЮШИНА А. А. ВОРОНИНА

ТЕХНОЛОГИИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ

Учебное пособие

Владимир 2014

УДК 621.396
ББК 32.988-5я7
А86

Рецензенты:
Кандидат технических наук, доцент
кафедры информатики и защиты информации
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых
Д. А. Полянский

Кандидат технических наук, доцент
кафедры информационного обеспечения в правовой сфере
Юридического института Московского государственного
университета путей сообщения
Л. М. Груздева

Печатается по решению редакционно-издательского совета ВлГУ

Артюшина, Л. А.
А86 Технологии и методы программирования : учеб. пособие / Л. А. Артюшина, А. А. Воронина ; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир : Изд-во ВлГУ, 2014. – 96 с. – ISBN 978-5-9984-0432-0.

Рассматриваются вопросы технологии разработки программного обеспечения при структурном подходе. Приводятся примеры выполнения каждого задания, раскрываются приемы и методы структурного программирования. Содержат требования к содержанию отчета.

Предназначено для подготовки студентов 1-го курса направления 090900 – Информационная безопасность и специальности 090305.65 – Информационно-аналитические системы безопасности всех форм обучения к выполнению лабораторных работ по дисциплине «Технологии и методы программирования» Может быть полезно для всех, начинающих изучать язык С.

Рекомендовано для формирования профессиональных компетенций в соответствии с ФГОС 3-го поколения.

Ил. 20. Табл. 10. Библиогр.: 5 назв.

УДК 621.396
ББК 32.988-5я7

ISBN 978-5-9984-0432-0

ПРЕДИСЛОВИЕ

Основная задача этого пособия – научить создавать программы на языке С в методологии структурного подхода к программированию.

Книга предназначена для студентов, изучающих язык С «с нуля», поэтому изложение материала начинается с рассмотрения самых простых примеров, усложнение производится постепенно. Для облегчения усвоения нового материала выполняется разбор и анализ лабораторных заданий, приведенных в пособии, подробно рассматривается технология отладки и тестирования программ.

Книга является логическим продолжением лекций преподавателя, поэтому не претендует на полноту изложения материала. Кроме этого существуют книги других авторов, справочники, контекстная помощь.

Цель, которая ставилась при написании книги, – дать правильное представление о технологии структурного программирования. Структурный подход охватывает все стадии разработки программного обеспечения (ПО): постановку задачи, анализ и исследование задачи, разработку алгоритма, собственно программирование, отладку и тестирование. В соответствии с поставленной целью при подаче нового материала авторы уделили большее внимание рассмотрению стадий разработки ПО в парадигме структурного программирования, и меньшее внимание – рассмотрению конструкций языка С.

Техническое и программное обеспечение

Для работы с программами вам потребуется компилятор языка С++. Для примеров из этой книги подойдет компилятор Microsoft С++. Другие компиляторы, рассчитанные на стандартный С++, будут безошибочно воспринимать большинство программ в том виде, в котором они приведены в книге.

В пособии содержится большое количество примеров программ. Все программы являются консольными, т.е. выполняющимися в текстовом окне компилятора. Это сделано для того, чтобы избежать сложностей, возникающих при работе с полноценными графическими приложениями Windows.

Лабораторные работы

Содержание лабораторных работ посвящено изучению особенностей разработки программного обеспечения при структурном подходе на примере языка программирования высокого уровня С.

Приводятся различные приемы и методы структурного программирования. Все лабораторные работы ориентированы на применение технологии структурного программирования. В каждой лабораторной работе приводится пример выполнения типового задания с учетом предъявляемых требований.

Каждая лабораторная работа содержит 16 вариантов индивидуальных заданий, указания к выполнению лабораторных работ. Необходимый справочный материал размещен в прил. 1 – 4.

ОБЩИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

1. Предварительно изучите теоретический материал соответствующей темы.
2. Письменные упражнения выполняйте в тетради для лабораторных занятий.
3. Упражнения, требующие написания программы, выполняются следующим образом:
    1) проанализировать исходные данные и предполагаемый результат с целью определения оптимальной структуры данных программы и используемых алгоритмов;
    2) выполнить структурную декомпозицию задачи;
    3) выполнить детализацию алгоритмов модулей задачи;
    4) разработать тестовые примеры, содержащие исходные данные и ожидаемые результаты;

5) реализовать программу по выполненной декомпозиции;
6) протестировать ее работу при различных входных параметрах;
7) показать преподавателю на примерах правильность результатов, полученных с помощью разработанной программы;
8) по требованию преподавателя кратко сформулировать алгоритм работы программы и выделить в программе основные блоки, ответственные за выполнение определенных действий (например, ввод исходных данных, объявление переменных, вывод на экран и т.д.);
9) по требованию преподавателя объяснить работу всех использованных в программе языковых конструкций;
10) по требованию преподавателя объяснить назначение и смысл всех использованных констант и переменных.

Содержание отчетов

Каждая лабораторная работа выполняется с оформлением соответствующей документации. В документации обязательно должны быть представлены следующие пункты:

1. Титульный лист (прил. 4).
2. Текст индивидуального задания по варианту.
3. Структурная декомпозиция задачи.
4. Детализация алгоритмов модулей задачи.
5. Текст программы решения задачи на языке высокого уровня С.
6. Тесты и результаты тестирования. Тесты должны включать различные наборы входных данных, позволяющие исследовать работу программы для каждой ветви алгоритма. Для каждого набора входных данных должен быть приведён полученный набор выходных данных.
7. Выводы по работе.

Лабораторная работа № 1

ЗНАКОМСТВО СО СРЕДОЙ MICROSOFT VISUAL C++ 2008 EXPRESS EDITION. СОЗДАНИЕ ИСХОДНОГО ФАЙЛА И ЕГО ПРЕОБРАЗОВАНИЕ В ГОТОВУЮ К ЗАПУСКУ ПРОГРАММУ

1. Цель работы

Приобретение обучающимися умений и навыков в работе с оборудованием компьютерного класса, со средой Microsoft Visual C++2008 Express Edition, ее настройка и дальнейшее использование для написания программ на языке С; знакомство с языком программирования С.

2. Порядок выполнения

Задание 1

1. Создайте каталог для своих работ. В качестве имени каталога используйте свою фамилию.
2. Создайте с помощью текстового редактора Блокнот файл с расширением .c, например f1.c.
3. Запустите среду программирования Microsoft Visual C++ 2008 Express Edition. Для этого воспользуйтесь командой главного меню Пуск | Программы | Microsoft Visual C++ 2008 Express Edition.
4. Создайте новый проект типа «консольное приложение». Для этого выполните следующие действия:
    1) Выберите в строке Меню главного окна File –New….
    2) В открывшемся диалоговом окне New выберите вкладку Project:
        - в окне Project types выберите тип проекта – Win32;
        - в окне Templates – Win32 Console Application;
        - введите в поле Name имя проекта, например First;
        - в поле Location введите имя каталога размещения файлов проекта (если указанный вами каталог отсутствует, он будет создан автоматически);
        - нажмите ОК.
    3) В открывшемся окне Мастера приложений Application Wizard выберите вкладку Application Settings, на ней определите подтип консольного приложения Empty project (пустой проект).

4) Нажмите кнопку Finish.
5) Добавьте к проекту исходный файл f1.c. Для этого:
   – скопируйте файл f1.c в папку проекта First;
   – щелкните правой кнопкой мыши по папке Source File, выберите Add | Existing Item… ;
   – в открывшемся диалоговом окне Insert Files выберите файл f1.c;
   – в окне проекта папка Source Files раскроется, и в нее будет помещен файл f1.c.
6) Щелкните два раза левой кнопкой мыши по ярлыку добавленного файла, при этом откроется окно редактора Editor.

5. Перейдите в окно редактора и наберите листинг программы, приведенный ниже:

```
1.     //f1.c
2.     #include <stdio.h>
3.     #include <conio.h>
4.     #include <locale.h>
5.     int main ()
6.     {
7.    float a,b,summa;
8.    setlocale(LC_CTYPE, "russian");
9.    printf ("введите a,b\n");
10.   scanf ("%f %f ",&a,&b);
11. summa=a+b;
12. printf ("\n сумма=%5.2f",summa);
13. getch ();
14. return 0;
15.    }
```

6. Откомпилируйте программу. Для этого выберите опцию главного меню Build | Build First. Диагностические сообщения компилятора и сборщика отображаются в окне вывода Output. Если все в порядке, последняя строка в окне вывода будет выглядеть следующим образом: `First - 0 error(s), 0 warning(s).`
7. Запустите программу на выполнение, щелкнув по кнопке Start Debugging.

3. Общая структура программы на языке С

Рассмотрим детально набранную программу.

Программа начинается с комментария. Знаки // обозначают начало комментария – все правее них до конца строки не обрабатывается компьютером и служит нам для пояснения программы. Комментарий также можно ограничивать парами символов /* (начало комментария) и */ (конец комментария). В этом случае комментарий может быть многострочный, то есть состоять из нескольких строк.

Далее в строках 2, 3, 4 записаны конструкции #include < stdio.h >, #include <conio.h> и #include <locale.h>. Эти записи означают подключение функций стандартного ввода и вывода, описание которых находится в библиотеках stdio.h и conio.h, и функции setlocale (), позволяющей в нашем случае выводить сообщения на русском языке.

Программная библиотека представляет собой набор функций, облегчающих работу пользователя. В прил. 1 приведены некоторые из часто используемых функций указанных библиотек.

Далее программа содержит заголовок функции с именем main (). Выполнение любой программы на С начинается с вызова функции main (). Поэтому каждая программа на языке С должна ее содержать.

Следующая строка

{

содержит открывающуюся фигурную скобку, обозначающую начало тела функции main (). Тело функции состоит из набора объявлений, определений и операторов. Каждое из них должно завершаться символом точки с запятой.

Рассмотрим их более подробно.

В строке 7 объявлены и определены три переменные a, b, summa. Объявление переменной предполагает указание имени переменной (например, summa) и ее типа (например, float). Если при объявлении переменной одновременно выделяется память под нее, то происходит определение переменной. Оператор `float a,b,summa;` в программе f1 является и объявлением и определением, поскольку выделяет память под переменные `a,b,summa.` Позже вы познакомитесь с теми видами объявлений, которые не являются определениями.

В языке С имена, которые используются для обозначения переменных, называются идентификаторами. Идентификатор может содержать латинские буквы, цифры и символ подчеркивания и начи-

наться обязан с буквы или символа подчеркивания. В стандарте ANSI языка С идентификатор определяется своими первыми 32 символами. Строчные и прописные буквы рассматриваются в С как разные символы. Идентификатор не должен совпадать с ключевыми словами (командами, конструкциями языка).

Многие программисты придерживаются негласного правила использовать в названиях переменных буквы только нижнего регистра. Есть программисты, которые употребляют в именах переменных как строчные, так и прописные буквы, например `IntVar`. Некоторые используют символ подчеркивания.

Важно:

- какой бы подход вы не использовали, желательно придерживаться его в рамках программы;
- как правило, имена, состоящие только из символов верхнего регистра, чаще всего применяются для обозначения констант, реже – имен классов, функций, проектов;
- желательно, чтобы имя переменной отражало смысл ее содержимого. Например, имя `summa` предпочтительней, чем `s`.

Основные типы переменных языка С приведены в прил. 2.

При объявлении переменная также может быть инициализирована (определено ее начальное значение) некоторой величиной из диапазона допустимых значений. Для этой цели используется оператор присваивания «=». Общая форма объявления переменной:

Тип_переменной идентификатор_переменной [=начальное значение];

В квадратных скобках указано необязательное выражение. Можно считать, что неинициализированная переменная не имеет определенного значения (точнее, ее значение непредсказуемо).

Объявление переменной может размещаться почти в любом месте программы. Однако оно всегда должно предшествовать первому обращению к этой переменной. Одна и та же переменная может быть объявлена несколько раз в разных блоках программы. Нельзя объявить дважды одну переменную в одном блоке программы (в цикле, функции и т.д.).

В 9, 10-й строках с помощью функций printf() и scanf() осуществляется форматированный ввод/вывод на консоль. Форматированный ввод и вывод означает, что функции могут читать и выводить данные в разном формате, которым вы можете управлять. Управляю-

щая строка содержит два типа информации: символы, которые непосредственно выводятся на экран, и команды формата, определяющие, как выводить аргументы. Команды формата начинаются с символа %, за которым следует код формата.

Во многих спецификаторах преобразования можно указать модификаторы, которые меняют их значение. Например, можно указывать минимальную ширину поля, количество десятичных разрядов и выравнивание по левому краю. Модификатор формата помещают между знаком процента и кодом формата.

В нашем случае в строке 12 модификатор 5.2 означает, что на экран должно выводиться число, целая часть которого содержит пять цифр, а дробная – 2 цифры. Список кодов формата указан в прил. 3.

printf () – функция вывода информации на консоль. С ее помощью в окне приложения можно вывести как строку простого текста, так и значения переменных различных типов.

Общая форма записи функции:

printf (“форматная строка“ [, перем1], [перемен2] […]);

Здесь в круглых скобках указаны обязательные параметры, а в прямоугольных скобках – параметры, которые указываются по необходимости.

Функция printf () также дает возможности управления выводом с помощью управляющих последовательностей, начинающихся с символа ESC (обратный слэш «\»). Список управляющих последовательностей языка С дан в прил. 3.

scanf() – функция ввода с консоли. Общая форма записи этой функции: *scanf (“форматная строка”, &перем1 [, &перем2] [, …]);*

Аргументы функции scanf() аналогичны соответствующим аргументам функции printf() за исключением того, что в качестве параметров scanf принимает не имена переменных, а их адреса. В силу этого перед именем каждой переменной в scanf должен стоять знак операции взятия адреса & (амперсанд).

В строке 13 с помощью функции `getch()` осуществляется «задержка» изображения на экране.

Запись `return 0;` является указанием функции main() возвратить значение 0 вызывающему окружению; в данном случае это может быть компилятор или операционная система.

Последняя строка программы
}

содержит закрывающуюся фигурную скобку. Она обозначает конец функции main () и конец основной части программы (в большинстве случаев – конец программы).

4. Сохранение, закрытие и открытие проектов

Проект сохраняется автоматически. Но можно сохранить проект, используя команды Save All, Save меню File.

Для открытия существующего проекта выберите последовательно пункты Open | Project/Solution из меню File.

5. Рекомендации по использованию встроенного отладчика Microsoft Visual C++ 2008

Отладка программ – процесс исправления ошибок в программе. Рассмотрим некоторые режимы отладки программ.

Чтобы запустить отладчик, необходимо нажать клавишу F10. В этом случае на экране появится желтая стрелка рядом с окном документа, указывающего на строку с открывающейся фигурной скобкой в main().

Если вы хотите начать трассировку не с начала, установите курсор на нужную строку.

Пошаговая трассировка. Этот режим позволяет следить за тем, как изменяются значения различных переменных. Нажимая клавишу F10, можно выполнять последовательно один оператор программы за другим. Значения переменных, выбранных компилятором, будут выводиться в окне закладки Autos. Если вы хотите пропустить пошаговое выполнение некоторого куска программы, нажмите клавиши Shift+F11.

Просмотр переменных. Если требуется создать собственный набор просматриваемых переменных, внесите их в окно просмотра закладки Autos. Для этого щелкните правой кнопкой мыши на имени переменной в исходном коде. Из появившегося меню выберите Add Watch (Добавить). Имя переменной и ее текущее значение появится в окне просмотра. Если переменная пока недоступна, окно просмотра выдаст сообщение об ошибке вместо значения переменной.

Пошаговая трассировка функций. Следует заметить, что режим трассировки по F10 рассматривает каждую встроенную функцию как одно выражение. Если вам необходимо осуществить пошаговую трассировку каждого выражения функции, нажмите F11. Если вы хотите прервать пошаговую трассировку функции, воспользуйтесь клавишами Shift+F11.

Точки останова. Этот режим позволяет временно прерывать работу программы в указанных местах. Для того чтобы вставить точки останова в листинги, нужно установить курсор на ту строку, в которой программа при трассировке должна остановиться. Затем нажать правую кнопку мыши и выбрать из меню Breakpoint | Insert Breakpoint. Напротив этой строки кода появится красный кружок. Теперь, если вы, например, запустите программу на выполнение, программа остановится на этой строке. На этом временном срезе можно проверить значения переменных, произвести пошаговую трассировку кода и т.д.

Чтобы удалить точку останова, встаньте на красный кружок, нажмите правую кнопку мыши и выберите из появившегося меню Delete Breakpoint.

Задание 2.

Поэкспериментируйте с программой f1.c. Построчно исполните код программы f1.c, используя встроенный отладчик Microsoft Visual C++ 2008. Проследите за тем, как изменяются значения переменных a, b, summa.

Лабораторная работа № 2

ВЕТВЛЕНИЯ

1. Цель работы

Приобретение обучающимися практических умений и навыков в работе с базовыми конструкциями структурного программирования, в тестировании и отладке программ.

В языке С существует несколько типов ветвлений. Рассмотрим каждый из них. Условный оператор *if* служит для выбора направления работы программы в зависимости от условий, сложившихся в данной точке программы на момент ее выполнения.

Общая форма записи условного оператора:

```
 if (условие)
{
      блок операторов 1;
 }
 else
{
      блок операторов 2;
 }
```

Если на момент выполнения условие истинно, программа передает управление блоку операторов 1 и далее первому оператору за пределами конструкции if-else. При этом блок операторов 2 не выполняется. Если на момент выполнения условие ложно, выполняется блок операторов 2, а блок операторов 1 не выполняется. В табл. 1 указаны простейшие операции отношения.

Таблица 1

Операция	Запись на Си
Больше	>
Меньше	<
Больше либо равно	>=
Меньше либо равно	<=
Равно	= =
Не равно	!=

Вложенные операторы условия

Операторы условия могут быть вложенными друг в друга в соответствии с тем программным алгоритмом, который они реализуют. Допускается произвольная степень их вложенности. Например:

```
if (a<=b) // начало внешнего оператора условия
{
 if (x!=0) printf ("x!=0 \n");//начало вложенного
                                оператора условия
         else // начало ветви else, относящейся
              // к вложенному оператору условия
```

```
    {
                x=1;
                 y=0;
     }          // конец ветви else, относящейся
                // к вложенному оператору условия
  }
  else          // начало ветви else, относящейся
                // к внешнему оператору условия
   {
  a=b;
  printf ("%d" ,a);
   }            // конец ветви else, относящейся
                // к внешнему оператору условия
```

Сокращенные варианты записи

При программировании обыденной является ситуация, когда требуется некоторое действие в ответ на сложившиеся условия. Например, если получены неверные исходные данные от пользователя, то выдать сообщение об ошибке и выйти из программы. В таких случаях используется сокращенная запись оператора условия с отсутствующим блоком else. Общая форма записи:

if (условие)
{
    блок операторов;
}

Здесь в случае истинности условия управление передается блоку операторов в фигурных скобках. В случае ложности условия этот блок пропускается.

Составные логические выражения

В программировании распространены двойные условия, которые в математике записываются в виде f<b<c. В программе такие условия должны быть переформулированы с использованием простых операций сравнения и логических операций «И», «ИЛИ», «НЕ». Обозначение логических операций приведено в табл. 2.

Таблица 2

Логическая операция	Знак С	Наименование знака
И	&&	Двойной амперсанд
ИЛИ	||	Двойная вертикальная черта
НЕ	~ (!)	Не

Например:

```
if ((a>b) && (a>c)) // если a больше b и a больше c
 printf ("a=%f",a); // вывести значение переменной a
else                  // иначе
{
 if ((b>a) && (b>c)) // если b больше a и b больше c
 prinrtf("b=%f",b); // вывести значение переменной b
 else
      prinrtf("c=%f",c);  // иначе вывести значение
}                          // переменной c
```

Оператор *switch* используется в том случае, если в программе присутствует большое дерево ветвлений и все ветвления зависят от значения какой-либо одной переменной.

Общий формат записи:

switch (n)
{
case 1:
оператор 1;
break;
case 2:
оператор 2;
break;
case 3:
оператор 3;
break;
……..
}

где *n* – целочисленная или символьная переменная;

1, 2, 3 – целочисленная или символьная константа;
Оператор 1 – тело первого case;
Оператор 2 – тело второго case;
Оператор 3 – тело третьего case;
break – (прервать) оператор завершает выполнение ветвления switch, если значение переменной в операторе switch совпадает с одним из значений констант, указанных внутри ветвления. Если значение переменной в операторе switch не совпадет ни с одним из значений констант, то управление будет передано в конец switch без выполнения каких-либо действий. В случае отсутствия оператора break управление будет передано операторам, относящимся к другим веткам switch.

Условная операция

Существует распространенная в программировании операция: переменной необходимо присвоить одно значение в случае выполнения некоторого условия и другое значение в случае невыполнения этого условия. С помощью конструкции if … else это будет выглядеть следующим образом:

```
if (alfa<beta)
          min=alfa;
else
          min=beta;
```

Подобные действия на практике настолько распространены, что была специально разработана условная операция, выполняющая эти действия. Эта операция записывается с помощью двух знаков и использует три операнда.

