МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ М.В. ЛОМОНОСОВА

Факультет
вычислительной математики
и кибернетики

Е.А. Бордаченкова, А.А. Панфёров

ЗАДАНИЯ ПРАКТИКУМА
1 курс

МОСКВА

2016

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В. ЛОМОНОСОВА
Факультет вычислительной математики и кибернетики

Е.А. Бордаченкова, А.А. Панфёров

Задания практикума. 1 курс

Учебное пособие
для студентов 1 курса

МАКС ПРЕСС

МОСКВА – 2016

УДК 004.42(075.8)
ББК 32.973-018.1я73
Б82

Печатается по решению Редакционно-издательского совета факультета вычислительной математики и кибернетики Московского государственного университета имени М.В. Ломоносова

Рецензенты:
В.Г. Баула – доцент кафедры АСВК ВМК МГУ;
Е.А. Кузьменкова – доцент кафедры СП ВМК МГУ

Бордаченкова Е.А., Панфёров А.А.

Б82 **Задания практикума. 1 курс:** Учебное пособие для студентов 1 курса. – М.: Издательский отдел факультета ВМиК МГУ имени М.В. Ломоносова (лицензия ИД N 05899 от 24.09.2001 г.); МАКС Пресс, 2016. – 48 с.
ISBN 978-5-89407-558-7
ISBN 978-5-317-05245-4

В книге представлены задания, предназначенные для выполнения в рамках практикума студентами первого курса факультета ВМК МГУ. Часть заданий посвящена совершенствованию навыков программирования на языке Паскаль, другая часть помогает освоить язык ассемблер. Задания сопровождаются методическими указаниями. Учебное пособие предназначено для студентов первого курса и преподавателей, ведущих занятия по практикуму на факультете ВМК.

Ключевые слова: обучение программированию, Паскаль, ассемблер.

УДК 004.42 (075.8)
ББК 32.973-018.1я73

Bordachenkova E.A., Panferov A.A.

Tasks of the practicum. 1 course: Textbook for the first-year students. – M.: Publishing Department of the faculty of mathematics and Cybernetics of MSU (license N 05899 from 24.09.2001); MAKS Press, 2016. – 48 p.

This textbook contains the tasks for the first-year students of CMC MSU. Some of them is devoted to improving of programming skills, other helps to learn assembly language. All tasks are accompanied by guidelines.

Keywords: programming training, Pascal, assembly language.

ISBN 978-5-89407-558-7
ISBN 978-5-317-05245-4

Предисловие

Предлагая настоящее пособие, авторы выражают надежду, что оно будет способствовать стандартизации практикума на первом курсе. В сборнике представлены задания как на отрабатывание сложных тем первого семестра, как и на освоение материала второго семестра. Задания расположены в таком порядке, чтобы к моменту их выполнения студенты обладали необходимыми теоретическими знаниями. Чрезвычайно важно, чтобы студенты ясно представляли теоретическую основу реализуемых алгоритмов.

Авторы стремились к тому, чтобы сделать задания компактными по формулировке и по содержанию и в то же время достаточно объемными. Каждое задание нацелено на освоение одной темы. В качестве языков реализации предлагаются Паскаль и ассемблер.

Задания 1 и 2 посвящены совершенствованию навыков работы с динамическими структурами данных.

Задание 3 демонстрирует связь между дисциплинами, изучаемыми в рамках базового обучения на младших курсах.

Основная цель задания 4 — углублённое изучение методов сортировки, важной темы курса «Алгоритмы и алгоритмические языки».

Задание 5 даёт первое представление о программировании на ассемблере. Его можно выдавать сразу после изучения на семинарах тем «Переходы» и «Ввод-вывод». Задание небольшое, поэтому студенты могут выполнять его параллельно с каким-нибудь более сложным заданием.

Цель задания 6 — освоение темы «Процедуры» на ассемблере.

Задание 7 продолжает отработку правил работы с процедурами на ассемблере и даёт опыт использования команд работы с битами.

Цель задания 8 — практика работы с многомодульными программами на ассемблере.

Формирование у студентов навыков создания относительно больших программ — одна из важных задач базового курса программирования. Выполняя задания практикума, студенты получают необходимый опыт в этой области. Желательно, чтобы преподаватели контролировали студентов и помогали им на различных этапах проектирования и отладки программ. Положительный опыт — поэтапная сдача заданий. С одной стороны, она организует по вре-

мени работу студентов, не позволяя откладывать всё на последний момент, с другой стороны, помогает сформировать методологически правильный подход к созданию программ: программирование сверху вниз, деление задачи на подзадачи, отдельная разработка и отладка логически законченных частей программы.