С помощью условной операции можно записать предыдущий фрагмент следующим образом: *min = (alfa<beta) ? alfa : beta;*

Правая часть оператора *(alfa<beta) ? alfa : beta* представляет собой условное выражение. Знак ? и двоеточие : обозначают условную операцию. Условие стоит перед знаком вопроса *(alfa<beta)* и является условием проверки. Это условие вместе с операндами *alfa и beta* составляют тройку операндов условной операции.

Если значение проверяемого условия истинно, то условное выражение становится равным значению *alfa*, в противном случае *beta*.

Скобки в данном случае использованы, чтобы визуально упростить читаемость этого оператора.

Пример выполнения лабораторной работы: требуется разработать программу, которая вычисляет значение функции $m(a) = \begin{cases} \frac{a}{2}, a > 1 \text{ и } b = 0 \\ a + 1, a = 2 \text{ или } b > 1 \end{cases}$.

Анализ задачи показывает, что программу можно строить как последовательность подзадач. Следовательно, на первом шаге декомпозиции с использованием метода пошаговой детализации получаем следующую структурную схему программы, представленную на рис. 1, *а*.

Подзадачу *Вычисление значения функции* можно оформить в виде функции. Эта же программа может быть реализована и без использования функций.

Выполнять детализацию алгоритма основной программы не будем из-за его очевидной простоты. Выполним детализацию алгоритма функции. Исходные данные такой функции – значения переменных a, b. Функция не должна менять их значения, поэтому значения переменных можно передать как параметры-значения или параметры-константы.

На рис. 1, *б*, 1, *в* приведена схема алгоритма проектируемой программы с использованием подпрограммы типа функция. На рис. 3 приведена схема алгоритма программы без использования функции.

Из-за очевидной простоты алгоритма основной программы проведем модульное тестирование по критерию комбинаторного покрытия условий только одного компонента программы – вычисление значения функции m (см. рис. 1, *в*). На рис. 2 представлен граф передачи управления, соответствующий схеме алгоритма функции (см. рис. 1, *в*).

Из построенного графа видно, что для функции по критерию комбинаторного покрытия условий необходимо покрыть тестами восемь комбинаций:

1) a>1, b=0;
2) a>1,b≠0;
3) a≤ 1,b=0;
4) a≤ 1,b≠0;
5) a = 2, b> 1;
6) a = 2, b≤1;
7) a≠2, b> 1;
8) a ≠2, b≤1.

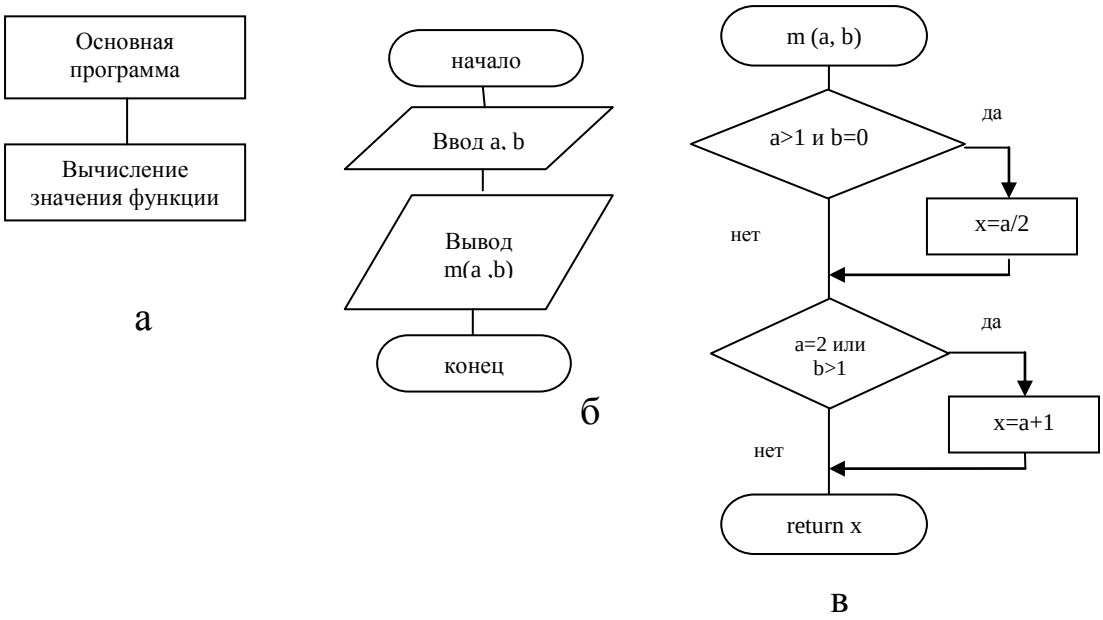


Рис. 1. Алгоритм программы примера с использованием функции

Эти комбинации можно проверить тестами:

a = 2, b = 0 — проверяет комбинацию (1);
a = 2, b=1 — проверяет комбинации (2), (6);
a=2, b=3 — проверяет комбинацию (5);
a = 1, b = 0 — проверяет комбинацию (3);
a = 1, b = 1 — проверяет комбинации (4), (8);
a=1, b=3 — проверяет комбинацию (7).

Текст программы на языке С с использованием функции:

```
//func.c
#include <stdio.h>
#include <locale.h>
float m (float a, float b, float x)
  {
if (a>1 && b==0) x/=2;
if (a==2 || b>1)x++;
  return x;
}

int main(void)
{
  float a,b,x;
   setlocale(LC_CTYPE, "russian");
  printf ("введите a,b\n");
  scanf ("%f%f", &a,&b);
```

```
  printf ("\nm(x)=%f",m(a,b));
  return (0);
}
```

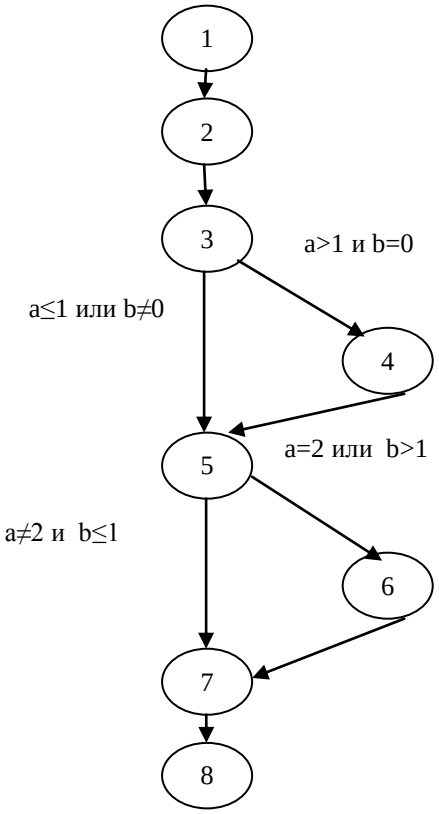
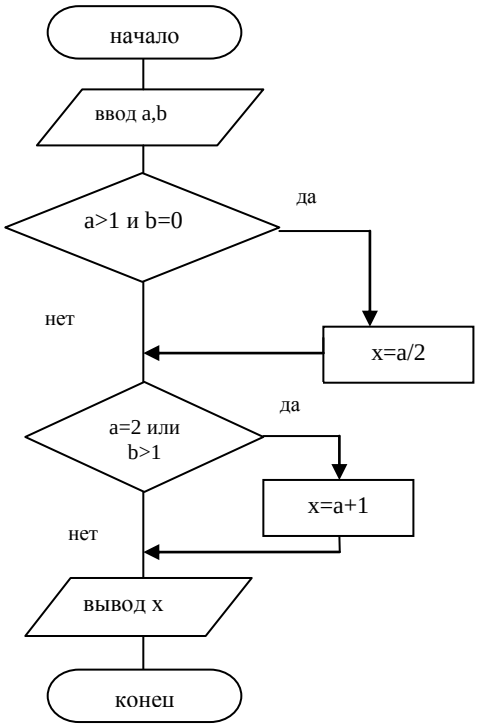


Рис. 2. Граф передачи управления для схемы алгоритма функции (в)



3

Рис. 3. Блок-схема алгоритма программы примера

Текст программы на языке С без использования функции:

```
#include <stdio.h>
#include <conio.h>
#include <locale.h>
int main (void)
{
  float a,b,x;
  setlocale(LC_CTYPE, "russian");
  printf ("введите a,b\n");
  scanf ("%f %f ",&a,&b);
  if ((a>1) && (b==0)) x/=2;
  if ((a==2) || (b>1)) x++;
  printf ("\nx=%f",x);
  return (0);
}
```

2. Порядок выполнения

Задание 1:

1. Разработайте структурную схему и выполните детализацию алгоритмов модулей к задаче индивидуального задания:
   а) с использованием функции;
   б) без использования функции.
2. Напишите программы на языке С для разработанных алгоритмов решения задачи.
3. Выполните отладку и компиляцию программ, получите исполняемые файлы.
4. Выполните тестирование программ.

3. Варианты заданий

Номер варианта	Функция	Номер варианта	Функция
1	$y=\begin{cases} x-2, & x>2.5 \\ 1+x^2, 0 \le x \le 2.5 \\ x\ln\lvert\cos(x)\rvert, x<0 \end{cases}$	9	$y=\begin{cases} 1+\sqrt{\cos(x)}, x>4 \\ x+1, 0 \le x \le 4 \\ 1-x^2, x<0 \end{cases}$
2	$y=\begin{cases} \sin(2.3x-1), x>2.5 \\ 1-3\ln\lvert 1-x\rvert, 0 \le x \le 2.5 \\ \frac{x^2}{2-x}, x<0 \end{cases}$	10	$y=\begin{cases} e^{-(x+8)}, x>3.61 \\ 1, 0 \le x \le 3.61 \\ \frac{x}{5}, x<0 \end{cases}$
3	$y=\begin{cases} \sqrt{(tg(x^2-1))}, x>1 \\ -2x, 0 \le x \le 1 \\ e^{\cos(x)}, x<0 \end{cases}$	11	$y=\begin{cases} x, x>1.5 \\ 2x^2\sqrt{\lvert\cos(2x)\rvert}, 0 \le x \le 1.5 \\ e^{-\cos(3x)}, x<0 \end{cases}$
4	$y=\begin{cases} x^2-3+2.5x, x>12.5 \\ e^x+5+\cos(0.001x), 0 \le x \le 12.5 \\ x^2, x<0 \end{cases}$	12	$y=\begin{cases} 1-\sqrt{\cos(2x)}, x>2.5 \\ x^2-x, 1 \le x \le 2.5 \\ 1+x^2, x<1 \end{cases}$
5	$y=\begin{cases} 1+\sqrt{\lvert\cos(x)\rvert}, x>1 \\ x+1, -0.5 \le x \le 1 \\ 1-x^2, x<-0.5 \end{cases}$	13	$y=\begin{cases} 2x, x>4.5 \\ 1-ln\lvert 1-x^2\rvert, 0 \le x \le 4.5 \\ e^{-x}, x \le 0 \end{cases}$
6	$y=\begin{cases} 2.5x^3+6x^2-30, x>1.5 \\ x+1, 0 \le x \le 1.5 \\ x, x<0 \end{cases}$	14	$y=\begin{cases} \sqrt{\ln(x^2-1}, x>2 \\ -2x^3, 0 \le x \le 2 \\ e^{\sin(x)}, x<0 \end{cases}$
7	$y=\begin{cases} 1+x, x>14.5 \\ e^{-x}, 3 \le x \le 14.5 \\ \cos(x), x<3 \end{cases}$	15	$y=\begin{cases} 1-\frac{2x^3}{1-x^2}, x>3.5 \\ \sqrt{\cos(2x-1)}, 0 \le x \le 3.5 \\ e^{-\cos(x)}, x<0 \end{cases}$
8	$y=\begin{cases} ln\lvert 1+x, x>3.8\rvert \\ e^{-x}, 2.8 \le x \le 3.8 \\ \cos(x), x<2.8 \end{cases}$	16	$y=\begin{cases} x+1, x>2.5 \\ 1-x^5, 0 \le x \le 2.5 \\ x+ln\lvert\sin(x)\rvert, x<0 \end{cases}$

Задание 2:

1. Составьте блок-схему решения задачи индивидуального задания, используя оператор выбора.
2. Напишите программу на языке С для разработанного алгоритма решения задачи.
3. Выполните отладку и компиляцию программы, получите исполняемые файлы.
4. Выполните тестирование программы.

Вариант 1

Арифметические действия над числами пронумерованы следующим образом: 1 – сложение, 2 – вычитание, 3 – умножение, 4 – деление. Дан номер действия и два числа A и B (B не равно нулю). Выполнить над числами указанное действие и вывести результат.

Вариант 2

Единицы длины пронумерованы следующим образом: 1 – дециметр, 2 – километр, 3 – метр, 4 – миллиметр, 5 – сантиметр. Дан номер единицы длины и длина отрезка L в этих единицах (вещественное число). Вывести длину данного отрезка в метрах.

Вариант 3

Составить программу, которая по возрасту человека (вводится с клавиатуры как целое число) определяет его принадлежность к возрастной группе: от 0 до 13 – мальчик; от 14 до 20 – юноша; от 21 до 70 – мужчина;
более 70 – старец.

Вариант 4

Локатор ориентирован на одну из сторон света («С» – север, «З» – запад, «Ю» – юг, «В» – восток) и может принимать одну из трех цифровых команд: -1 – поворот налево, 1 – поворот направо, 2 – поворот на 180 градусов. Дан символ C – исходная ориентация локатора и число N – посланная ему команда. Вывести ориентацию локатора после выполнения команды.

Вариант 5

Даны два целых числа: D (день) и M (месяц), определяющие правильную дату невисокосного года. Вывести значения D и M для даты, следующей за указанной (например, дано D = 1 M = 1, надо вывести D = 2 M = 1; дано D = 31 M = 12, надо вывести D = 1 M = 1; дано D = 28 M = 2, надо вывести D = 1 M = 3).

Вариант 6

Элементы равнобедренного прямоугольного треугольника пронумерованы следующим образом: 1 – катет (a), 2 – гипотенуза (c), 3 – высота, опущенная на гипотенузу (h), 4 – площадь (S). Дан номер одного из этих элементов и его значение. Вывести значения остальных элементов данного треугольника (в том же порядке).

Вариант 7

Даны два целых числа: D (день) и M (месяц), определяющие правильную дату невисокосного года. Вывести значения D и M для даты, предшествующей указанной (например, дано D = 1 M = 1, надо вывести D = 31 M = 12; дано D = 1 M = 3, надо вывести D = 28 M = 2; дано D = 15 M = 12, надо вывести D = 14 M = 12).

Вариант 8

Элементы окружности пронумерованы следующим образом: 1 – радиус (R), 2 – диаметр (D), 3 – длина (L), 4 – площадь круга (S). Дан номер одного из этих элементов и его значение. Вывести значения остальных элементов данной окружности (в том же порядке). В качестве значения π использовать стандартную константу Pi.

Вариант 9

Дано целое число в диапазоне 20 – 69, определяющее возраст (в годах). Вывести строку – описание указанного возраста, обеспечив правильное согласование числа со словом «год», например: «20 лет», «32 года», «41 год».

Вариант 10

Составьте программу, которая по введенному вами k – числу грибов печатает фразу «Мы нашли в лесу k грибов», причем согласовывает окончание слова «гриб» с k. (Количество грибов может быть любым целым положительным числом: 1, 3, 34, 127 и т. д. Окончание фразы определяется значением последней цифры).

Вариант 11

Составить программу, которая печатает номера дней в месяце, если вводится день недели. Считаем, что 1–е число месяца – понедельник, в месяце 31 день. Выводить на экран словесное описание дня недели и соответствующие числа месяца (например, вводится число 2, на экране появляется: «Вторник – 2, 9, 16, 23, 30»).

Вариант 12

Составить программу, вычисляющую площадь геометрической фигуры. Тип фигуры определяется символом (c): O – окружность, T – равнобедренный прямоугольный треугольник и K – квадрат. Целое число, вводимое вслед за символом, определяет соответствующий элемент для вычисления площади (для окружности это радиус, для треугольника – длина катета, для квадрата – длина стороны).

Вариант 13

Написать программу, вычисляющую стоимость 10-минутного междугороднего разговора, в зависимости от кода города: Москва (905) – 4.15 руб./мин, Ростов (194) – 1.98 руб./мин., Краснодар (491) – 2.69 руб./мин, Киров (800) – 5.00 руб./мин.

Вариант 14

С клавиатуры вводятся два целых числа, обозначающих возраст человека и его пол (1 – мужской, 2 – женский). Составить программу, которая в зависимости от введенных данных определяет принадлежность человека к определенной группе: от 0 до 13 – мальчик (девочка); от 14 до 20 – юноша (девушка); от 21 до 70 – мужчина (женщина); более 70 – старец (старушка).

Вариант 15

Составьте программу для определения числа дней в месяце, если даны: номер месяца N – целое число от 1 до 12.

Вариант 16

Дано целое число n, соответствующее количеству углов геометрической фигуры. Составить программу, которая по введенному числу n печатает название фигуры (например, при n = 3 программа напечатает «треугольник», при n = 5 – «пятиугольник», при n > 8 – «многоугольник»). В случае если вводится число меньше 2, выводится сообщение об ошибке.

Лабораторная работа № 3

ЦИКЛЫ

1. Цель работы

Приобретение обучающимися практических умений и навыков в работе с базовыми конструкциями структурного программирования.

Действие циклов заключается в последовательном повторении определенной части программы некоторое количество раз. Повторение продолжается до тех пор, пока выполняется соответствующее условие. Когда значение выражения, задающего условие, становится ложным, выполнение цикла прекращается, а управление передается оператору, следующему непосредственно за циклом.

Область видимости переменных: переменные, определенные внутри тела цикла, невидимы вне его. Невидимость означает, что программа не имеет доступа к этим переменным. Если вы попытаетесь присвоить какое-либо значение любой переменной, определенной внутри тела цикла, вне тела цикла, компилятор выдаст сообщение о том, что эта переменная *не определена* (undeclared identifier).

Форматирование и стиль оформления циклов: хороший стиль программирования предполагает сдвиг тела цикла вправо относительно оператора, управляющего циклом, и относительно остального программного кода за исключением обрамляющих фигурных скобок. Например, как в программе, печатающей значение функции $y = x^2$:

```
//primer3_1.c
#include <stdio.h>
#include <conio.h>
#include <locale.h>

int main ()
{
  float xn, xk, dx,i, y;
  setlocale(LC_CTYPE, "russian");
  printf ("введите xn, xk, dx\n");
  scanf ("%f%f%f", &xn, &xk, &dx);
  printf ("|     x      |     y      |\n");
  for (i=xn; i<=xk; i+=dx)
```

```
        {
         y=i*i;
         printf ("|   %5.2f   |   %5.2f  |\n", i, y);
        }
  getch ();
  return 0;
  }
```

В С существуют три типа циклов: for, while, do.

Цикл for организует выполнение фрагмента программы фиксированное число раз. Как правило, этот тип цикла используется тогда, когда число повторений известно заранее.

Общий формат записи:

for (<инициализация>; <условие продолжения>;<изменение счетчика>)
тело цикла;

Цикл for начинается с выполнения блока *инициализация*, где определяется начальное значение переменной, которую обычно называют *счетчиком цикла*. Далее выполняются операторы (оператор) цикла. Затем проверяется *условие продолжения*, в случае, если это условие истинно, управление передается заголовку for и значение счетчика цикла автоматически изменяется в зависимости от параметра *изменение счетчика*.

Отладка программы осуществляется с помощью средств компилятора, главным из которых является *пошаговое выполнение*.

Поэкспериментируйте с программой `primer3_1.c` проекта Primer3, используя встроенный отладчик Microsoft Visual C++ 2008:

1) построчно исполните код программы, проследите за тем, как изменяется значение переменной i;
2) поместите переменную y в окно просмотра, построчно выполните программу, проследите за тем, как изменяется значение переменной y;
3) установите точку прерывания перед оператором for, поместите переменные xn, xk, dx в окно просмотра, запустите программу на выполнение;
4) в точке останова просмотрите значения переменных xn, xk, dx.

Циклы while и do используются в тех случаях, когда число повторений заранее не известно. Причем цикл while подходит в тех случаях, когда тело цикла может не выполниться ни разу, а цикл do – когда обязательно хотя бы однократное выполнение цикла.

Общий формат записи цикла *while*:

while (условие выполнения)

тело цикла;

Исполнение тела цикла продолжается до тех пор, пока условие истинно.

Следующая программа демонстрирует механизм работы цикла while. Пользователю предлагается ввести серию значений, если вводимое значение равно нулю, происходит выход из цикла. Очевидно, что в такой ситуации нельзя заранее предугадать, сколько ненулевых значений введет пользователь:

```
1. //primer 3_2.c
2. #include <stdio.h>
3. #include <conio.h>
4. #include <locale.h>
5. int main ()
6. {
7. int n;
8. setlocale(LC_CTYPE, "russian");
9. printf ("введите число. выход-0 \n");
10. scanf ("%d", &n);
11. while (n!=0)
12.  {
13.    printf ("введите число\n");
14.    scanf ("%d", &n);
15.  }
16. printf ("\nВыход");
17. getch ();
18. return 0;
19. }
```

Важно: переменную цикла необходимо инициализировать до начала исполнения тела цикла, что продемонстрировано в строке 10. Тело цикла должно содержать оператор, изменяющий значение переменной цикла, иначе цикл будет бесконечным (строка 14).

Цикл do. Общий формат записи цикла do:

do

тело цикла;

while (условие выполнения);

Оператор действует следующим образом: выполняются операторы циклической части, проверяется условие выполнения, если оно истинно, выполняется тело цикла. Если же оно ложно, то цикл заканчивается.

Измененный текст программы primer 3_2.c демонстрирует работу цикла do:

```
//primer 3_4.c
#include <stdio.h>
#include <conio.h>
#include <locale.h>

int main ()
 {
   int n;
setlocale(LC_CTYPE, "russian");
   do
   {
     printf ("введите число. выход-0 \n");
     scanf ("%d", &n);
   }
   while (n!=0);
   printf ("\nВыход");
    getch ();
    return 0;
}
```

Пример выполнения лабораторной работы: вывести на экран в виде таблицы значения функции F на интервале от $x_{нач}$ до $x_{кон}$ с шагом dx, где *a, b, c* – действительные числа.

$$F = \begin{cases} ax^3 + bx^3 \text{ при } a < 0 \text{ и } b \neq 0 \\ \frac{x-a}{x-c} \text{ при } a > 0 \text{ и } b = 0 \\ \frac{x+5}{c(x-10)} \text{ в остальных случаях.} \end{cases}$$

Значения *a, b, c, $x_{нач}$, $x_{кон}$, dx* ввести с клавиатуры.

Выполним структурную декомпозицию программы (рис. 1 – 4).

Рис. 1. Структурная
декомпозиция программы

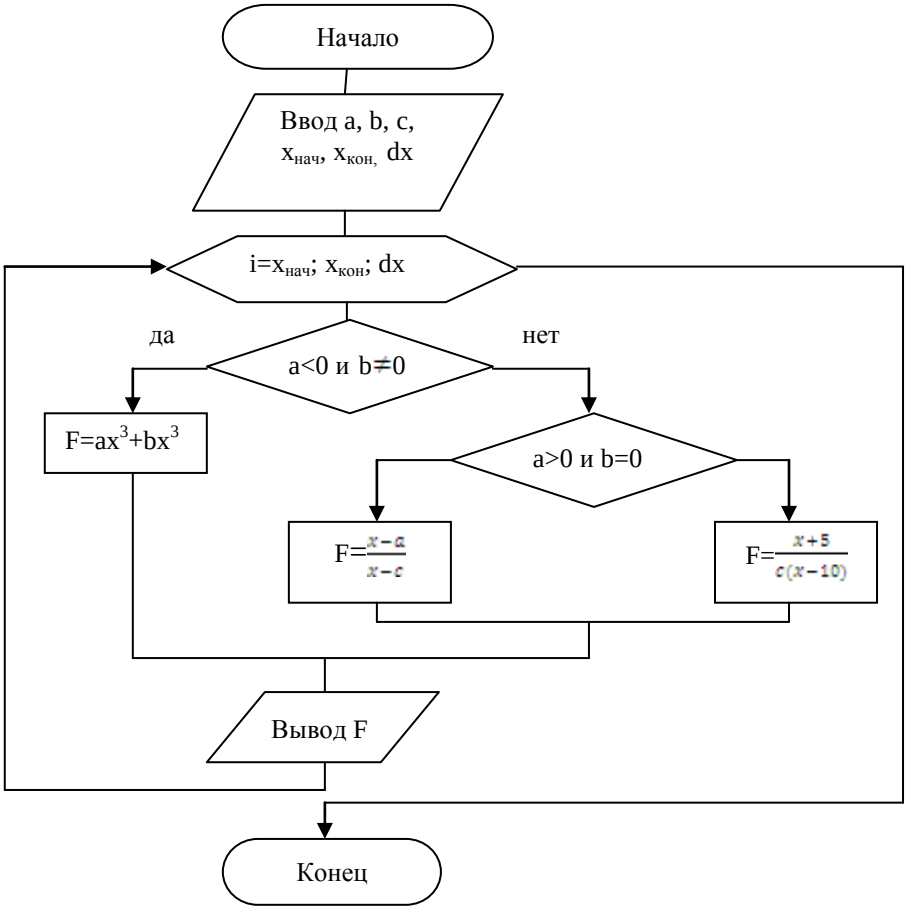


Рис. 2. Алгоритм программы без использования функции

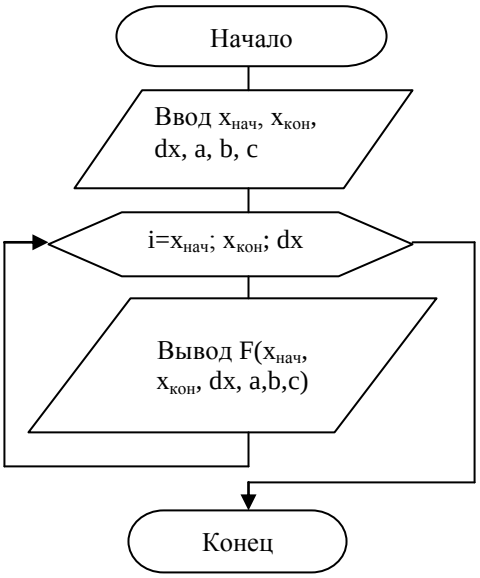


Рис. 3. Алгоритм программы
с использованием функции

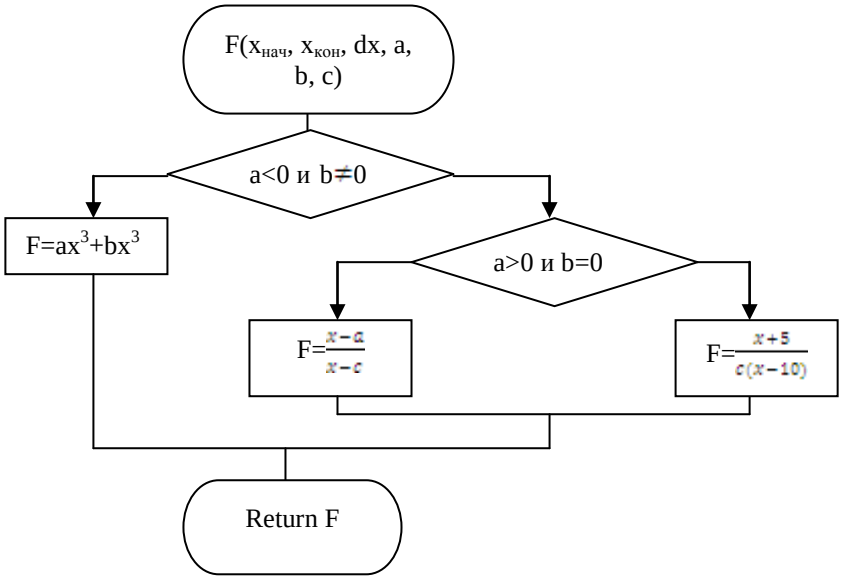


Рис. 4. Алгоритм функции

Подберем тестовые примеры для алгоритма программы, написанной с использованием функции. Воспользуемся нисходящей стратегией пошагового тестирования. Нисходящее тестирование начинается с верхнего (головного модуля программы). Согласно описанной стратегии вначале проведем тестирование модуля основной программы, затем – тестирование модуля вычисления значения функции F.

Анализ алгоритма модуля основной программы примера показывает, что, если значения переменных $x_{нач}$ или $x_{кон}$ будут равны значению переменной с или значение переменной с = 0, при вычислении значения функции F возникнет ситуация деления на ноль, которую необходимо обработать. Например, повторением ввода исходных данных до тех пор, пока не будут введены корректные входные данные.

Для того чтобы удовлетворить критерию комбинаторного покрытия условий, необходимо покрыть тестами шесть комбинаций:

1) $c \neq 0$, $x_{нач} \neq c$, $x_{кон} \neq c$;
2) $c \neq 0$,$x_{нач} = c$, $x_{кон} \neq c$;
3) $c \neq 0$,$x_{нач} \neq c$, $x_{кон} = c$;
4) $c = 0$; $x_{нач} \neq c$, $x_{кон} \neq c$;
5) $c = 0$; $x_{нач} = c$, $x_{кон} \neq c$;
6) $c = 0$; $x_{нач} \neq c$, $x_{кон} = c$;

Эти комбинации можно проверить шестью тестами:

$c = 2$, $x_{нач} = 1$, $x_{кон} = 4$; проверяется комбинация (1)
$c = 2$, $x_{нач} = 2$, $x_{кон} = 4$; проверяется комбинация (2)
$c = 2$, $x_{нач} = 1$, $x_{кон} = 2$; проверяется комбинация (3)
$c = 0$, $x_{нач} = 1$, $x_{кон} = 4$; проверяется комбинация (4)
$c = 0$, $x_{нач} = 0$, $x_{кон} = 4$; проверяется комбинация (5)
$c = 0$, $x_{нач} = -2$, $x_{кон} = 0$; проверяется комбинация (6)

Анализ алгоритма модуля основной программы примера с использованием функции также показывает, что используемая в модуле конструкция цикла представляет собой простые итерации, в которых конечное значение параметра цикла i больше или равно начальному значению. Это означает, что цикл содержит одно условие $i \leq x_{кон}$. Следовательно, требуются тесты для ситуаций:

1) $x_{нач} < x_{кон}$;
2) $x_{нач} >= x_{кон}$ (т.е. выполнение последней итерации цикла).

В этом случае для тестирования конструкции цикла достаточно использовать критерий покрытия условий. Тесты, удовлетворяющие критерию покрытия условий:

- для проверки условия (1) можно использовать тесты, проверяющие комбинации (1), (2), (3), (4), (5),(6).
- для проверки условия (2) можно дополнить указанные выше тесты следующими тестами: $c = 2$, $x_{нач} = 4$, $x_{кон} = 4$ и $c = 0$, $x_{нач} = 4$, $x_{кон} = 4$.

Таким образом, для тестирования головного модуля используются восемь тестов:

1) c = 2, $x_{нач}$ = 1, $x_{кон}$ = 4;
2) c = 2, $x_{нач}$ = 2, $x_{кон}$ = 4;
3) c = 2, $x_{нач}$ = 1, $x_{кон}$ = 2;
4) c = 0, $x_{нач}$ = 1, $x_{кон}$ = 4;
5) c = 0, $x_{нач}$ = 0, $x_{кон}$ = 4;
6) c = 0, $x_{нач}$ = -2, $x_{кон}$ = 0;
7) c = 2, $x_{нач}$ = 4, $x_{кон}$ = 4;
8) c = 0, $x_{нач}$ = 4, $x_{кон}$ = 4.

При тестировании модуля вычисления значения функции необходимо проанализировать следующие условные конструкции:

1) a < 0 и b ≠0;
2) a > 0 и b = 0.

Соответствующий схеме алгоритма функции (см. рис. 4) граф передачи управления представлен на рис. 5.

Из построенного графа видно, что для функции по критерию комбинаторного покрытия условий необходимо покрыть тестами восемь комбинаций:

1) a < 0, b ≠ 0;
2) a < 0, b = 0;
3) a ≥ 0, b = 0;
4) a ≥ 0, b ≠ 0;
5) a > 0, b = 0;
6) a > 0, b ≠ 0;
7) a ≤ 0, b ≠ 0;
8) a ≤ 0, b = 0.

Эти комбинации можно проверить четырьмя тестами:

a = -2, b = 1 – проверяются комбинации (1), (7);

a = -1, b = 0 – проверяются комбинации (2), (8);

a = 1, b = 0 – проверяются комбинации (3), (5);

a = 0, b = 1 – проверяются комбинации (4), (6).

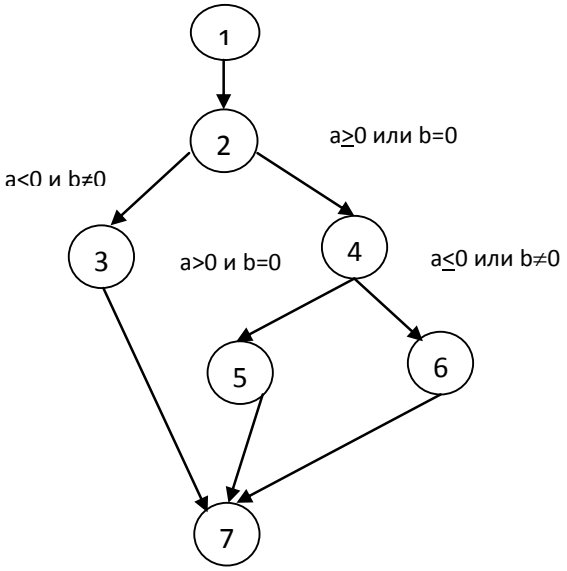


Рис. 5. Граф передачи управления

Так как модуль вычисления значения функции F получает входные данные из головного модуля (с учетом результатов построения тестов для головного модуля c ≠ 0, $x_{нач}$ ≠ c, $x_{кон}$ ≠ c, $x_{нач}$ ≠ $x_{кон}$), имеет смысл расширить тесты, проверяющие правильность вычисления значения функции F, следующими значениями входных данных: c = 2, $x_{нач}$ = 1, $x_{кон}$ = 4. Получаем следующий набор тестов для тестирования модуля, вычисляющего значение функции F:

$a = -2, b = 1, c = 2, x_{нач} = 1, x_{кон} = 4;$
$a = -1, b = 0, c = 2, x_{нач} = 1, x_{кон} = 4;$
$a = 1, b = 0, c = 2, x_{нач} = 1, x_{кон} = 4;$
$a = 0, b = 1, c = 2, x_{нач} = 1, x_{кон} = 4.$

В данном тексте программы на языке С не установлена русская локаль, поэтому тексты сообщений даны латиницей

```
//primer3_6.c
#include <stdio.h>
#include <conio.h>
#include <math.h>
float F (float i, float a, float b,float c)
{
  float y;
  if ((a<0) && (b!=0)) y=a*pow(i,3)+b*pow(i,3);
  else
  if ((a>0) && (b=0)) y=(i-a)/(i-c);
  else y=(i+5)/(c*(i-10));
  return y;
}
int main ()
{
 float xn, xk, dx,i, a,b,c;
do
{
  printf ("input a,b,c,xn, xk, dx\n");
  scanf("%f%f%f%f%f%f", &a, &b, &c, &xn, &xk, &dx);
  if (c==0 || xn==c || xk==c || xn==xk)
    printf ("input ERROR\n");
}
while ((c==0) || (xn==c) || (xk==c) ||
(xn==xk));

printf ("|     x      |     F      |\n");
for (i=xn; i<=xk; i+=dx)
printf ("|  %5.2f  |  %5.2f  |\n", i, F(i,a,b,c));
getch ();
return 0;
}
```

Текст программы на языке С без использования функции:

```
//primer3_5.c
#include <stdio.h>
#include <conio.h>
#include <math.h>
int main ()
{
float xn, xk, dx,i, a,b,c, F;
printf ("input a,b,c,xn, xk, dx\n");
  do
  {
  printf ("input a,b,c,xn, xk, dx\n");
  scanf ("%f%f%f%f%f%f", &a, &b, &c, &xn, &xk,
  &dx);
  if (c==0 || xn==c || xk==c || xn==xk)
       printf ("input ERROR\n");
  }
  while ((c==0) || (xn==c) || (xk==c) ||
  (xn==xk));

printf ("|      x       |      F       |\n");
for (i=xn; i<=xk; i+=dx)
    {
        if ((a<0) && (b!=0))
          F=a*pow(i,3)+b*pow(i,3);
        else
             if ((a>0) && (b=0)) F=(i-a)/(i-c);
             else F=(i+5)/(c*(i-10));
     printf ("|  %5.2f    |    %5.2f   |\n", i, F);
    }
getch ();
return 0;
}
```

Задание 1. Вывести на экран в виде таблицы значения функции F на интервале от $x_{нач}$ до $x_{кон}$ с шагом dx, где *a, b, c* – действительные числа. Значения *a, b, c,* $x_{нач}$, $x_{кон}$, dx ввести с клавиатуры.

2. Порядок выполнения

Для выполнения задания

1. Разработайте структурную схему и выполните детализацию алгоритмов модулей к задаче индивидуального задания:

а) с использованием функции;
б) без использования функции.

2. Напишите программы на языке С для разработанных алгоритмов решения задачи.
3. Выполните отладку и компиляцию программ, получите исполняемые файлы.
4. Выполните тестирование программ.

3. Варианты заданий

Номер варианта	Функция	Номер варианта	Функция
1	$F=\begin{cases} ax^2+b \text{ при } x<0 \text{ и } b\neq 0 \\ \frac{x-a}{x-c} \text{ при } x>0 \text{ и } b=0 \\ \frac{x}{c} \text{ в остальных случаях} \end{cases}$	9	$F=\begin{cases} ax^2+b^2x \text{ при } a<0 \text{ и } x\neq 0 \\ x-\frac{a}{x-c} \text{ при } a>0 \text{ и } x=0 \\ 1+\frac{x}{c} \text{ в остальных случаях} \end{cases}$
2	$F=\begin{cases} \frac{1}{ax}-b \text{ при } x+5<0 \text{ и } c=0 \\ \frac{x-a}{x} \text{ при } x+5>0\ b\ c\neq 0 \\ \frac{10x}{c-4} \text{ в остальных случаях} \end{cases}$	10	$F=\begin{cases} ax^2-bx+c \text{ при } x<3 \text{ и } b\neq 0 \\ x-\frac{a}{x-c} \text{ при } a>0 \text{ и } x=0 \\ 1+\frac{x}{c}, \text{ в остальных случаях} \end{cases}$
3	$F=\begin{cases} ax^2+bx+c \text{ при } a<0 \text{ и } c\neq 0 \\ \frac{-a}{x-c} \text{ при } a>0 \text{ и } c=0 \\ a(x+c) \text{ в остальных случаях} \end{cases}$	11	$F=\begin{cases} ax^2+\frac{b}{c} \text{ при } x<1 \text{ и } c\neq 0 \\ \frac{x-a}{(x-c)^2} \text{ при } x>1{,}5 \text{ и } c=0 \\ \frac{x^2}{c^2} \text{ в остальных случаях} \end{cases}$
4	$F=\begin{cases} -ax-c \text{ при } c<0 \text{ и } x\neq 0 \\ \frac{x-a}{-c} \text{ при } c>0 \text{ и } x=0 \\ \frac{bx}{c-a} \text{ в остальных случаях} \end{cases}$	12	$F=\begin{cases} ax^3+b^2+c \text{ при } x<0{,}6 \text{ и } b+c\neq 0 \\ \frac{x-a}{x-c} \text{ при } x>0{,}6 \text{ и } b+c=0 \\ \frac{x}{c}+\frac{x}{a} \text{ в остальных случаях} \end{cases}$
5	$F=\begin{cases} a-\frac{x}{10+b} \text{ при } x<0 \text{ и } b\neq 0 \\ \frac{x-a}{x-c} \text{ при } x>0 \text{ и } b=0 \\ 3x+\frac{2}{c} \text{ в остальных случаях} \end{cases}$	13	$F=\begin{cases} ax^2+b \text{ при } x-1<0 \text{ и } b-x\neq 0 \\ \frac{x-a}{x} \text{ при } x-1>0 \text{ и } b+x=0 \\ \frac{x}{c} \text{ в остальных случаях} \end{cases}$
6	$F=\begin{cases} ax^2+b^2x \text{ при } c<0 \text{ и } b\neq 0 \\ \frac{x+a}{x+c} \text{ при } c>0 \text{ и } b=0 \\ \frac{x}{c} \text{ в остальных случаях} \end{cases}$	14	$F=\begin{cases} -ax^3-b \text{ при } x+c<0 \text{ и } a\neq 0 \\ \frac{x-a}{x-c} \text{ при } x+c>0 \text{ и } a=0 \\ \frac{x}{c}+\frac{c}{x} \text{ в остальных случаях} \end{cases}$
7	$F=\begin{cases} -ax^2-b \text{ при } x<5 \text{ и } c\neq 0 \\ \frac{x-a}{x} \text{ при } x>5 \text{ и } c=0 \\ \frac{-x}{c} \text{ в остальных случаях} \end{cases}$	15	$F=\begin{cases} -ax^2+b \text{ при } x<0 \text{ и } b\neq 0 \\ \frac{x}{x-c}+5{,}5 \text{ при } x>0 \text{ и } b=0 \\ \frac{x}{-c} \text{ в остальных случаях} \end{cases}$
8	$F=\begin{cases} -ax^2 \text{ при } c<0 \text{ и } a\neq 0 \\ \frac{a-x}{cx} \text{ при } c>0 \text{ и } a=0 \\ \frac{x}{c} \text{ в остальных случаях} \end{cases}$	16	$F=\begin{cases} a(x+c)^2-b \text{ при } x=0 \text{ и } b\neq 0 \\ \frac{x-a}{-c} \text{ при } x=0 \text{ и } b=0 \\ a+\frac{x}{c} \text{ в остальных случаях} \end{cases}$

Лабораторная работа № 4

ОДНОМЕРНЫЕ МАССИВЫ

1. Цель работы

Приобретение обучающимися практических умений и навыков применения типовых алгоритмов обработки одномерных массивов.