Авторы выражают признательность И. В. Горячей, Т. Ю. Грацианової и Д. Ю. Волканову, беседы с которыми оказали влияние на содержание книги. Наша особая благодарность Владимиру Николаевичу Пильщикову, чьё влияние мы постоянно ощущаем.

Введение

Во втором семестре в рамках предмета Программирование, кроме семинарских занятий, появляются задания практикума. Если на семинарах решаются небольшие задачи, то задания практикума более объёмные.

На выполнение задания отводится довольно большой отрезок времени — до месяца. Студенты работают над заданиями самостоятельно, консультируясь при необходимости с преподавателем. Ещё одна особенность заданий практикума состоит в том, что задание, как правило, включает в себя несколько более-менее законченных задач.

Важно придерживаться технологии программирования «сверху вниз» на каждом этапе работы. Сначала нужно сформулировать решение в самых общих чертах, определить, в каких терминах формулируется решение — какие нужны переменные на уровне основной программы. Затем выделить логически законченные части — подзадачи, определить связи между основной задачей и подзадачами. Фактически, на этом этапе формируется назначение и интерфейс (параметры) процедур. Далее каждую подзадачу нужно решать независимо от остальных частей, опять же, применяя метод программирования «сверху вниз».

Прежде, чем приступить непосредственно к программированию, нужно разобраться в математическом методе, который требуется реализовать: прочесть методические указания, возможно, обратиться к учебникам. Следует вручную выполнить несколько шагов метода, это поможет выявить взаимосвязи между значениями переменных и организовать процесс вычислений.

Важным аспектом выполнения задания практикума является отладка. Процесс отладки заключается в тестировании программы (процедуры) и исправлении обнаруженных ошибок. Тестирование — это выполнение программы с такими входными данными, для которых заранее известен правильный результат. Если выданный программой результат не совпадает с ожидаемым, значит, в программе есть ошибка. Тестовых данных должно быть достаточное количество, чтобы проверить все возможные основные варианты работы программы. Каждую процедуру нужно отлаживать отдельно, для этого надо написать программу, которая вызывает отлаживаемую про-

цедуру, передавая подходящие параметры, и печатает полученные результаты.

Работа с модулями в среде Турбо Паскаль

В некоторых заданиях требуется организовать программу в виде набора модулей. Модуль — логически замкнутая часть программы, которую можно компилировать отдельно от остальных частей. Программа, имеющая модульную структуру, состоит из основной программы модулей: программа = основная программа + модули. Обычно в модуль выносят близкие по назначению процедуры.

Использование модулей упрощает разработку больших программ. Во-первых, вынесение процедур позволяет сделать прозрачным текст основной программы, не перегружать его деталями. Во-вторых, модульную программу проще модифицировать — в зависимости от различных условий можно подключать к основной программе разные модули с процедурами. В-третьих, процедуры, требующиеся во многих программах, можно сгруппировать в «стандартный модуль», разрешив программам использовать его; примером является модуль `CRT` системы Турбо Паскаль. Последнее преимущество вытекает из раздельной компиляции модулей (раздельная компиляция — ключевая характеристика модуля). При внесении изменений в многомодульную программу достаточно перекомпилировать только измененный модуль и связанные с ним модули.

Работа с модулями в системе Турбо Паскаль устроена следующим образом.

Модуль содержит описания констант, типов, переменных, процедур. Информацию, описанную в одном модуле, может использовать основная программа и другие модули. Описание модуля выглядит так:

```
unit ⟨имя модуля⟩;
interface
  ⟨интерфейсная часть⟩
implementation
  ⟨реализация⟩
end.
```

В интерфейсной части описываются имена (константы, типы, процедуры и функции), видимые вне модуля. Для процедур и

функций приводятся только заголовки. В части реализации даются полные описания для всех процедур и функций, перечисленных в интерфейсной части, и описываются локальные для модуля имена, недоступные другим частям программы, но необходимые для работы модуля.

Модуль необходимо сохранить в файл с именем ⟨имя модуля⟩.pas. Важно: имя модуля, указанное в заголовке, должно совпадать с именем файла. Модуль можно откомпилировать (с помощью команды Compile); откомпилированный модуль записывается в файл ⟨имя модуля⟩.tpu (Turbo Pascal Unit). Однако, специально компилировать модули нет необходимости, т. к. их компиляция происходит при запуске основной программы. Заметим, что модуль нельзя выполнить (командой Run).