В случае простых переменных каждой области памяти для хранения одной величины соответствует свое имя. Если требуется работать с группой величин одного типа, их располагают в памяти последовательно и дают им общее имя, а различают по порядковому номеру. Такая последовательность однотипных величин называется массивом.

Как и обычная переменная, перед использованием массив должен быть объявлен. Основная форма объявления массива размерности n такова: тип данных *имя массива* [размер 1] [размер 2] …. [размер n];

В данной лабораторной работе ограничимся рассмотрением одномерного массива. При описании одномерного массива объявляющая запись имеет вид: тип *имя массива* [размер];

где тип – базовый тип элементов массива;

размер – количество его элементов.

Массивы, как и любые другие объекты, можно размещать либо с помощью операторов описания в сегментах данных или стека, либо в динамической памяти с помощью операций выделения памяти. Вследствие этого размерность массива может быть задана только константой или константным выражением, поскольку при таком подходе для её изменения достаточно скорректировать значение константы всего лишь в одном месте программы.

Способы инициализации одномерных массивов:

1-й способ. При описании массив можно инициализировать, т.е. присвоить его элементам начальные значения, например: *int mas[5]={31, 54, 77, 52, 93};*

Если инициализирующих значений меньше, чем элементов массива, остаток массива обнуляется, если больше – лишние значения не используются.

Важно: элементы массива нумеруются с нуля. Автоматический контроль выхода индекса за границы массива не производится, поэтому программист должен следить за этим самостоятельно.

2-й способ. Начальные значения элементов массива можно ввести с клавиатуры. Программа primer 3_7.c демонстрирует этот способ.

```
//primer 3_7.c
#include <stdio.h>
#include <conio.h>
#define n 5
 int main (){
  int i;
  int mas[n];
  for (i=0; i<n; i++)
  {
     printf ("input %d", i);
     printf (" element\n");
     scanf("%d",&mas[i]);
  }
  for (i=0; i<n; i++)
       printf (" %d ", mas[i]);
  getch ();
return (0);
}
```

3-й способ. Присвоить элементам массива начальные значения можно с использованием формулы. Частным случаем такого способа является заполнение массива с помощью генератора случайных чисел.

Чтобы получить случайное число, применяется функция rand, которая возвращает случайные значения, которые имеют диапазон от 0 и до установленной константы.

Вызов: *тип переменной имя = rand() % конечное число.*

Если необходимо сдвинуть интервал:

тип имя переменной =A + rand () % (B+1 – A),

где A – начальное число.

Перед вызовом функции rand() вызывают функцию srand(), обнуляющую системный таймер. Эти функции требуют подключения библиотек time (для работы с системным временем) и stdlib (библиотека содержит описание функций).

В программе primer 3_8.c одномерный массив из шести элементов заполняется случайными числами из интервала от 0 до 6.

```
//primer 3_8.c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#define n 6
int main ()
{
   int a[n], i;
   srand (time (NULL));
   for (i=0; i<n; i++)
        a[i]=rand()%7;
   printf ("\n");
   for (i=0; i<n; i++)
        printf ("%d ",a[i]);
   getchar ();
return (0);
}
```

Типовые алгоритмы обработки одномерных массивов

Существуют типовые алгоритмы обработки одномерных массивов. Приведем типовые программы на языке С.

Поиск элемента в упорядоченном массиве. Рассмотрим алгоритм быстрого поиска элемента в упорядоченной совокупности a_1, ….., a_n. При этом будем считать, что a_1, ….., a_n – целочисленный массив, b – некоторое целое число.

Пусть $a_1 < \ldots . < a_n$. Рассмотрим задачу: входит ли данное число b в массив a_1, ….., a_n, и если входит, то каково значение p, для которого $a_p = b$? Тривиальный алгоритм решения этой задачи основывается на последовательных сравнениях b с элементами a_1, ….., a_n. В этом случае среднее число требуемых сравнений можно считать равным n/2. Известен алгоритм, который требует гораздо меньших затрат.

Предположим, что в массиве a_1,..., a_n имеется элемент, равный b, т.е. существует такое p, что $a_p = b$. По результату любого сравнения

$a_s < b$ ($1 \le s \le b$) мы сразу определяем, лежит ли p в диапазоне от 1 до s или же в диапазоне от s + 1 до n: второе будет иметь место, если неравенство $a_s < b$ справедливо, а первое – если несправедливо. Если s находится примерно посередине между 1 и n, то сравнение $a_s < b$ будет сужать диапазон поиска примерно вдвое. Этот прием можно использовать многократно. Получается алгоритм, называемый алгоритмом деления пополам. В соответствии с этим алгоритмом надо взять первоначально 1 и n в качестве границ поиска индекса элемента; далее до тех пор, пока границы не совпадут, шаг за шагом сдвигать эти границы следующим образом: сравнить b с a_s где s – целая часть среднего арифметического границ, если $a_s < b$, то заменить прежнюю нижнюю границу на s + 1, оставив верхнюю границу без изменения, иначе оставить без изменения нижнюю границу, а верхнюю заменить на s. Поиск закончится, когда границы совпадут.

Сказанное можно записать в виде последовательности операторов (p и q – верхняя и нижняя границы), когда p и q совпадут, p дает результат выполнения.

Схематично процесс поиска для a_1,..., a_n соответственно равных 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 и b = 19 представлен на рис.1.

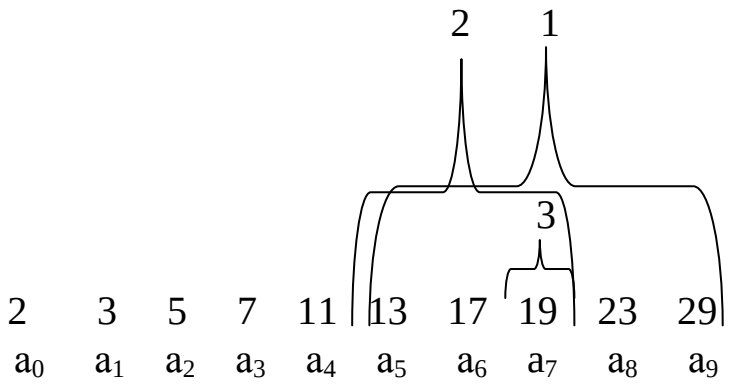


Рис. 1. Процесс поиска с использованием алгоритма деления пополам

Заметим следующее. Мы исходим из предположения, что среди элементов a_1,..., a_n имеется такой, который равен b. Если заранее неизвестно, имеется ли такой элемент, то, получив p, необходимо дополнительно проверить, действительно ли $a_p = b$. Если обнаружится, что равенство несправедливо, то из этого будет следовать, что среди a_1, …, a_n нет элемента, равного b. Программа primer 3_9.c демонстрирует применение этого алгоритма.

```
//primer 3_9.c
#include <stdio.h>
#include <conio.h>
#define n 10

int main ()
{     int i,p,q,s,b=19;
    int a[n]={2,3,5,7,11,13,17,19,23,29};
    p=1;
    q=n;
    while (p<q)
    {
        s=(p+q)/2;
        if (a[s]<b) p=s+1;
        else q=s;
    }
    if (a[p]==b)printf ("%d ", p);
    else printf ("now ");
    getch ();
return 0;
}
```

Упорядочивание (сортировка) массива. Под *сортировкой* будем понимать размещение набора данных в определенном порядке, что используется для облегчения поиска нужного элемента или группы элементов. От эффективности алгоритма сортировки зависит, например, производительность работы баз и банков данных.

Имеются три способа сортировки: выбором, вставками и обменом. Рассмотрим алгоритм *сортировки простыми вставками*. Алгоритм сортировки методом вставок предназначен для решения задачи упорядочивания совокупности попарно различных чисел (a_1, a_2,..., a_n). Опишем этот алгоритм применительно к упорядочиванию по возрастанию элементов одномерного массива.

При сортировке вставкой сначала упорядочиваются два элемента массива. Затем делается вставка третьего элемента в соответствующее место по отношению к первым двум элементам. Затем делается вставка четвертого элемента в список из трех элементов. Этот процесс повторяется до тех пор, пока все элементы не будут упорядочены. На рис. 2 продемонстрировано, как этот алгоритм работает для массива A[6] = (5, 2, 4, 6, 1, 3).

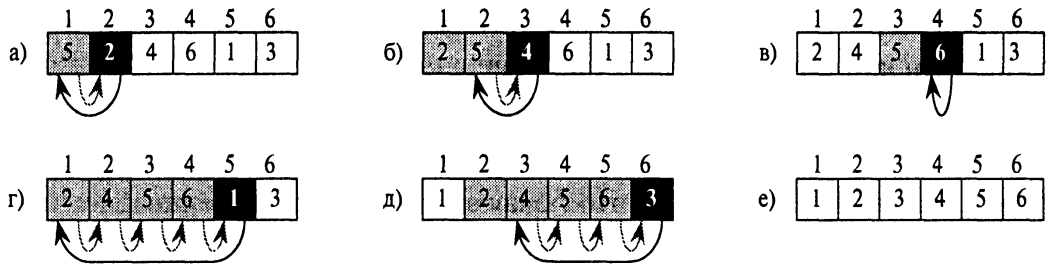


Рис. 2. Алгоритм сортировки простыми вставками

В алгоритме, работающем по методу вставок, применяется инкрементный подход: располагая отсортированным подмассивом A [l..j - 1], мы помещаем очередной элемент A[j] туда, где он должен находиться, в результате чего получаем отсортированный подмассив A [l..j]. В программе primer 3_10.c показана реализация этого способа сортировки.

```
//primer 3_10.c
#include <stdio.h>
#define n 6

int main ()
{
 int j, i, temp;
 int a[n]={5, 2, 4, 6, 1, 3};

 for (i=1; i<n; i++)
 {
   temp=a[i];
   for (j=i-1; j>=0; j--)
   if (temp<a[j])
     {
       a[j+1]=a[j];
       a[j]=temp;
   }
 }
 for (i=0; i<n; i++)
  printf( "%d ", a[i]);

getchar ();
return 0;
}
```

Метод «пузырька». Данный метод относится к сортировкам обменом. Метод «пузырька» получил своё название оттого, что продвижение максимальных элементов массива к его вершине происходит постепенно, подобно всплытию пузырька на поверхность воды. Этот метод требует нескольких проходов массива. На каждом проходе сравнивается пара соседних друг с другом элементов.

Если пара расположена в порядке возрастания, переставляем эти элементы местами. В противном случае элементы остаются на исходных позициях. Процедура должна быть повторена n-1 раз для гарантированного достижения результата. Пример поэтапной работы алгоритма показан на рис. 3. Изображенные на каждом проходе перестановки должны выполняться последовательно слева направо.

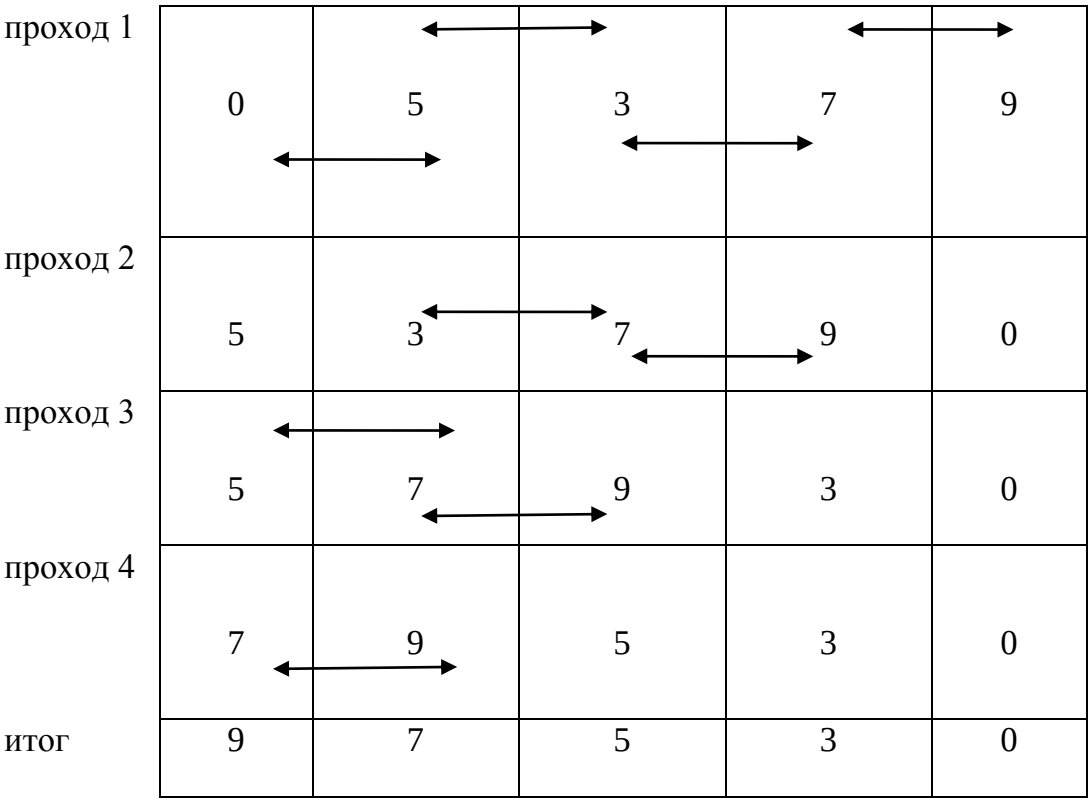


Рис. 3. Алгоритм сортировки «пузырьком»

Листинг программы primer 3_11.c демонстрирует реализацию этого метода.

```
//primer 3_11.c
#include <stdio.h>
#include <conio.h>
#define n 5

int main ()
{
 int i,j, temp=0;
 int a[n]={0,5,3,7,9};
 for (i=0; i<n-1; i++)
 {
   for (j=0; j<n-1; j++)
    if (a[j]<a[j+1])
      {
      temp=a[j];
      a[j]=a[j+1];
      a[j+1]=temp;
      }
  }
  printf ("\n");
for (i=0; i<n; i++)
   printf ("%d ", a[i]);
  getch ();
 return 0;
}
```

Сортировка выбором. Алгоритм состоит в том, что выбирается наименьший элемент массива и меняется местами с первым элементом, затем рассматриваются элементы, начиная со второго, и наименьший из них меняется местами со вторым элементом и так далее n-1 раз (при последнем проходе цикла при необходимости меняются местами предпоследний и последний элементы массива). Последовательность шагов при n = 5 изображена на рис. 4 ниже.

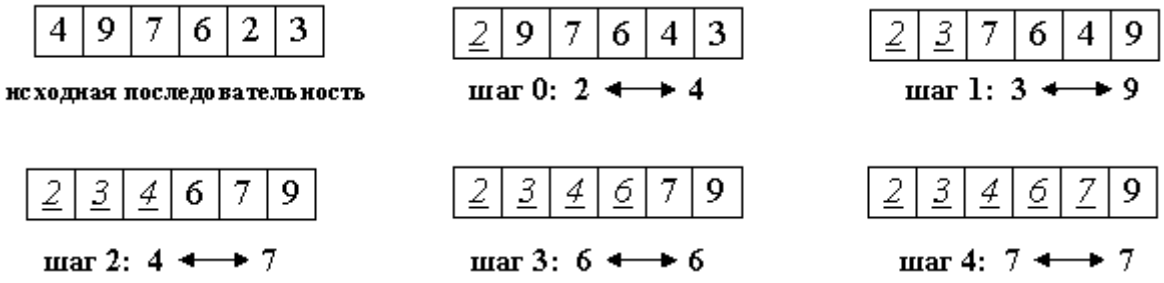


Рис. 4. Алгоритм сортировки выбором

Вне зависимости от номера текущего шага i последовательность a[0]...a[i] (выделена курсивом) является упорядоченной. Таким образом, на (n-1)-м шаге вся последовательность, кроме a[n], оказывается отсортированной, а a[n] стоит на последнем месте по праву: все меньшие элементы уже ушли влево.

Листинг программы primer 3_12.c демонстрирует реализацию этого метода:

```
//primer 3_12.c
#include <stdio.h>
#include <conio.h>
#define n 7

int main ()
{
 int a[n]={1,3,4,5,6,7,1};
 int i, j, index, temp;
 for (i=0; i<n-1; i++)
 {
   index=i;
   for (j=i+1; j<n; j++)
    if (a[j]<a[i])
        {
          index=j;
          temp=a[i];
          a[i]=a[index];
          a[index]=temp;}
        }
  for (i=0; i<n; i++)
      printf("\n%d ",a[i]);
getch();
    return (0);
}
```

Поиск минимального (максимального) элемента массива. Алгоритм поиска будет выглядеть следующим образом:

1. Будем считать минимальным (максимальным) элементом первый элемент массива.
2. Последовательно сравнить минимальный (максимальный) элемент с элементами массива, начиная от второго элемента. Если нашелся элемент меньший (больший) минимального (максимального), поместить его на место минимального (максимального) элемента. Ниже приведен соответствующий код программы.

```
//primer 3_13.c
#include <stdio.h>
#include <conio.h>
#define n 5

int main ()
{
  int min, i;
  int a[n]={4,2,9,1,3};
  min=a[0];
  for (i=1; i<n;i++ )
      if (a[i]<min) min=a[i];
  printf ("\n min=%d", min);
  getch ();
return 0;
}
```

Объединение двух одномерных массивов. Типовой алгоритм объединения двух одномерных массивов заключается в применении дополнительного массива, что и продемонстрировано в программе primer 3_14.c. Новая деталь в этой программе – это использование числа вместо имени переменной при задании размера массива.

```
//primer 3_14.c
#include <stdio.h>
#include <conio.h>

int main (){
    int i;
    int mas[10];
    int a[5]={4,2,9,1,3};
    int b[5]={5,7,8,11,12};
    for (i=0; i<5; i++)
    {
      mas[i]=a[i];
      mas [5+i]=b[i];
     }
    for (i=0; i<10; i++)
        printf ("%d ", mas[i]);
    getch ();
    return 0;
}
```

Циклический сдвиг элементов массива. При циклическом сдвиге вправо выталкиваемые элементы с конца массива заполняют освобождающиеся места в начале массива. Например, при сдвиге вправо на 3 разряда массива A (1, 2, 3, 4, 5, 6, 7) получаем массив A (5, 6, 7, 1, 2, 3, 4). Ниже представлена соответствующая программа.

```
//primer 3_15.c
#include <stdio.h>
#include <conio.h>
#include <locale.h >
#define n 7

int main ()
{
setlocale(LC_CTYPE, "russian");
int m, i, j;
int a[n]={1, 2, 3, 4, 5, 6, 7};
printf ("введите m\n");
scanf ("%d",&m);
m= m % n;
for (i=0; i<m; i++)
   {
   int temp=a[n-1];
   for (j=0; j<n-1; j++)
   {
      a[n-j-1]=a[n-j-2];
    }
   a[0]=temp;
}
for (i=0; i<n; i++)
   printf ("%d ",a[i]);
getch ();
return 0;
}
```

Типовые алгоритмы обработки одномерных массивов применяются для решения самых разнообразных задач. Продемонстрируем применение типовых алгоритмов для решения задачи «Преобразование последовательности»: задана последовательность, содержащая n

целых чисел (n >= 3). Необходимо найти число, которое встречается в этой последовательности наибольшее количество раз, а если таких чисел несколько, то найти минимальное из них и после этого переместить все такие числа в конец заданной последовательности. Порядок расположения остальных чисел должен остаться без изменения.

Например, последовательность 1, 2, 3, 2, 3, 1, 2 после преобразования должна превратиться в последовательность 1, 3, 3, 1, 2, 2, 2.

Требуется написать программу, которая решает названную задачу.

Выполним структурную декомпозицию программы. Анализ условия задачи показывает, что ее решение можно разбить на следующие подзадачи:

1. Основная программа: ввод - вывод элементов массива.
2. Нахождение в одномерном массиве последовательности наибольшей длины.
3. Сдвиг элементов одномерного массива.

Подзадача «Нахождение в одномерном массиве последовательности наибольшей длины» разбивается на две подзадачи: сортировка и подсчет количества вхождений.

Подзадача «Сдвиг элементов массива» является типовой и не требует разбивки на подзадачи. Следовательно, на первом шаге декомпозиции с использованием метода пошаговой детализации получаем (рис. 5).