Если программа использует модуль, в начале текста программы, сразу после заголовка, нужно поместить ключевое слово **uses**, за которым через запятую перечислить все используеме модули:

```
program ...;
uses ⟨имя модуля1⟩, ⟨имя модуля2⟩, ⟨имя модуля3⟩;
  ⟨тело программы⟩
end.
```

Модуль может использовать другие модули, в этом случае **uses** со списком используемых модулей указывается после заголовка модуля.

Поскольку модуль нельзя выполнить самостоятельно, для отладки процедур модуля нужно написать вспомогательную программу, вызывающую эти процедуры.

Задание 1. Обработка многочленов
Язык Паскаль. Списки

Постановка задачи

Требуется написать программу, которая обрабатывает многочлены с целыми коэффициентами. Программа должна обладать удобным интерфейсом: печатать приглашение для ввода данных, пояснение выводимого результата. На вход программе подаются многочлен и, в зависимости от варианта задания, число, одночлен или другой многочлен. Признаком конца ввода каждого данного является точка.

Вид многочлена описывается следующей РБНФ:

⟨многочлен⟩ ::= [-]⟨одночлен⟩{⟨знак⟩⟨одночлен⟩}
⟨одночлен⟩ ::= ⟨коэффициент⟩`x^`⟨степень⟩
⟨знак⟩ ::= `+` | `-`
⟨коэффициент⟩ ::= ⟨число⟩
⟨степень⟩ ::= ⟨число⟩
⟨число⟩ ::= ⟨цифра⟩{⟨цифра⟩}
⟨цифра⟩ ::= `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

В многочлене нет слагаемых с коэффициентами, равными нулю; исключением является нулевой многочлен, который вводится как `0x^0`.

Пример вводимого многочлена: `-1x^3+25x^1-13x^0`.

Слагаемые (одночлены) во входной последовательности могут быть расположены в определённом порядке, либо никак не упорядочены, это определяется вариантом задания.

Во время работы программы многочлен хранится в виде списка одночленов с ненулевыми коэффициентами, упорядоченных по убыванию степеней (без заглавного звена). Нулевой многочлен представляется в виде пустого списка.

Если результатом является многочлен, он должен быть напечатан в соответствии с РБНФ, пробелов перед числами и после них быть не должно.

Варианты задания

1. **Порядок одночленов во входном тексте**
   В вариантах 1–3 все степени одночленов различны, т. е. в многочлене нет слагаемых с одинаковыми степенями переменной x.
   1) Одночлены упорядочены по возрастанию степеней.
   2) Одночлены упорядочены по убыванию степеней.
   3) Одночлены вводятся в произвольном порядке.
   4) Одночлены вводятся в произвольном порядке; могут встретиться одночлены с одинаковыми степенями переменной.
2. **Обработка многочлена**
   1) Умножить многочлен $P(x)$ на число c.
   2) Вычислить производную многочлена.
   3) Вычислить n-ую производную многочлена.
   4) Умножить многочлен $P(x)$ на одночлен $Q(x)$.
   5) Прибавить к многочлену $P(x)$ одночлен $Q(x)$.
   6) Вычислить значение многочлена в точке.
   7) Вычислить целые корни многочлена.
   8) Сложить два многочлена $P(x)$ и $Q(x)$.
   9) Вычислить произведение двух многочленов $P(x)$ и $Q(x)$.

Требования к программе

1. После завершения вычислений вся выделенная динамическая память должна быть высвобождена.
2. Описать в программе следующие процедуры:
   1) ввод многочлена (результат — список, внутреннее представление многочлена);
   2) ввод одночлена (результат — ссылка на звено, содержащее одночлен);
   3) ввод числа;

4) обработка многочлена в соответствии с вариантом задания;

5) печать списка;

6) удаление списка.

3. Входные данные должны вводиться посимвольно; запрещено вызывать процедуру `read` с целочисленным параметром.

4. После ввода многочлена напечатать построенный список.

5. Вычисление значения многочлена в точке реализовать на основе схемы Горнера.

6. Не использовать операторы перехода.

Методические указания

1. Для чтения входной последовательности удобно использовать глобальную символьную переменную. Глобальная переменная будет изменяться процедурой ввода одночлена и процедурой ввода числа. После завершения процедуры ввода числа эта переменная будет содержать символ, следующий за числом. В разных случаях это будет либо `x`, либо знак плюс или минус, либо точка.

2. Целыми корнями многочлена могут быть только делители свободного члена (положительные и отрицательные).