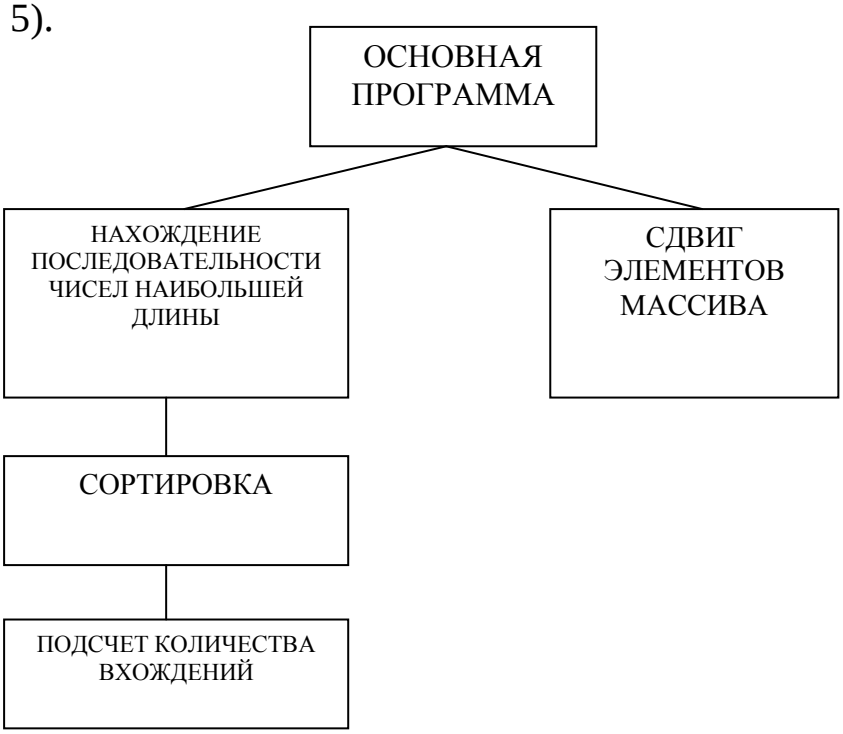


Рис. 5. Структурная декомпозиция программы

Первая подзадача. Так как размер массива задается с клавиатуры и значения элементов массива не заданы, выбираем способ заполнения массива – ввод с клавиатуры.

Вторая подзадача «Нахождение последовательности чисел наибольшей длины» разбивается на две типовые подзадачи:

1) сортировка одномерного массива;

2) нахождение последовательности чисел наибольшей длины.

Первая подзадача. Используем любую известную сортировку, например сортировку пузырьком.

Вторая подзадача. Поиск числа, входящего в последовательность наибольшее количество раз. *Идея решения:* перебираем элементы массива, сравнивая два соседних элемента. Если они равны, в переменной k накапливаем длину и сохраняем в переменной max. Как только числа стали неравны, содержимое переменной k = 1. Так как массив отсортирован по возрастанию, если найдется большее по значению число с таким же количеством вхождений, оно не заменит минимальное.

Третья подзадача. По окончании подсчета числа вхождений в переменной temp останется минимальное из чисел, входящих в последовательность наибольшее количество раз. Для сохранения порядка следования элементов, не равных temp, можно использовать другой массив размера исходного массива. Используем для этой цели массив a. *Идея решения*: просматривать элементы массива b, если элемент не равен temp, записываем его в массив a и подсчитываем количество этих элементов для того, чтобы с индекса k+1 поставить элемент, хранящийся в temp. Блок-схема алгоритма программы представлена на рис. 6.

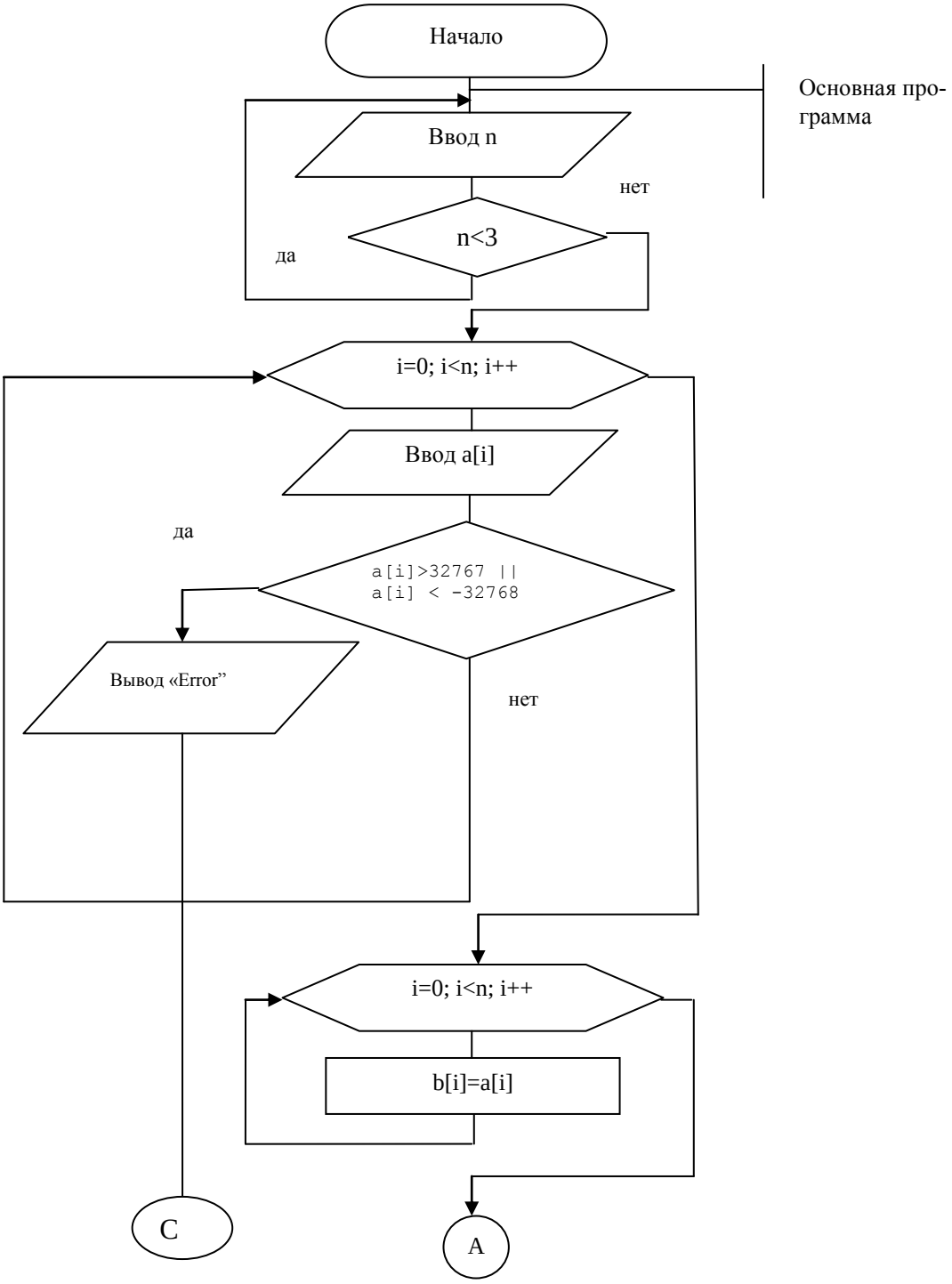


Рис. 6. Алгоритм программы примера (см. также с. 49, 50)

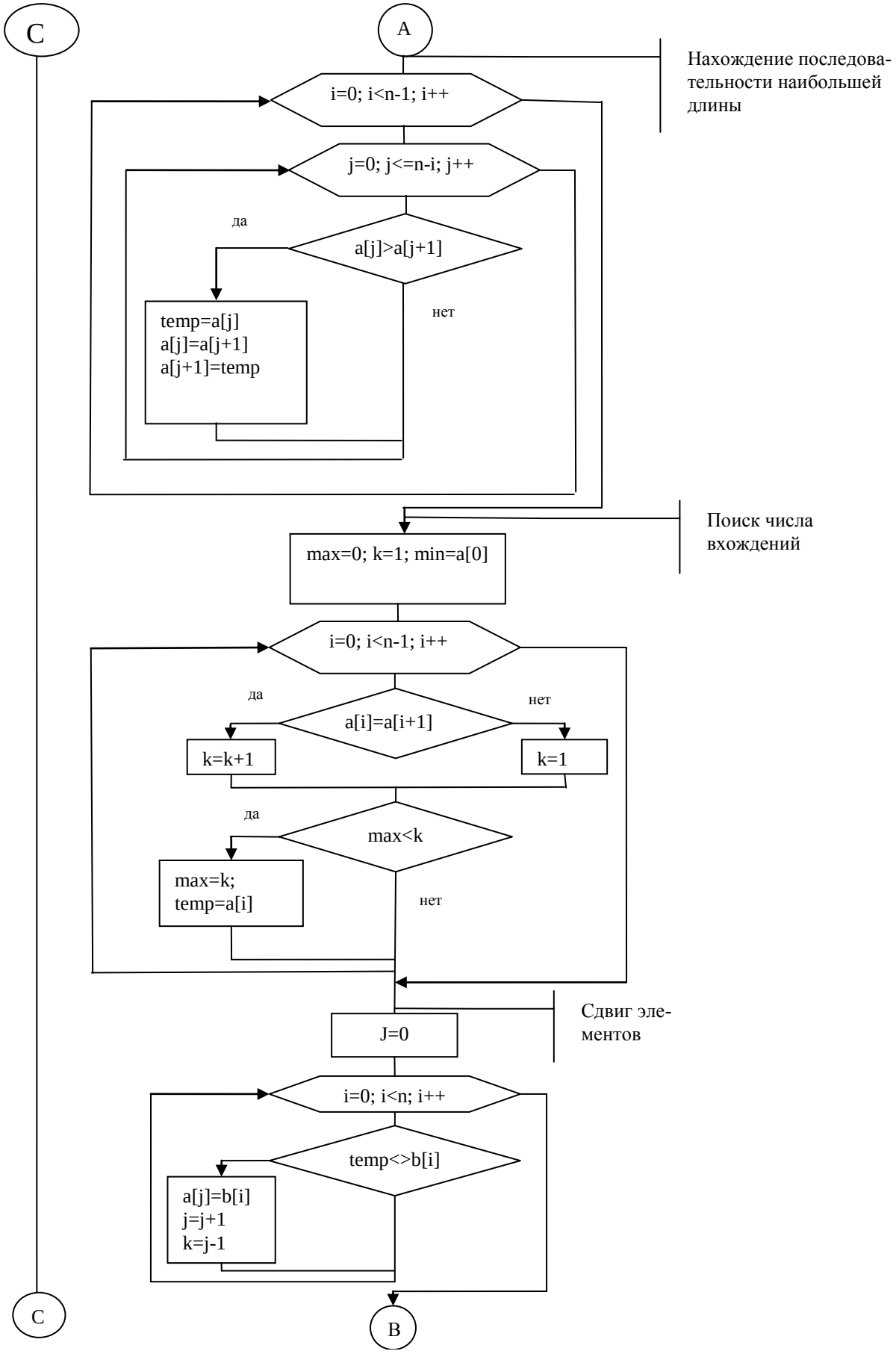


Рис. 6. Продолжение

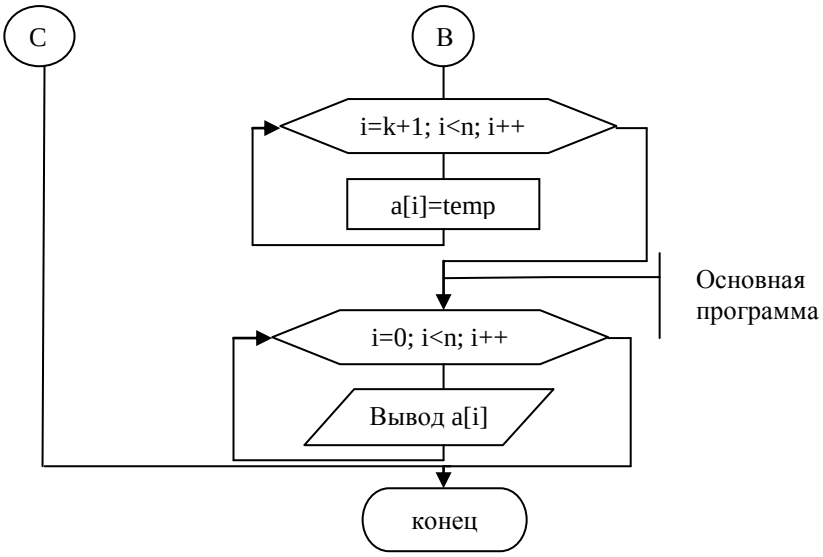


Рис. 6. Окончание

Полный текст программы приведен ниже:

```
//primer 3_16.c
#include <stdio.h>
#include <conio.h>
#define m 100  //резервируем память под 100
элементов массива

int main ()
{
  int a[m],b[m];
  int n, i, j, temp, max, k, min;

do {
         printf ("input n\n");
         scanf ("%d", &n);
         }
     while (n<3);

    printf ("\n input massiv\n");
    for (i=0; i<n; i++)
         {
           scanf("%d", &a[i]);
           if (a[i]>32767 || a[i]<-32768)
            printf ("Input Error\n");
            break;
        }
    for (i=0; i<n; i++)
        printf("%d ", a[i]);
```

```
//сохраним массив a в массиве b
    for (i=0; i<n; i++)
        b[i]=a[i];

/*Нахождение в одномерном массиве
последовательности наибольшей длины сортировка */
    for (i=0;i<n-1; i++)
    {
        for (j=1; j<=n-i; j++)
        if (a[j]>a[j+1] )
        {
           temp=a[j];
           a[j]=a[j+1];
           a[j+1]=temp;
        }
    }
    for (i=0; i<n; i++)
          printf("%d ", a[i]);
//поиск числа вхождений
max=0;k=1; min=a[0];
for (i=0; i<n-1; i++)
{
    if (a[i]==a[i+1])
        k=k+1;
    else k=1;
    if (max<k)
      {
        max=k;
        temp=a[i];
        }
}

//Сдвиг элементов одномерного массива
  j=0;
  for (i=0; i<n; i++)
    {
        if (temp!=b[i])
          {
               a[j]=b[i];
               j=j+1;
               k=j-1;
          }
    }
for (i=k+1; i<n; i++)  a[i]=temp;
```

```
//Основная программа
for (i=0;i<n;i++) printf ("%d ",a[i]);
getch ();
 return 0;
    }
```

Подберем для тестирования тестовые данные. Воспользуемся методологией эквивалентного разбиения[1]. Согласно этой методологии необходимо разработать набор «интересных» условий (класс эквивалентности), которые должны быть протестированы, и минимальный набор тестов, их проверяющий. Классы эквивалентности выделяются путем выбора каждого входного условия и разбиением его на две и более групп. Для проведения этой операции используют табл. 1, представленную ниже.

Таблица 1

Входные условия	Правильные классы эквивалентности	Неправильные классы эквивалентности

Где правильные классы эквивалентности – правильные входные данные программы, неправильные классы эквивалентности – входные значения, представляющие все другие возможные состояния условий. Существует ряд правил выделения классов эквивалентности:

- если входное условие описывает *область* значений (например «целое данное может принимать значения от 1 до 999»), то определяются один правильный класс эквивалентности (1 ≤ значение целого данного ≤ 999) и два неправильных (значение целого данного < 1 и значение целого данного > 999);
- если входное условие описывает *число* значений (например «в автомобиле могут ехать от одного до шести человек»), то определяются один правильный класс эквивалентности и два неправильных (ни одного и более шести человек);

[1] Майерс Г. Искусство тестирования программ / под ред. Б.А. Позина ; пер. с англ. М. : Финансы и статистика,1982. С. 64.

– если входное условие описывает *множество* входных значений и есть основание полагать, что каждое значение программа трактует особо (например «известны способы передвижения на АВТОБУСЕ, ГРУЗОВИКЕ, ТАКСИ, ПЕШКОМ или МОТОЦИКЛЕ»), то определяется правильный класс эквивалентности для каждого значения и один неправильный класс эквивалентности (например «НА ПРИЦЕПЕ»);
– если входное условие описывает ситуацию «должно быть» (например «первым символом идентификатора должна быть буква»), то определяется один правильный класс эквивалентности (первый символ – буква) и один неправильный (первый символ – не буква);
– если есть любое основание считать, что различные элементы класса эквивалентности трактуются программой неодинаково, то данный класс эквивалентности разбивается на меньшие классы эквивалентности.

С учетом сказанного определим классы эквивалентности для нашей задачи (табл. 2).

Таблица 2

Входные условия	Правильные классы эквивалентности	Неправильные классы эквивалентности
Количество целых чисел	n >= 3 (1)	n < 3 (2)
Значения области изменения входных переменных	-32 768 $\leq$ a[i] $\leq$ 32767 (3)	a[i] < -32768 (4) a[i] > 32767 (5)
Наличие групп чисел, повторяющихся разное количество раз	Имеются группы чисел, повторяющихся разное количество раз (6)	Имеются группы чисел, повторяющихся одинаковое количество раз (7), и все числа повторяются разное количество раз (8)

Согласно определенным нами классам эквивалентности необходимо покрыть тестами 6 случаев (табл. 3).

Таблица 3

№ п/п	Тестовые данные	Результат	Проверяемые классы
1	n = 8 1, 2, 3, 2, 3, 1, 2, 3	1 3 3 1 3 2 2 2	(1), (7)
2	n = 2	Неправильные входные данные	(2)
3	n = 5 -32768 1 2 2 -32768	1 2 2 -32768 -32768	(3)
4	n = 4 -697674 844170 717982 697674	Неправильные данные	(4), (5)
5	n = 8 4 6 6 6 2 2 9 1	4 2 2 9 1 6 6 6	(6)
6	n = 5 2 6 1 9 3	1 2 3 6 9	(8)

2. Порядок выполнения

В вариантах заданий под вставкой числа n в массив после k-го элемента следует понимать:

1) увеличение количества элементов массива на 1, при этом исходный размер массива должен это допускать;
2) смещение всех элементов, начиная с (k+1)-го на одну позицию вправо;
3) присваивание (k+1)-му элементу массива значения n.

Для выполнения задания:

1. Разработайте структурную схему и выполните детализацию алгоритмов модулей к задаче индивидуального задания без использования функции.
2. Напишите программы на языке С для разработанного алгоритма решения задачи.
3. Выполните отладку и компиляцию программы, получите исполняемые файлы.
4. Выполните тестирование программы.

3. Варианты заданий[2]

Вариант 1

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1) сумму отрицательных элементов массива;
2) произведение элементов массива, расположенных между максимальным и минимальным элементами.

Упорядочить элементы массива по возрастанию.

Вариант 2

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1) сумму положительных элементов массива;
2) произведение элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами.

Упорядочить элементы массива по убыванию.

Вариант 3

В одномерном массиве, состоящем из n целых элементов, вычислить:

1) произведение элементов массива с четными номерами;
2) сумму элементов массива, расположенных между первым и последним нулевыми элементами.

Преобразовать массив таким образом, чтобы сначала располагались все положительные элементы, а потом — все отрицательные (элементы, равные 0, считать положительными).

Вариант 4

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1) сумму элементов массива с нечетными номерами;
2) сумму элементов массива, расположенных между первым и последним отрицательными элементами.

Сжать массив, удалив из него все элементы, модуль которых не превышает 1. Освободившиеся в конце массива элементы заполнить нулями.

---

[2] Использованы материалы : Павловская Т.А., Щупак Ю.А. С/С++. Структурное программирование. СПб. : Питер, 2003.

Вариант 5

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1) минимальный элемент массива;
2) сумму элементов массива, расположенных между первым и последним положительными элементами.

Преобразовать массив таким образом, чтобы сначала располагались все элементы, равные нулю, а потом – все остальные.

Вариант 6

В одномерном массиве, состоящем из n целых элементов, вычислить:

1) номер максимального элемента массива;
2) произведение элементов массива, расположенных между первым и вторым нулевыми элементами.

Преобразовать массив таким образом, чтобы в первой его половине располагались элементы, стоявшие в нечетных позициях, а во второй половине – элементы, стоявшие в четных позициях.

Вариант 7

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1) номер минимального элемента массива;
2) сумму элементов массива, расположенных между первым и вторым отрицательными элементами.

Преобразовать массив таким образом, чтобы сначала располагались все элементы, модуль которых не превышает 1, а потом – все остальные.

Вариант 8

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1) максимальный по модулю элемент массива;
2) сумму элементов массива, расположенных между первым и вторым положительными элементами.

Преобразовать массив таким образом, чтобы элементы, равные нулю, располагались после всех остальных.

Вариант 9

В одномерном массиве, состоящем из n целых элементов, вычислить:

1) минимальный по модулю элемент массива;

2) преобразовать массив таким образом, чтобы в первой его половине располагались элементы, стоявшие в нечетных позициях, а во второй половине элементы, стоявшие в четных позициях.

Вариант 10

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1) номер минимального по модулю элемента массива;
2) сумму модулей элементов массива, расположенных после первого отрицательного элемента.

Сжать массив, удалив из него все элементы, величина которых находится в интервале [a, b]. Освободившиеся в конце массива элементы заполнить нулями.

Вариант 11

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1) номер максимального по модулю элемента массива;
2) сумму элементов массива, расположенных после первого положительного элемента.

Преобразовать массив таким образом, чтобы сначала располагались все элементы, целая часть которых лежит в интервале [a, b], а потом – все остальные.

Вариант 12

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1) количество элементов массива, лежащих в диапазоне от A до B;
2) сумму элементов массива, расположенных после максимального элемента.

Упорядочить элементы массива по убыванию модулей элементов.

Вариант 13

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1) количество элементов массива, равных 0;
2) сумму элементов массива, расположенных после минимального элемента.

Упорядочить элементы массива по возрастанию модулей элементов.

Вариант 14

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1) количество элементов массива, больших C;
2) произведение элементов массива, расположенных после максимального по модулю элемента.

Преобразовать массив таким образом, чтобы сначала располагались все отрицательные элементы, а потом – все положительные (элементы, равные 0, считать положительными).

Вариант 15

В одномерном массиве, состоящем из n вещественных элементов, вычислить:

1) количество отрицательных элементов массива;
2) сумму модулей элементов массива, расположенных после минимального по модулю элемента.

Заменить все отрицательные элементы массива их квадратами и упорядочить элементы массива по возрастанию.

Вариант 16

В одномерном массиве, состоящем из n целых элементов, вычислить:

1) количество положительных элементов массива;
2) сумму элементов массива, расположенных после последнего элемента, равного нулю.

Преобразовать массив таким образом, чтобы сначала располагались все элементы, целая часть которых не превышает 1, а потом – все остальные.

Лабораторная работа № 5

ДВУМЕРНЫЕ МАССИВЫ

1. Цель работы

Приобретение обучающимися практических умений и навыков применения типовых алгоритмов обработки двумерных массивов.

Многомерные массивы задаются указанием каждого измерения в квадратных скобках, например, оператор int mass [6][8] задает описание двумерного массива из 6 строк и 8 столбцов. В памяти такой массив располагается в последовательных ячейках построчно. Для доступа к элементу многомерного массива указываются все его индексы, например mass [5][3]. Аналогом двумерного массива являются квадратные и прямоугольные таблицы, которые часто называют *матрицами.*

Способы инициализации многомерного массива.

а) *С помощью оператора присваивания.* При инициализации многомерного массива он представляется либо как массив из массивов, при этом каждый массив заключается в свои фигурные скобки (в этом случае левую размерность при описании можно не указывать), либо задается общий список элементов в том порядке, в котором элементы располагаются в памяти:

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

б) *С помощью генератора случайных чисел*

```
 int a[n] [m], i,j;
 srand (time (NULL));
 for (i=0; i<n; i++)
    for (  j=0; j<m; j++)
   a[i][j]=5+rand()% (25+1-5); // интервал от 5 до 25
```

в) *Ввод значений с клавиатуры*

```
int a[n] [m], i, j;
  for (i=0; i<n; i++)
    for (  j=0; j<m; j++)
         scanf (“%d”, &a[i][j]);
```

Прямоугольные матрицы. Матрицы, в которых число строк не равно числу столбцов, называются прямоугольными. Перечислим некоторые действия, которые можно выполнять над матрицами:

а) суммой однотипных матриц A и B называют матрицу C, каждый элемент которой равен сумме соответствующих элементов матриц A и B;

б) разностью матриц A и B называют матрицу C, каждый элемент которой равен разности соответствующих элементов матриц A и B;

в) произведением двух матриц A и B называется такая матрица C, у которой элементы определяются по формуле $C_{ij} = \sum_{k=1}^{n} a_{ik}\, b_{kj}$, где i=1….m, j=1….p. То есть нужно перемножить соответствующие элементы i-й строки матрицы A на элементы j-го столбца матрицы B и полученные произведения сложить. Примечание: число столбцов матрицы A должно равняться числу строк матрицы B.

Квадратные матрицы. Матрицы, в которых число строк равно числу столбцов, называются квадратными. Перечислим основные свойства квадратных матриц.

1. Квадратные матрицы имеют главную и побочную диагонали. Рассмотрим массив A[4][4]. Если:

A[0,0]	A[0,1]	A[0,2]	**A[0,3]**
A[1,0]	**A[1,1]**	**A[1,2]**	A[1,3]
A[2,0]	**A[2,1]**	**A[2,2]**	A[2,3]
A[3,0]	A[3,1]	A[3,2]	**A[3,3]**

Побочная диагональ

Главная диагональ

i = j элементы расположены на главной диагонали;
i > j элементы расположены ниже главной диагонали;
i < j элементы расположены выше главной диагонали;
i ≥ j элементы расположены на главной диагонали и ниже;
i ≤ j элементы расположены на главной диагонали и выше;
i + j = n - 1элементы расположены на побочной диагонали;
i + j < n - 1 элементы расположены над побочной диагональю;
i + j > n - 1 элементы расположены под побочной диагональю.

2. Квадратная матрица, у которой все элементы, исключая элементы главной диагонали, равны нулю, называется диагональной матрицей:

$$D=\begin{matrix}4 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 3\end{matrix}$$

3. Диагональная матрица, у которой все элементы, стоящие на главной диагонали, равны 1, называется единичной матрицей.

$$D=\begin{matrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{matrix}$$

4. Если в матрице A(m, n) поменять местами строки и столбцы, то получится матрица A^t(m, n), которая называется транспонированной.

$$A=\begin{matrix}a_{00} & a_{01} & a_{0\,n-1}\\ a_{1\,0} & a_{1\,1} & a_{1\,n-1}\\ a_{m-1\,0} & a_{m\,1} & a_{m-1\,n-1}\end{matrix}$$

$$A^t=\begin{pmatrix}a_{00} & a_{1\,0} & a_{m-1\,0}\\ a_{0\,1} & a_{11} & a_{m\,1}\\ a_{0\,n-1} & a_{1\,n-1} & a_{m-1\,n-1}\end{pmatrix}$$

Приведем типовые алгоритмы обработки матриц на языке С:

1. *Сумма элементов столбца (строки)*

```
// primer 3_17.c
#include <stdio.h>
int main ()
{
  int i,j;
```

```
  int a[3][3]={{1,2},{2,2},{3,5}};
   for (i=0; i<3; i++){
     for (j=0; j<3; j++)
        printf("%d% ",a[i][j]);
     printf ("\n");
     }
for (i=0; i<3; i++)
  {
    int s=0;
for (j=0; j<3; j++)
    s+=a[i][j];
 printf ("\n s=%d", s);
     }
getchar ();
  return (0);
}
```

2. *Перестановка строк (столбцов) матрицы*

На примере задач этого типа разберем, как необходимо выполнить лабораторную работу.

Под перестановкой строк (столбцов) матрицы понимается упорядочивание (перестановка) строк (столбцов) согласно какому-либо условию.

Типовой алгоритм перестановки заключается в следующем:

1) поиск номера k-й строки (столбца) с элементом, отвечающим заданному условию;
2) перестановка k-й строки (столбца) с i-й строкой (столбцом) матрицы;
3) вывод строк (столбцов) упорядоченной матрицы.

Рассмотрим задачу упорядочивания строк целочисленной матрицы размера n*n, все элементы которой различны, по возрастанию первых элементов строк. На примере этой задачи рассмотрим, как выполнять лабораторную работу. Выполним структурную декомпозицию задачи (рис. 1).

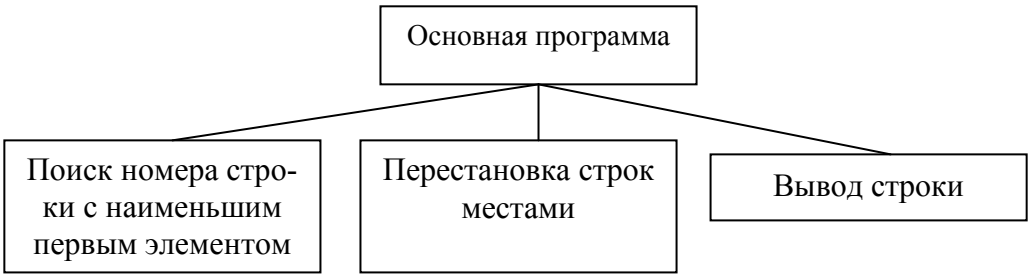


Рис. 1. Структурная декомпозиция программы примера

Схема алгоритма программы примера (рис. 2, 3):

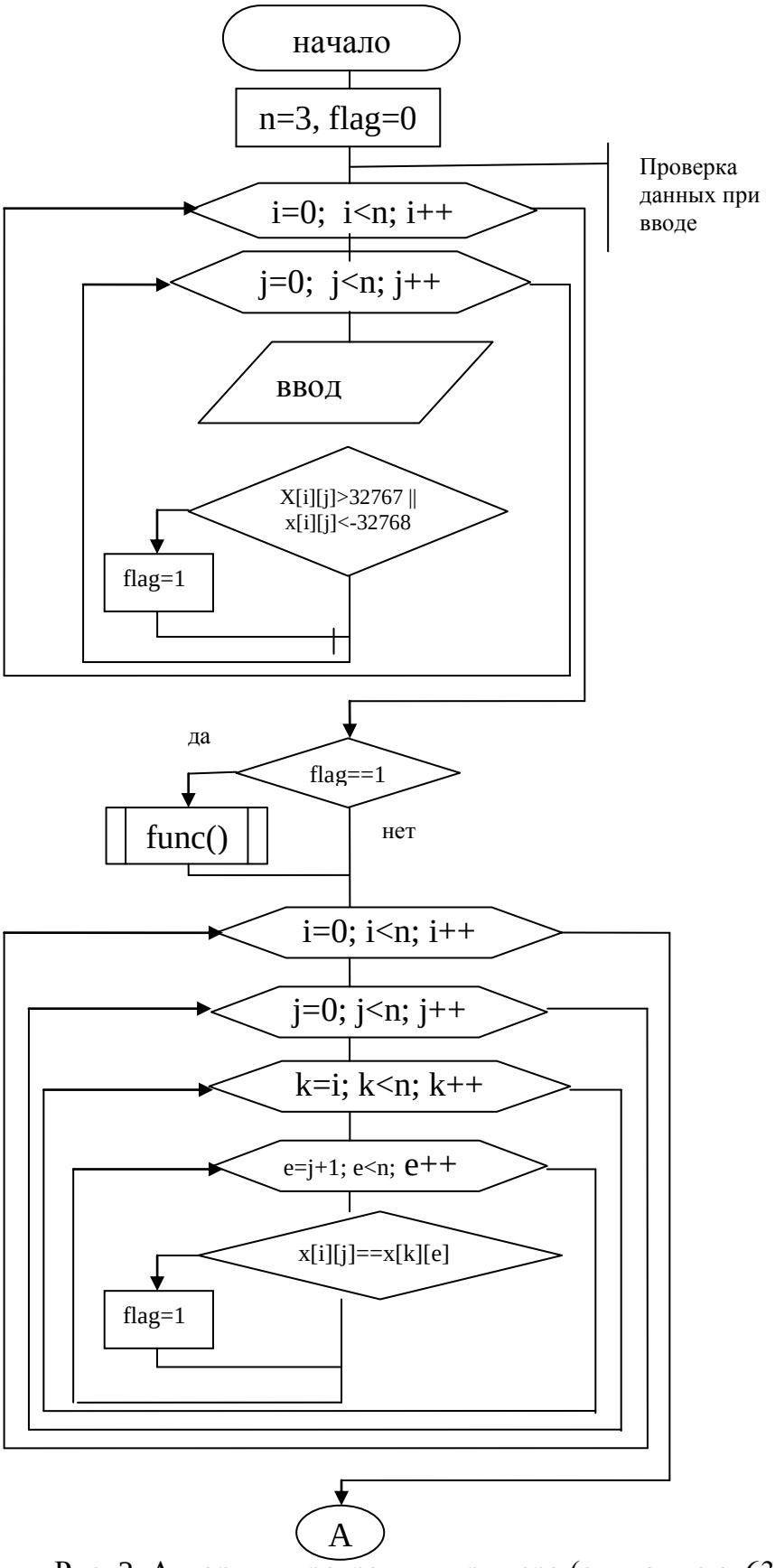


Рис. 2. Алгоритм программы примера (см. также с. 63)

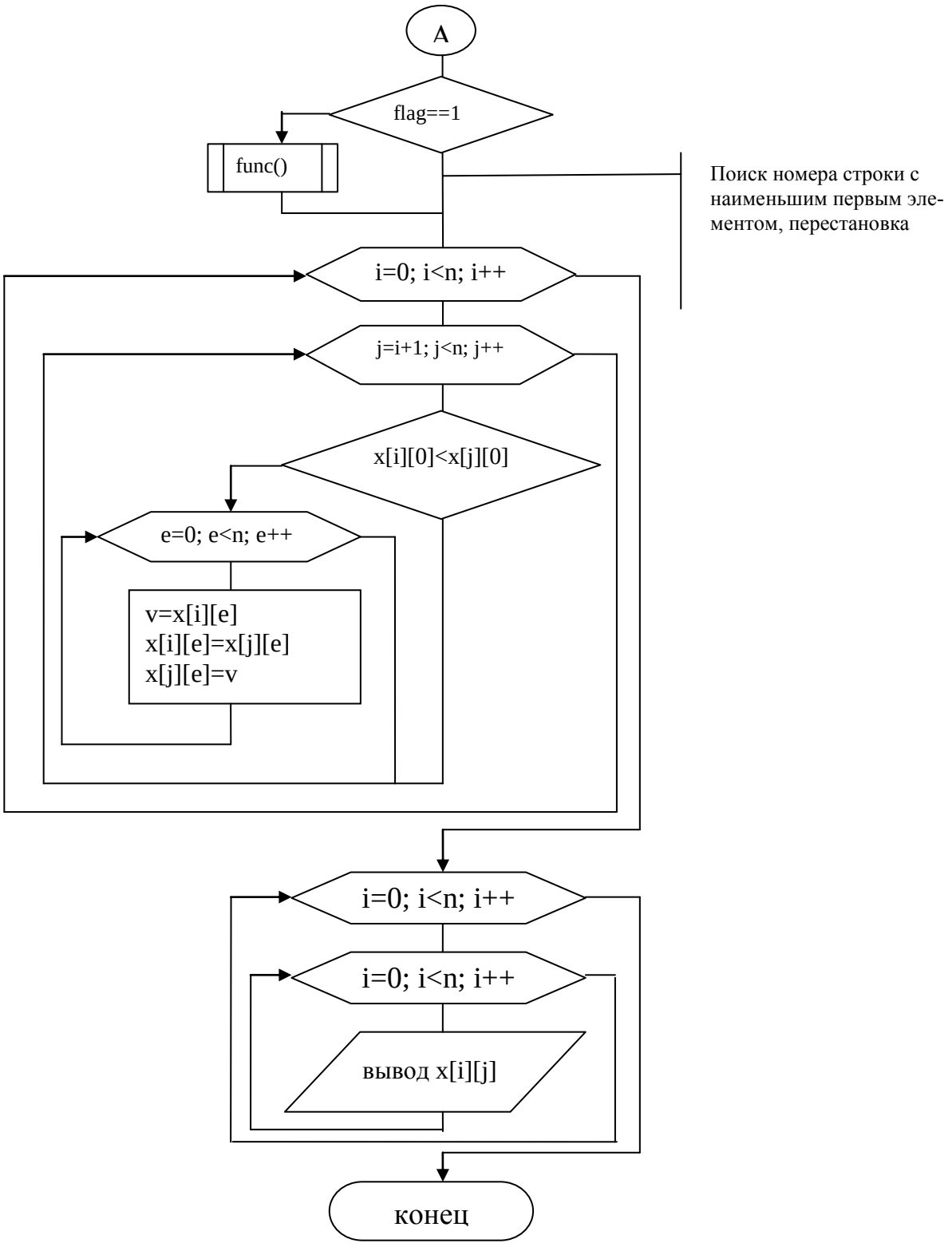


Рис. 2. Окончание

Подберем для тестирования тестовые данные. Как и в случае с одномерными массивами воспользуемся методологией эквивалентного разбиения. Определим классы эквивалентности для нашей задачи (табл. 1).

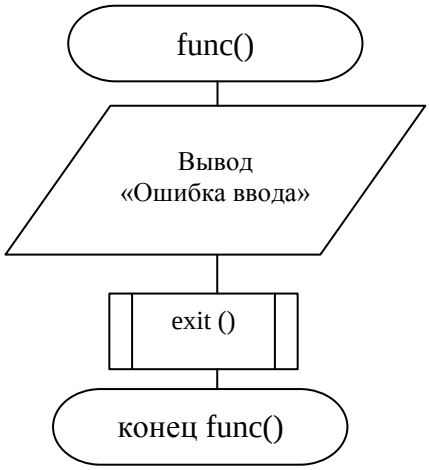


Рис. 3. Алгоритм функции

Таблица 1

Входные условия	Правильные классы эквивалентности	Неправильные классы эквивалентности
Значения элементов матрицы	Все элементы матрицы различны (1)	Не все элементы матрицы различны (2)
Значения области изменения входных переменных	-32 768≤x[i][j]≤32767 (3)	x[i][j]< -32768 (4) и x[i][j]>32767 (5)
Упорядочивание строк матрицы	Строки матрицы не упорядочены по первым элементам строк (6)	Строки матрицы упорядочены по первым элементам строк (7)

Согласно определенным нами классам эквивалентности необходимо покрыть тестами 7 случаев (табл. 2).

Таблица 2

№ п/п	Тестовые данные	Результат	Проверяемые классы
1	2 4 3 11 8 7 5 6 1	2 4 3 5 6 1 11 8 7	(1), (6)
2	2 2 4 5 6 1 11 8 7	Неправильные данные	(2)
3	2 4 3 11 -32770 7 5 6 65536	Неправильные данные	(4), (5)
4	-32768 2 7 5 6 1 32767 4 3	-32768 2 7 5 6 1 32767 4 3	(3)
5	11 8 7 2 4 3 5 6 1	2 4 3 5 6 1 11 8 7	(6)
6	5 6 1 11 8 7 2 4 3	2 4 3 5 6 1 11 8 7	(6)
7	2 4 3 5 6 1 11 8 7	2 4 3 5 6 1 11 8 7	(7)

```
// primer 3_18.c
#include <stdio.h>
#define n 3;

    void func()
      {
      printf ("Input Error\n");
      getch ();
      exit ();
     }
 int main ()
{
        int x[n][n];
    int i,j,k,v,e,flag=0;
```

```
  //проверка, нет ли значений, выходящих за границы

    printf ("input x[i][j]\n");
  for (i=0; i<n; i++)
  {
      for (j=0; j<n; j++)
      {
        scanf ("%d", &x[i][j]);
        if (x[i][j]>32767 || x[i][j]<-32768)
           flag=1;
      }
   }

  if(flag==1)
      func ();
 // проверка, нет ли одинаковых элементов в массиве
    flag=0;
   for (i=0; i<n; i++)
    for (j=0; j<n; j++)
    {
        for (k=i; k<n; k++)
        {
            for (e=j+1; e<n; e++)
             if (x[i][j]==x[k][e])
                   flag=1;
         }
    }

if(flag==1)
    func ();
 // вывод элементов массива, если не было ошибок при вводе
   for (i=0; i<n; i++)
    {
        for (j=0; j<n; j++)
          printf ("%d  ", x[i][j]);
      printf ("\n");
    }
    printf ("\n");
```

```
  // поиск номера строки с наименьшим первым
элементом, перестановка
     for (i=0; i<n-1; i++)
     {
         for (j=i+1; j<n; j++)
        if (x[i][0]<x[j][0])
          {
              for (e=0; e<n; e++)
             {
                v=x[i][e];
                x[i][e]=x[j][e];
                x[j][e]=v;
                }
          }
     }

// вывод упорядоченной матрицы
  for (i=0; i<n; i++)
  {
     for (j=0; j<n; j++)
       printf ("%d  ",x[i][j]);
    printf ("\n");
  }
getch ();
return (0);
}
```

3. Транспонирование матриц

При транспонировании матрицы элементы, расположенные на главной диагонали исходной и транспонированной матриц, одни и те же. То есть, транспонировать матрицу – значит зеркально отразить ее элементы относительно главной диагонали. Сделать это можно, введя новый массив, например как в программе primer 3_19.c

```
//primer 3_19.c
#include <stdio.h>
int main ()
{
    int i,j;
    int b[3][3];
    int a[3][3]={{1,2,3},{4,5,6}, {7,8,9}};
    for (i=0; i<3; i++)
    {
```

```
        for (j=0; j<3; j++)
          printf ("%d ", a[i][j]);
      printf ("\n");
    }
    printf ("\n");
for (i=0; i<3; i++)
  for (j=0; j<3; j++)
      b[i][j]=a[j][i];
for (i=0; i<3; i++)
    {
        for (j=0; j<3; j++)
          printf ("%d ", b[i][j]);

    printf ("\n");
    }

getchar ();
return (0);
}
```

4. Повороты матриц

Задачи такого типа встречаются очень часто. Рассмотрим метод их решения.