3. В процессе обработки многочлена в некоторых вариантах могут появляться слагаемые с коэффициентами, равными нулю. Но в списке, представляющем многочлен, не должно быть нулевых слагаемых, их следует удалять. Чтобы не затемнять основную логику преобразования многочлена, удобно сначала реализовать полную обработку многочлена без анализа равенства нулю коэффициентов, а затем удалить из списка звенья, содержащие нулевые слагаемые. При этом удаление нулевых звеньев из списка описать в виде процедуры.

Задание 2. Частотный словарь
Язык Паскаль. Деревья

Постановка задачи

Необходимо написать программу решения следующей задачи. В файле содержится текст английской книги (фрагмент). Требуется по фрагменту книги построить частотный словарь, затем проанализировать текст фрагмента согласно варианту задания и напечатать полученный результат. Частотный словарь состоит слов, входящих в текст, с указанием количества вхождений каждого слова. Для корректного построения словаря все слова должны быть нормализованы, т. е. стоять в начальной форме (существительные — в единственном числе, глаголы — в неопределённой форме). Перед обработкой программой текст следует подготовить вручную, поставив слова в начальную форму.

Программа должна

1. прочитать внешний текстовый файл-книгу и построить словарь;
2. вывести частотный словарь на экран или в текстовый файл, по желанию пользователя;
3. проанализировать словарь и вывести на экран информацию, требуемую вариантом.

Входом программы является внешний текстовый файл, содержащий английские слова и знаки препинания: пробел, запятая, точка. Все слова записаны строчными буквами.

Если в варианте задания предусмотрена печать последовательности слов, нужно напечатать их в словарном (лексикографическом) порядке.

Варианты задания

1. Посчитать количество различных слов в книге.
2. Напечатать все слова, начинающиеся на заданную букву, для каждого слова указать количество его вхождений в книгу.

3. Напечатать все слова, оканчивающиеся на заданную букву, для каждого слова указать количество его вхождений в книгу.
4. Напечатать все слова, наиболее часто встречающиеся в книге.
5. Напечатать самые редкие слова — входящие в книгу наименьшее количество раз.
6. Напечатать слова максимальной длины.
7. Напечатать слова минимальной длины.
8. Для каждой буквы алфавита напечатать, сколько слов начинается с этой буквы.

Требования к программе

1. В программе должны быть описаны следующие процедуры:
    1) процедура чтения текста и построения словаря, параметр — словарь;
    2) процедура печати дерева-словаря, параметр — словарь;
    3) процедура обработки словаря согласно варианту задания с печатью результата, параметр — словарь.

Чтение слова также оформить в виде процедуры.

2. Программа должна обработать одну страницу реальной книги на английском языке. Файл, содержащий эту страницу, нужно подготовить в текстовом редакторе, преобразовав исходный текст так, чтобы он соответствовал условию задачи: все буквы строчные, слова нормализованы, удалены все знаки препинания кроме пробелов, запятых и точек.

3. Перед завершением программы необходимо удалить дерево, освободив динамическую память.

4. По результатам выполнения задания нужно написать отчет. Содержание отчёта:
    1) Вариант задания.
    2) Информация о работе программы:
        а) исходный английский текст с указанием источника и номера страницы,

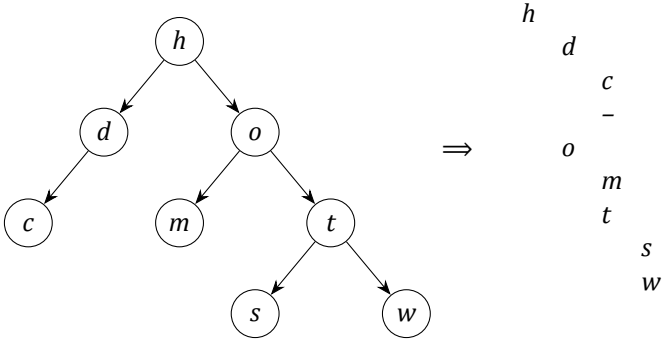
б) подготовленный для обработки текст с нормализованными словами,
в) дерево-словарь,
г) результаты работы программы.

3) Информация о программе:
а) название программного файла,
б) название файла с входным текстом,
в) интерфейс основных процедур (название, смысл параметров, действие).

Методические указания

1. Максимальную длину слова описать в виде константы.

2. Дерево удобно печатать вертикально, каждый узел располагать в отдельной строке с отступом, соответствующим уровню вложенности узла. Если один из потомков отсутствует, помечать это знаком «–». Например,



Для управления сдвигами в процедуре печати дерева нужно описать вложенную рекурсивную процедуру с дополнительным параметром — шириной отступа или глубиной вложенности текущего поддерева.