Пусть дана квадратная матрица A_{nn}, состоящая из целых чисел. Повернем ее на 90° по часовой стрелке. Для наглядности используем матрицу $A_{3,3}$:

$$A = \begin{pmatrix} a_{00}\, a_{01} a_{02} \\ a_{10}\, a_{11} a_{12} \\ a_{20} a_{21} a_{22} \end{pmatrix}$$

Матрица после поворота

$$\mathrm{A}' = \begin{pmatrix} a_{20}\, a_{10} a_{00} \\ a_{21}\, a_{11} a_{01} \\ a_{22} a_{12} a_{02} \end{pmatrix}$$

Установим соответствие между элементами матриц A и A'. Из соответствия следует следующее отношение матриц A и A': A': $i = j', j + i' = n - 1 \rightarrow A'(i'j') = A(n - 1 - j, i)$.

Программа primer 3_20.c реализует рассмотренное отношение матриц.

```
//primer 3_20.c
#include <stdio.h>
#define n 3;

int main ()
{
    int i,j;
    int b[n][n];
    int a[n][n]={{1,2,3},{4,5,6}, {7,8,9}};
    for (i=0; i<n; i++)
    {
        for (j=0; j<n; j++)
          printf ("%d ", a[i][j]);
      printf ("\n");
    }
    printf ("\n");
     for (i=0; i<n; i++)
     {
         for (j=0; j<n; j++)
         {
             b[i][j]=a[n-1-j][i];
             printf ("%d ", b[i][j]);
         }
         printf ("\n");
      }
    getchar ();
    return 0;
}
```

2. Порядок выполнения

Для выполнения задания:

1. разработайте структурную схему и выполните детализацию алгоритмов модулей к задаче индивидуального задания без использования функции;
2. напишите программы на языке С для разработанного алгоритма решения задачи;
3. выполните отладку и компиляцию программы, получите исполняемые файлы;
4. выполните тестирование программы.

3. Варианты заданий[3]

Вариант 1

Характеристикой строки целочисленной матрицы назовем сумму ее положительных четных элементов. Переставляя строки заданной матрицы, расположить их в соответствии с ростом характеристик.

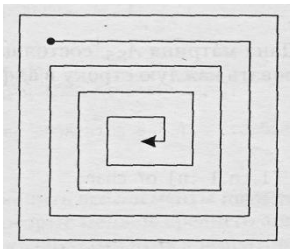
Вариант 2

Дана целочисленная прямоугольная матрица. Определить номер строки, в которой находится самая длинная серия одинаковых элементов.

Вариант 3

Дана целочисленная квадратная матрица. Определить максимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.

Вариант 4



Дана целочисленная квадратная матрица $A_{5,5}$. Вывести значения элементов на печать, выполнив обход матрицы по «спирали», как показано на рисунке:

[3] Использованы материалы: Павловская Т.А., Щупак Ю.А. С/С++. Структурное программирование. СПб. : Питер, 2003.

Вариант 5

Дана целочисленная прямоугольная матрица. Определить номера строк и столбцов всех седловых точек матрицы.

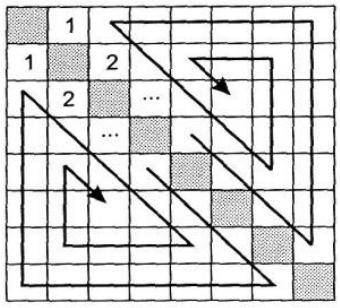
Примечание. Матрица A имеет седловую точку A_{ij}, если A_{ij} является минимальным элементом в i-й строке и максимальным в j-м столбце.

Вариант 6

Для заданной матрицы размером 8 × 8 найти такие k, чтобы k-я строка матрицы совпадала с k-м столбцом.

Вариант 7

Образовать два одномерных массива путем перезаписи в них элементов из заданного целочисленного двумерного массива размером n*n, при этом в один из формируемых массивов переписать все элементы, стоящие выше главной диагонали, а другой – ниже главной диагонали в порядке, указанном на рисунке.



Вариант 8

Соседями элемента A_{ij} в матрице назовем элементы A_{kj} с i-1 <= k <= i+1, j-1 <= l <= j+1, (k, l) ≠ (i, j). Операция сглаживания матрицы дает новую матрицу того же размера, каждый элемент которой получается как среднее арифметическое имеющихся соседей соответствующего элемента исходной матрицы. Построить результат сглаживания заданной вещественной матрицы размером (10 ×10).

Вариант 9

Элемент матрицы называется локальным минимумом, если он строго меньше всех имеющихся у него соседей. Подсчитать количество локальных минимумов заданной матрицы размером 10 × 10.

Вариант 10

Коэффициенты системы линейных уравнений заданы в виде прямоугольной матрицы. С помощью допустимых преобразований привести систему к треугольному виду.

Вариант 11

Уплотнить заданную матрицу, удаляя из нее строки и столбцы, заполненные нулями. Найти номер первой из строк, содержащих хотя бы один положительный элемент.

Вариант 12

Осуществить циклический сдвиг элементов прямоугольной матрицы на n элементов вправо или вниз (в зависимости от введенного режима), n может быть больше количества элементов в строке или в столбце.

Вариант 13

Осуществить циклический сдвиг элементов квадратной матрицы размером M × M вправо на k элементов таким образом: элементы 1-й строки сдвигаются в последний столбец сверху вниз, из него – в последнюю строку справа налево, из нее – в первый столбец снизу вверх, из него – в первую строку; для остальных элементов – аналогично.

Вариант 14

Дана целочисленная квадратная матрица. Определить:

1) сумму элементов в тех строках, которые не содержат отрицательных элементов;
2) минимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.

Вариант 15

Путем перестановки элементов квадратной вещественной матрицы добиться того, чтобы ее максимальный элемент находился в левом верхнем углу, следующий по величине – в позиции (1,1) и т.д., заполнив таким образом всю главную диагональ. Найдите номер первой из строк, не содержащей ни одного положительного элемента.

Вариант 16

Дан квадратный двумерный массив. Назовем столбец этого массива псевдоупорядоченным, если все элементы, стоящие «выше» диагонального (лежащего на главной диагонали), меньше либо равны ему, а все элементы, стоящие «ниже» диагонального, больше его. Проверить, все ли столбцы массива являются псевдоупорядоченными.

Лабораторная работа № 6

КОМБИНИРОВАННЫЙ ТИП ДАННЫХ

1. Цель работы

Приобретение обучающимися практических умений и навыков работы с типами, определяемыми пользователем.

При работе с массивами основное ограничение заключается в том, что каждый элемент должен иметь один и тот же тип данных. Однако при решении многих задач возникает необходимость хранить и обрабатывать совокупность данных различного типа как единое целое. Для этого применяется тип данных *структура*.

Структура является объединением данных различных типов (любого основного типа, массивом, указателем, объединением или структурой). Переменные, входящие в состав структуры, называются *полями* структуры. Структура определяет новый тип данных, который в дальнейшем можно использовать наряду со стандартными типами данных.

Определение структуры

```
struct  имя структуры
{
   члены структуры;
    }[список описателей];
```

Простая структура

Начнем рассмотрение со структуры, содержащей три поля, два из которых имеют целый тип и одно поле – вещественный тип. Эта структура предназначена для хранения информации о комплектующих деталях изделий, выпускаемых фабрикой. Компания производит несколько типов изделий, поэтому номер модели изделия включен в структуру как первое из ее полей. Номер самой детали представлен вторым полем, а ее стоимость – третьим полем.

```
struct part
{
    int modelnumber;
    int partnumber;
    float cost;
};
```

Определение структуры part необходимо для того, чтобы создавать на его основе переменные типа part.

Определение структурной переменной

```
int main (void)
{
    struct part p1;
. . . . . .
     return(0);
}
```

Первый оператор функции main () выглядит следующим образом: struct part p1; он представляет собой определение переменной p1, имеющей тип part. Определение переменной означает, что под эту переменную выделяется память. Правило: под структурную переменную всегда отводится столько памяти, сколько достаточно для хранения всех ее полей.

Доступ к полям структуры

Когда структурная переменная определена, доступ к ее полям возможен с применением *операции точки (доступ к полю структуры)*. В выражении на первом месте ставится имя структурной переменной, затем – операции точки, на третьем месте – имя поля. Например: p1. modelnumber

Типовые алгоритмы работы со структурами

1. *Инициализация полей структуры*

1-й способ. С помощью оператора присваивания

```
p1. modelnumber = 6244;
p1. partnumber = 373;
p1. cost = 217.55;
```

2-й способ. Ввод значений с клавиатуры

```
scanf( "%d", &p1. modelnumber);
```

3-й способ. Инициализация полей перечислением

```
struct part p1={6244, 373, 217.55};
```

2. Присваивание значений одной структурной переменной другой

```
struct part p1= {6244, 373, 217.55}, p2;
p2=p1;
printf (“model %d ”, p2. modelnumber);
printf (“part %d “, p2. partnumber);
printf (“cost %f “, p2. cost);
```

Результат работы программы:

```
model 6244 part 373 cost 217.55
```

3. Сравнение структурных переменных. Массивы структур

```
//primer 3_21.c
#include <stdio.h>
struct part
{
 int modelnumber;
 int partnumer;
}mas[5];

int main ()
{
    int i, max;
      printf ("vvedite\n");

for (i=0; i<5; i++)
       scanf("%d%d",&mas[i].modelnumber,
&mas[i].partnumer);

max=mas[0].partnumer;

for (i=1; i<5; i++)
          if(max<mas[i].partnumer)
max=mas[i].partnumer;
printf ("\npartnumber %d", max);
getchar();
return 0;
}
```

Битовые поля. Это особый вид полей структуры, которые обеспечивают доступ к отдельным битам памяти. Они используются для плотной упаковки данных, например флажков типа «да / нет». Минимальная адресуемая ячейка памяти – 1 байт, а для хранения флажка достаточно одного бита.

Вне структур битовые поля объявлять нельзя. Нельзя также организовывать массивы битовых полей и применять к полям операцию определения адреса. В общем случае тип структуры с битовым полем задается в следующем виде:

```
struct {unsigned идентификатор1: длина поля1;
unsigned идентификатор2: длина поля2;};
```

Битовые поля могут быть любого целого типа. Длина поля задается целым выражением или константой, которая определяет число битов, отведенное соответствующему полю. Поле нулевой длины обозначает выравнивание на границу следующего слова. Знаковые компоненты автоматически размещаются на соответствующие границы слов. Битовые поля можно использовать точно так же, как обычные поля структуры.

```
struct part
{
   unsigned a1: 1;

unsigned a2: 3;
} flag;
```

Рассмотрим задачу, на примере которой покажем, как выполнять лабораторную работу: на вход подаются сведения о сдаче экзаменов учениками 9-х классов некоторой средней школы. В первой строке сообщается количество учеников N, которое не меньше 10, но не превосходит 100, каждая из следующих N строк имеет такой формат:

<Фамилия> <Имя> <Оценки>, где <Фамилия> – строка, состоящая не более чем из 20 символов, <Имя> – строка, состоящая не более чем из 15 символов, <Оценки> – три целых числа, соответствующие оценкам по пятибалльной системе. Требуется написать программу, которая будет выводить на экран фамилии и имена трех худших по общему баллу учеников. Если среди остальных есть ученики, набравшие тот же общий балл, что и худший из трех, то следует вывести и их фамилии и имена. Структурная декомпозиция программы представлена на рис. 1.

Рис. 1. Структурная декомпозиция программы

Схема алгоритма программы примера представлена на рис. 2, 3

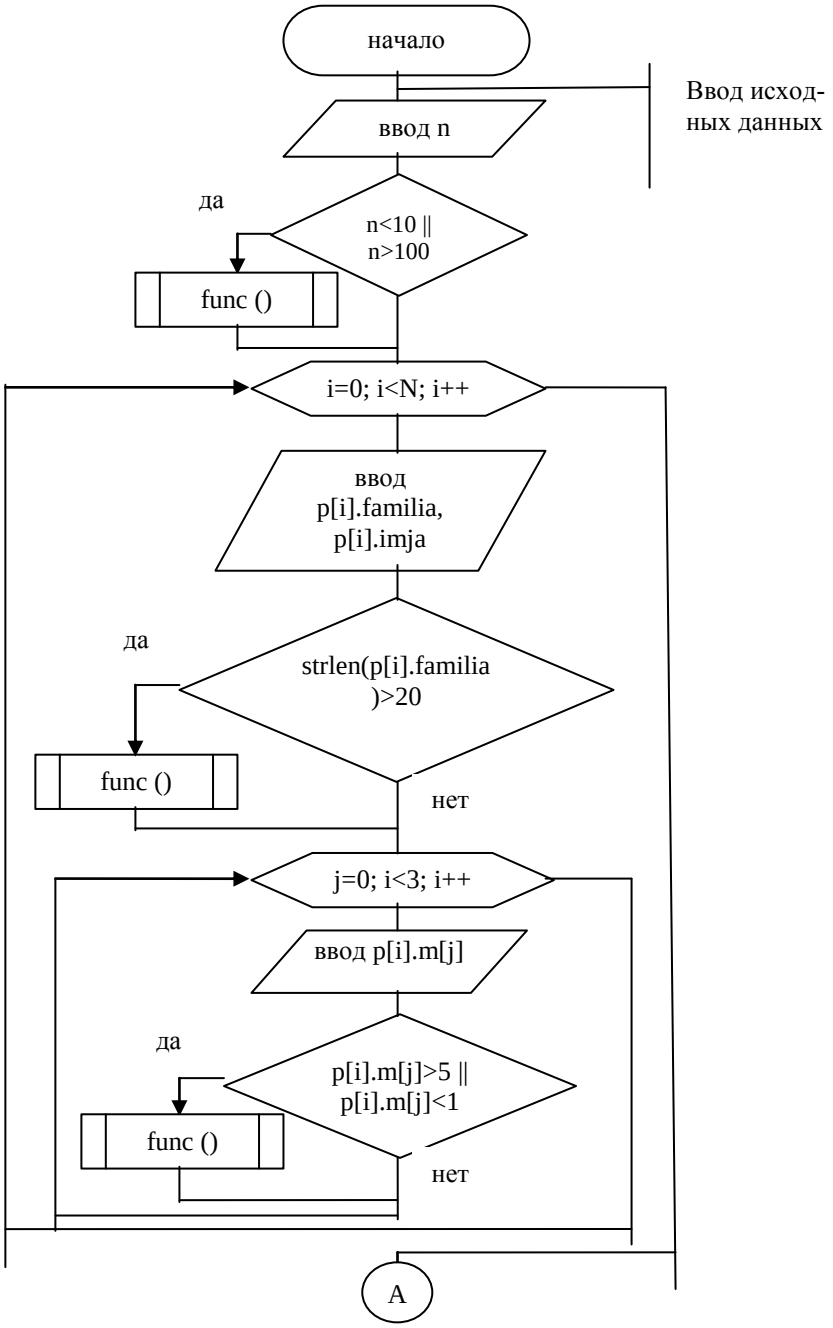


Рис. 2. Алгоритма программы примера (см. также с. 78)

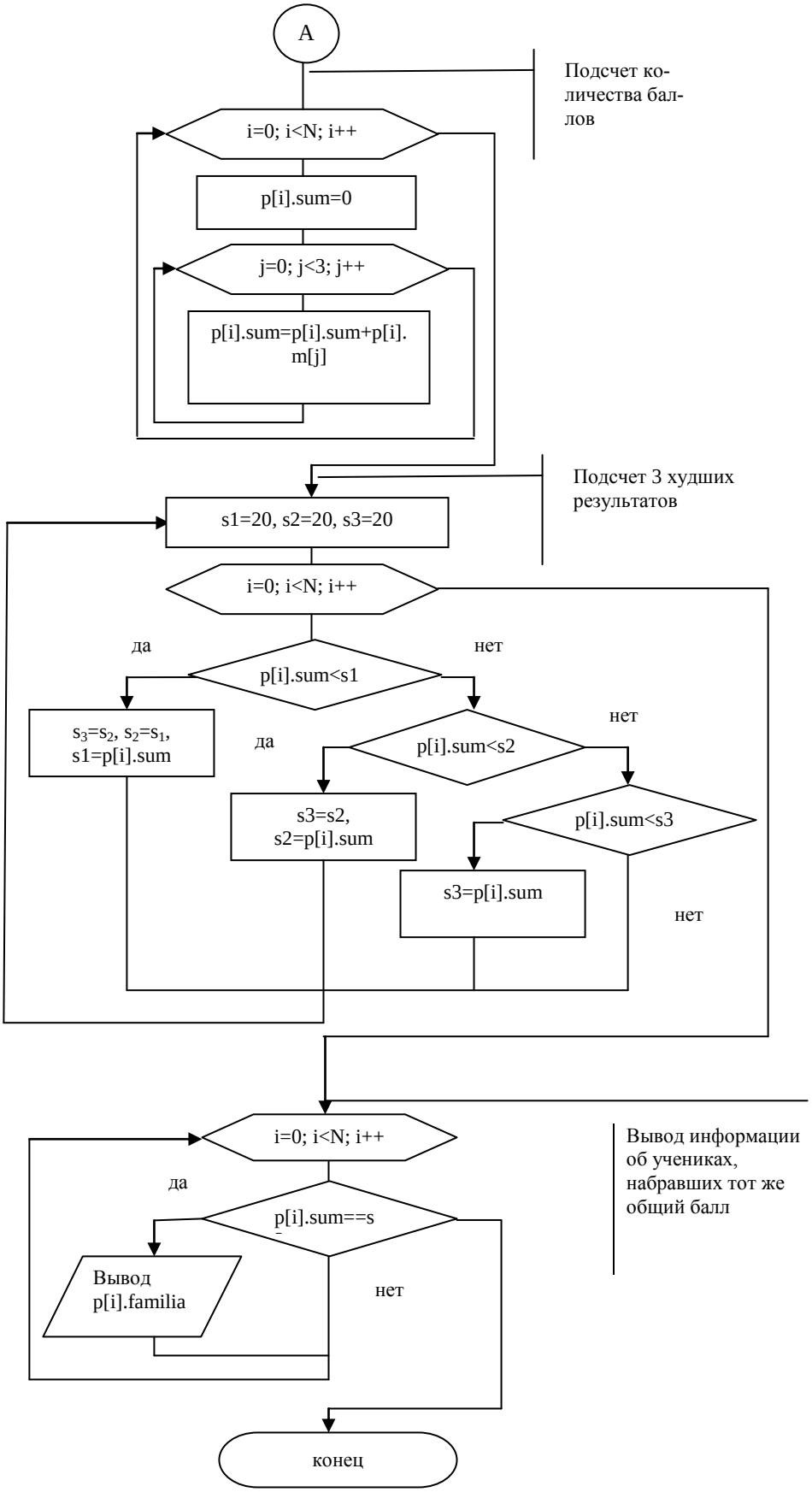


Рис. 2. Окончание

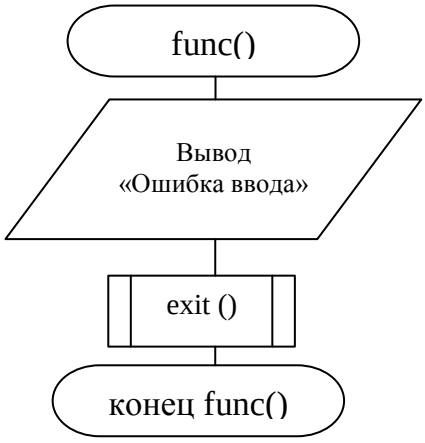


Рис. 3. Алгоритм функции

```
//primer 3_24.c

#include <conio.h>
#include <stdio.h>
#include <string.h>
void func()
      {
      printf ("Input Error\n");
      getch ();
      exit ();
     }
struct student
{
  char familia[20];
  char imja[15];
  short int m[3];
  int sum;
}p[100];
int main ()
{
    //введение исходных данных
    int n,i,j, s1, s2, s3;
    printf ("input N\n");
    scanf("%d", &n);
    if(n<10 || n>100) func ();
```

```
    for (i=0; i<n; i++)
    {
        printf ("input familia, imja\n");
        scanf ("%s", &p[i].familia);
        if (strlen (p[i].familia)>20)func ();
        scanf ("%s", &p[i].imja);
          if (strlen (p[i].imja)>15) func ();
        for (j=0; j<3; j++)
         {
             printf ("input evalution\n");
             scanf ("%d",&p[i].m[j]);
             if (p[i].m>5 || p[i].m<1) func ();
        }
    }
    //вычисление общего балла
    for (i=0; i<n; i++)
    {
        p[i].sum=0;
        for (j=0; j<3; j++)
        {
            p[i].sum=p[i].sum+p[i].m[j];
         }
    }
//поиск 3 худших результатов
    s1=20; s2=20; s3=20;
    for (i=0; i<n; i++)
    {
     if (p[i].sum<s1){s3=s2; s2=s1; s1=p[i].sum;}
        else
            if (p[i].sum<s2) {s3=s2; s2=p[i].sum;}
            else
                if (p[i].sum<s3) s3=p[i].sum;
    }

//поиск учеников,имеющих то же количество баллов,
что и худший из трех
for (i=0; i<n; i++)
        if (p[i].sum==s3)printf
("%s\n",p[i].familia);
    getch ();
    return 0;
    }
```

Подберем для тестирования тестовые данные. Воспользуемся методологией эквивалентного разбиения. Определим классы эквивалентности для нашей задачи (табл. 1).

Таблица 1

Входные условия	Правильные классы эквивалентности	Неправильные классы эквивалентности
Количество учеников	$10 \le n \le 100$ (1)	$n < 10$ (2) или $n > 100$ (3)
Количество символов в строке <Фамилия>	$0 < k1 \le 20$ (4)	$k1 > 20$ (5)
Количество символов в строке <Имя>	$0 < k2 \le 15$ (6)	$k1 > 15$ (7)
Строка <Оценки>	Три целых числа по пятибалльной системе (8)	Вводимые числа превышают 5 баллов (9) или меньше 1балла (10)

Согласно определенным нами классам эквивалентности необходимо покрыть тестами 2 случая (табл. 2).

Таблица 2

№ п/п	Тестовые данные	Результат	Проверяемые классы
1	n = 10 Артюшина Лариса 3 3 4 Амочкин Александр 4 5 2 Соколова Алла 3 3 3 Вуколова Алина 5 1 2 Смирнова Нина 1 2 3 Титова Оля 1 2 2 Гусева Нина 5 5 5 Гусенкова Елена 5 4 3 Симонов Виталий 1 4 2 Сухобоков Артем 1 1 2	Смирнова Нина Титова Оля Сухобоков Артем	(1), (4), (6), (8)
2	n = 2 Артюшина – Соколовская КЛЛЛЛЛЛЛЛЛЛариса 3 3 6 Амочкин Александр 4 5 0	Неправильные данные	(2), (5), (7), (9), (10)

Дополним разработанные тесты тестами, проверяющими логику работы программы. В приведенной выше программе ключевой является ситуация вычисления наихудших результатов. Необходимо протестировать случай, когда наихудший балл имеет один ученик (класс эквивалентности 12), и случай, когда наихудший балл имеют несколько учеников (класс эквивалентности 13). Первый случай покрывается тестом №1.

Разработаем тест для проверки второго случая (табл. 3).

Таблица 3

№ п/п	Тестовые данные	Результат	Проверяемые классы
3	n=10 Артюшина Лариса 3 3 4 Амочкин Александр 4 5 2 Соколова Алла 3 3 3 Вуколова Алина 5 1 2 Смирнова Нина 1 2 3 Титова Оля 1 2 2 Гусева Нина 5 5 5 Гусенкова Елена 5 4 3 Симонов Виталий 1 1 2 Сухобоков Артем 1 1 2	Смирнова Нина Титова Оля Симонов Виталий Сухобоков Артем	(13)

2. Порядок выполнения

Для выполнения задания:

1) разработайте структурную схему и выполните детализацию алгоритмов модулей к задаче индивидуального задания;
2) напишите программы на языке С для разработанного алгоритма решения задачи;
3) выполните отладку и компиляцию программы, получите исполняемые файлы;
4) выполните тестирование программы.

3. Варианты заданий[4]

Вариант 1

На вход программы подаются сведения об участниках массовки, пришедших на съемки фильма и получивших зарплату пропорционально отработанному времени. В первой строке задано текущее время начала съемки: через двоеточие два целых числа, соответствующие часам (от 00 до 23 – ровно 2 символа) и минутам (от 00 до 59 – ровно 2 символа). Во второй строке сообщается количество участников съемки N, которое не меньше 10, но не превосходит 1000. Каждая из следующих N строк имеет следующий формат: <Фамилия> <Время начала съемки>, где <Фамилия> – строка, состоящая не более чем из 20 символов, < Время начала съемки > – через двоеточие два целых числа, соответствующие часам и минутам. Сведения отсортированы в порядке времени начала съемки. Требуется написать программу, которая выведет фамилии участников массовки, которые после 6 часов съемок должны освободиться в хронологическом порядке.

Пример входных данных:

10: 00

3

Иванов 14:00

Петров 15: 00

Сидоров 11:30

Результат работы программы для этого примера

Петров

Иванов

Вариант 2

Описать структуру с именем STUDENT, содержащую следующие поля:

- фамилия и инициалы;
- номер группы;
- успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия:

[4] Использованы материалы: Павловская Т.А., Щупак Ю.А. С/С++. Структурное программирование : практикум. СПб. : Питер, 2003. 240 с.

- ввод с клавиатуры данных в массив, состоящий из десяти структур типа STUDENT; записи должны быть упорядочены по возрастанию среднего балла;
- вывод на дисплей фамилий и номеров групп для всех студентов, включенных в массив, если средний балл студента больше 4.0;
- если таких студентов нет, вывести соответствующее сообщение;
- вывод на дисплей фамилий и номеров групп для всех студентов, включенных в массив, имеющих оценки 4 и 5;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 3

На вход программы подаются сведения о номерах школ учащихся, участвовавших в районной олимпиаде по информатике. В первой строке сообщается количество учащихся N (N<=1000), каждая из следующих N строк имеет формат: <Фамилия> <Инициалы> <Номер школы>, где <Фамилия> – строка, состоящая не более чем из 20 символов, <Инициалы> – строка, состоящая из 4 символов (буква, точка, буква, точка), <Номер школы> – не более чем двузначный номер. Данные при вводе разделить одним пробелом. Пример входной строки: Иванов П. С. 57

Требуется написать программу, которая будет выводить на экран информацию, из какой школы было меньше всего участников (таких школ может быть несколько). При этом необходимо вывести информацию только по школам, пославшим хотя бы одного участника.

Вариант 4

Описать структуру с именем AEROFLOT, содержащую следующие поля:

- название пункта назначения рейса;
- номер рейса;
- тип самолета.

Написать программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив, состоящий из 7 элементов типа AEROFLOT; записи должны быть упорядочены по возрастанию номера рейса;
- вывод на экран номеров рейсов и типов самолетов, вылетающих в пункт назначения, название которого совпало с названием, введенным с клавиатуры;
- если таких рейсов нет, вывести соответствующее сообщение.

Вариант 5

На автозаправочных станциях (АЗС) продается бензин с маркировкой 92, 95 и 98. В городе N был проведен мониторинг цены бензина на различных АЗС. Напишите эффективную по времени работы и по используемой памяти программу, которая будет определять для каждого вида бензина, сколько АЗС продают его дешевле всего. На вход программы в первой строке подается число данных о стоимости бензина. В каждой из последующих N строк находится информация в следующем формате:

<Компания> <Улица> <Марка> <Цена>, где <Компания> – строка, состоящая не более чем из 20 символов без пробелов, <Улица> – строка, состоящая не более чем из 20 символов без пробелов, <Марка> – одно из чисел – 92, 95 или 98, <Цена> – целое число в диапазоне от 1000 до 3000, обозначающее стоимость одного литра бензина в копейках. <Компания> и <Улица>, <Улица> и <Марка>, а также <Марка> и <Цена> разделены ровно одним пробелом. Пример входной строки:

Синойл Цветочная 95 2250

Программа должна выводить через пробел 3 числа – количество АЗС, продающих дешевле всего 92-й, 95-й и 98-й бензин соответственно. Если бензин какой-то марки нигде не продавался, то следует вывести 0. Пример выходных данных:

12 1 0

Вариант 6

Описать структуру с именем STUDENT, содержащую следующие поля:

- фамилия и инициалы;
- номер группы;
- успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив, состоящий из десяти структур типа STUDENT; записи должны быть упорядочены по возрастанию номера группы;
- вывод на дисплей фамилий и номеров групп для всех студентов, включенных в массив, если средний балл студента больше 4.0;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 7

Вступительные экзамены в технический вуз состоят из трех экзаменов: физика (максимальный балл – 9), информатика (максимальный балл – 9), литература (максимальный балл – 5). На вход программы подаются сведения о сдаче этих экзаменов абитуриентами. В первой строке вводится количество абитуриентов N, во второй – количество мест К (К<N), на которые эти абитуриенты претендуют. Каждая из следующих N строк имеет такой формат: <Фамилия> <Оценка1> <Оценка2> <Оценка3>, где <Фамилия> – строка, состоящая не более чем из 20 символов, оценки – числа от 0 до максимальной оценки по предмету соответственно (ноль ставится в том случае, если экзамен не сдавался, например, после полученной на предыдущем экзамене тройки). Все баллы, большие 3, считаются удовлетворительными. Пример входных строк:

Иванов 8 9 4

Петров 3 0 0

Требуется написать программу, которая определяла бы по имеющимся данным количество абитуриентов, набравших полупроходной балл в данный вуз, или сообщала, что такой балл отсутствует (полупроходным называется такой балл, что лишь часть абитуриентов, набравших такой балл и не получивших ни одной неудовлетворительной оценки, попадает в К лучших, которые должны были быть зачислены на 1-й курс). Считается, что студенты, получившие только удовлетворительные оценки, обязательно присутствуют.

Вариант 8

На вход программы подаются сведения о пассажирах, желающих сдать свой багаж в камеру хранения на заранее известное время до полуночи. В первой строке сообщается число пассажиров N, которое не меньше 3, но не превосходит 1000; во второй строке – количество ячеек в камере хранения M, которое не меньше 10, но не превосходит 1000. Каждая из следующих N строк имеет такой формат:

<Фамилия> <Время сдачи багажа> <Время освобождения ячейки>,

где <Фамилия> – строка, состоящая не более чем из 20 непробельных символов; <Время сдачи багажа> – через двоеточие два целых числа, соответствующие часам (от 00 до 23 – ровно 2 символа) и минутам (от 00 до 59 – ровно 2 символа); <Время освобождения ячейки> имеет тот же формат.

<Фамилия> и <Время сдачи багажа>, а также <Время сдачи багажа> и <Время освобождения ячейки> разделены одним пробелом. Время освобождения больше времени сдачи.

Сведения отсортированы в порядке времени сдачи багажа. Каждому из пассажиров в камере хранения выделяется свободная ячейка с минимальным номером. Если в момент сдачи багажа свободных ячеек нет, то пассажир уходит, не дожидаясь освобождения одной из них.

Требуется написать программу, которая будет выводить на экран для каждого пассажира номер ему предоставленной ячейки (можно сразу после ввода данных очередного пассажира). Если ячейка пассажиру не предоставлена, то его фамилия не печатается.

Пример входных данных:

3
10
Иванов 09:45 12:00
Петров 10:00 11:00
Сидоров 12:00 13:12

Результат работы программы на этих входных данных:

Иванов 1
Петров 2
Сидоров 1

Вариант 9

Имеется список сотрудников организации с указанием их фамилии, имени и даты рождения. Администрация ежедневно поздравляет всех сотрудников, родившихся в этот день. Напишите программу, которая будет определять, в какой из дней года родилось больше всего сотрудников, и выводить этот день (или несколько дней). На вход программы в первой строке подается количество людей в списке N. Значение N может быть велико, например может быть больше 10.000. В каждой из последующих N строк находится информация в следующем формате: <Фамилия> <Имя> <Дата рождения>, где <Фамилия> – строка, состоящая не более чем из 20 символов без пробелов, <Имя> – строка, состоящая не более чем из 20 символов без пробелов, <Дата рождения> – строка, имеющая вид ДД.ММ.ГГГГ, где ДД – двузначное число от 01 до 31, ММ – двузначное число от 01 до 12, ГГГГ – четырехзначное число от 1800 до 2100. Пример входной строки: Иванов Сергей 27.03.1993. Программа должна вывести один

или несколько дней года (по одному в строке) в формате ДД. ММ, при этом можно не выводить начальный ноль в номере дня или месяца. Пример выходных данных:

27.3

Вариант 10

На вход программы подаются 365 строк, которые содержат информацию о среднесуточной температуре всех дней 2007 года. Формат каждой из строк следующий: сначала записана дата в виде dd.mm (на запись номера дня и номера месяца в числовом формате отводится строго два символа, день от месяца отделен точкой), затем через пробел записано значение температуры – число со знаком плюс или минус с точностью до одной цифры после десятичной точки. Данная информация отсортирована по значению температуры, т.е. хронологический порядок нарушен. Требуется написать программу, которая будет выводить на экран информацию о месяцах с максимальной среднемесячной температурой. Найденные максимальные значения следует выводить в отдельной строке для каждого месяца в виде: номер месяца, значение среднемесячной температуры, округленное до одной цифры после десятичной точки.

Вариант 11

После единых выпускных экзаменов по информатике в район пришла информация о том, какой ученик, какой школы, сколько набрал баллов. Районный методист решила выяснить номер школы, ученики которой набрали наибольший средний балл с точностью до целых. Программа должна вывести на экран номер такой школы и ее средний балл. Если наибольший средний балл набрало больше одной школы, вывести количество таких школ. Напишите программу, которая должна вывести на экран требуемую информацию. Также известно, что в районе школ с некоторыми номерами не существует. На вход программы сначала подается число учеников, сдававших экзамен. В каждой из следующих N строк находится информация об учениках в формате: <Фамилия> <Имя> <Номер школы> <Количество баллов>, где <Фамилия> – строка, состоящая не более чем из 30 символов без пробелов, <Имя> – строка, состоящая не более чем из 20 символов, <Номер школы> – число в диапазоне от 1 до 99, <Количество баллов> – число в диапазоне от 1 до 100. Эти данные записаны через пробел, то есть в каждой строке ровно 3 пробела.

Вариант 12

После единых выпускных экзаменов по информатике в район пришла информация о том, какой ученик, какой школы, сколько набрал баллов. Эта информация в том же виде была разослана в школы. Завуч школы №30 решила наградить двух учащихся, которые лучше всех сдали информатику. Программа должна вывести на экран фамилии и имена этих учеников. Если наибольший балл набрало больше двух человек – вывести количество таких учеников. Если наибольший балл набрал один человек, а следующий балл набрало несколько человек, нужно вывести только фамилию и имя лучшего. Напишите программу, которая должна вывести на экран требуемую информацию. Известно, что информатику сдавало больше 5 учеников школы №30. На вход программы сначала подается число учеников, сдававших экзамен. В каждой из следующих N строк находится информация об учениках в формате: <Фамилия> <Имя> <Номер школы> <Количество баллов>, где <Фамилия> – строка, состоящая не более чем из 30 символов без пробелов, <Имя> – строка, состоящая не более чем из 20 символов. <Номер школы> – число в диапазоне от 1 до 99, <Количество баллов> – число в диапазоне от 1 до 100. Эти данные записаны через пробел, то есть в каждой строке ровно 3 пробела.

Вариант 13

Описать структуру с именем TRAIN, содержащую следующие поля:

- название пункта назначения;
- номер поезда;
- время отправления.

Написать программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив, состоящий из восьми элементов типа TRAIN, записи должны быть размещены в алфавитном порядке по названиям пунктов назначения;
- вывод на экран информации о поездах, отправляющихся после введенного с клавиатуры времени;
- если таких поездов нет, выдать на дисплей соответствующее сообщение.

Вариант 14

В командных олимпиадах по программированию для решения предлагается не больше 11 задач. Команда может решать предложенные задачи в любом порядке. Подготовленные решения команда посылает

в единую проверяющую систему соревнований. Вам предлагается написать программу, которая будет статистически обрабатывать пришедшие запросы, чтобы определить наиболее популярные задачи. Следует учитывать, что количество запросов в списке может быть очень велико, так как многие соревнования проходят с использованием Интернет. На вход программы в первой строке подаётся количество пришедших запросов N. В каждой из последующих N строк записано название задачи в виде текстовой строки. Длина строки не превосходит 100 символов, название может содержать буквы, цифры, пробелы и знаки препинания.

Пример входных данных:

6
A+B
Крестики-Нолики
Прямоугольник
Простой делитель
A+B
Простой делитель

Программа должна вывести список из трёх наиболее популярных задач с указанием количества запросов по ним. Если в запросах упоминаются менее трех задач, то выведите информацию об имеющихся задачах. Если несколько задач имеют ту же частоту встречаемости, что и третья по частоте встречаемости задача, их тоже нужно вывести.

Пример выходных данных для приведённого выше примера входных данных:

A+B 2
Простой делитель 2
Крестики-Нолики 1
Прямоугольник 1

Вариант 15

На вход программы подаются сведения о сдаче экзаменов учениками 9-х классов некоторой средней школы. В первой строке сообщается количество учеников N, которое не меньше 10, но не превосходит

100, каждая из следующих N строк имеет следующий формат: <Фамилия> <Имя> <Оценки>, где <Фамилия> – строка, состоящая не более чем из 20 символов, <Имя> – строка, состоящая не более чем из 15 символов, <Оценки> – через пробел три целых числа, соответствующие оценкам по пятибалльной системе. <Фамилия> и <Имя>, а также <Имя> и <Оценки> разделены одним пробелом. Пример входной строки: Иванов Петр 4 2 4 Требуется написать наиболее эффективную программу, которая будет выводить на экран фамилии и имена неуспевающих учеников (имеющих по результатам экзаменов хотя бы одну двойку), располагая их в порядке уменьшения числа двоек.

Вариант 16

Описать структуру с именем MARSH, содержащую следующие поля:

- название начального пункта маршрута;
- название конечного пункта маршрута;
- номер маршрута.

Написать программу, выполняющую следующие действия:

- ввод с клавиатуры данных в массив, состоящий из восьми элементов типа MARSH; записи должны быть упорядочены по номерам маршрутов;
- вывод на экран информации о маршруте, номер которого введен с клавиатуры;
- если таких маршрутов нет, выдать на дисплей соответствующее сообщение.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Иванова, Г. С. Технология программирования : учебник для вузов / Г. С. Иванова. – М. : Изд-во МГТУ им. Н.Э. Баумана, 2002. – 320 с. – ISBN 5-7038-2077-4.

2. Майерс, Г. Искусство тестирования программ / Г. Майерс ; под ред. Б. А. Позина ; пер. с англ. – М. : Финансы и статистика, 1982. – 176 с.

3. Окулов, С. М. Основы программирования / С. М. Окулов. – М. : БИНОМ. Лаборатория знаний, 2005. – 440 с. – ISBN 5-94774-217-9.

4. Павловская, Т. А. C/C++. Структурное программирование : практикум / Т. А. Павловская, Ю. А. Щупак. – СПб. : Питер, 2003. – 240 с. – ISBN 5-94723-447-5.

5. Смит, Б. Методы и алгоритмы вычислений на строках / Б. Смит ; пер. с англ. – М. : И.Д. Вильямс, 2006. – 496 с. – ISBN 5-8459-1081-1 (рус.).

ПРИЛОЖЕНИЯ

Приложение 1

Часто используемые функции библиотеки stdio.h

Библиотека stdio.h	
Функция	Назначение
printf()	Вывод текста на экран
scanf()	Ввод с клавиатуры
fopen()	Открытие файла
fclose()	Закрытие файла
Библиотека <conio.h>	
getch ()	Считывает символ напрямую из консоли без использования буфера и echo-вывода
getche ()	Считывает символ напрямую из консоли без использования буфера, но с использованием echo-вывода

Приложение 2

Основные типы переменных языка С

Тип	Название типа	Диапазон возможных значений
char	Символьный	Символы ASCII, числа от -128 до 127; целые числа от 0 до 255
int	Целый	от -32768 до 32767
float	Вещественный	от $3,4 \cdot 10^{-38}$ до $3,4 \cdot 10^{+38}$
double	Вещественный двойной точности	от $1,7 \cdot 10^{-308}$ до $1,7 \cdot 10^{+308}$
void	Пустой, не имеющий значения	
bool	Логический	true или false

Приложение 3

Коды формата для стандартных типов данных языка С

Переменная	Команда формата
Целое десятичное число со знаком	%d
Вещественное число	%f
Вещественное число двойной точности	%lf
Текстовый символ	%c
Целое число без знака	%u

Список управляющих последовательностей языка С

Управляющий символ	Название	Действие
\n	lf (line feed)	Перевод строки
\a	bel (audible bell)	Звуковой сигнал
\b	bs (backspase)	Возврат на шаг (забой)
\t	ht (horizontal tab)	Табуляция
\v	vt (vertical tab)	Вертикальная табуляция

Приложение 4

Образец оформления титульного листа

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

Институт инновационных технологий
Факультет информационных технологий
Кафедра информатики и защиты информации

Лабораторная работа № …..
(номер лабораторной работы)
Вариант …..
(номер варианта)

Принял
_____ ФИО

Исполнитель,
студент гр. (название группы) _____ФИО

Владимир

ОГЛАВЛЕНИЕ

Учебное издание

АРТЮШИНА Лариса Андреевна
ВОРОНИНА Анна Аркадьевна

ТЕХНОЛОГИИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ

Учебное пособие

Подписано в печать 02.04.14.
Формат 60×84/16. Усл. печ. л. 5,58. Тираж 55 экз.
Заказ
Издательство
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых.
600000, Владимир, ул. Горького, 87.