Особенности работы с внешними файлами в системе Турбо Паскаль

Работа с внешними файлами в системе Турбо Паскаль несколько отличается от описанной правилами языка Паскаль.

Прежде всего, нужно описать в программе переменную файлового типа. Затем необходимо связать с файловой переменной внешний файл с помощью предописанной процедуры `assign(fv, FileName)`, здесь `fv` — имя файловой переменной, `FileName` — имя внешнего файла. Имя внешнего файла указывается в апострофах, как строковая константа Паскаля. Это имя строится по правилам записи имён в операционной системе MS-DOS: оно должно содержать от одного до восьми символов (буквы латинского алфавита и цифры), за которыми следует точка и так называемое расширение, характеризующее тип файла. Текстовые файлы обычно имеют расширение `txt`.

Пример связывания файловой переменной с внешним файлом

```
{var book: text;}
assign(book, 'Hobbit.txt')
```

После этого работа с файловой переменной осуществляется согласно правилам Паскаля: можно использовать стандартные процедуры (`reset(fv)`, `read(fv,…)`, `rewrite(fv)`, `writeln(fv)`, `write(fv,…)`).

По окончании работы с файлом его следует закрыть с помощью процедуры `close(fv)`.

Задание 3. Вычисление корней уравнений и определенных интегралов Язык Паскаль

Постановка задачи[1]

С заданной точностью *eps* вычислить площадь плоской фигуры, ограниченной тремя кривыми, уравнения которых $y = f_1(x)$, $y = f_2(x)$ и $y = f_3(x)$ определяются вариантом задания.

При решении задачи необходимо:

1) с некоторой точностью eps_1 вычислить абсциссы точек пересечения кривых, используя предусмотренный вариантом задания метод приближенного решения уравнения $f(x) = g(x)$. Отрезки, где программа будет искать точки пересечения и где применим численный метод, определить вручную.
2) представить площадь заданной фигуры как алгебраическую сумму определенных интегралов и вычислить эти интегралы с некоторой точностью eps_2 по квадратурной формуле, предусмотренной вариантом задания.

Величины eps_1 и eps_2 подобрать вручную так, чтобы гарантировалось вычисление площади фигуры с точностью eps.

Варианты задания

Во всех вариантах $eps = 0.001$.

А. Уравнения кривых $y = f_i(x)$

1)	$f_1 = 2^x + 1$	$f_2 = x^5$	$f_3 = (1 - x)/3$
2)	$f_1 = 1.5/(x + 1) + 3$	$f_2 = 2.5x - 9.5$	$f_3 = 5/x \quad (x > 0)$
3)	$f_1 = e^{-x} + 3$	$f_2 = 2x - 2$	$f_3 = 1/x$
4)	$f_1 = e^x + 2$	$f_2 = -1/x$	$f_3 = -2(x + 1)/3$
5)	$f_1 = 0.35x^2 - 0.95x + 2.7$	$f_2 = 3^x + 1$	$f_3 = 1/(x + 2)$
6)	$f_1 = 0.6x + 3$	$f_2 = (x - 2)^3 - 1$	$f_3 = 3/x$

[1] Материал с некоторыми изменениями взят из книги Н. П. Трифонов, В. Н. Пильщиков «Задания практикума на ЭВМ (1 курс)»

7)	$f_1 = \ln x$	$f_2 = -2x + 14$	$f_3 = 1/(2 - x) + 6$
8)	$f_1 = e^x + 2$	$f_2 = -2x + 8$	$f_3 = -5/x$
9)	$f_1 = 3/((x-1)^2 + 1)$	$f_2 = \sqrt{x + 0.5}$	$f_3 = e^{-x}$
10)	$f_1 = 1 + 4/(x^2 + 1)$	$f_2 = x^3$	$f_3 = 2^{-x}$

Б. Методы приближенного решения уравнений:

1) метод деления отрезка пополам;
2) метод хорд (секущих);
3) метод касательных (Ньютона);
4) комбинированный метод (хорд и касательных).

В. Квадратурные формулы:

1) формула прямоугольников;
2) формула трапеций;
3) формула парабол (Симпсона).

Требования к программе

1. Программа должна состоять из двух модулей: основной модуль, вычисляющий площадь криволинейного треугольника, и модуль, содержащий процедуры, которые реализуют численные методы.

2. Метод приближенного решения уравнения и приближенное вычисление интеграла реализовать в виде процедур:

   а) `root(f,g,a,b)` — вещественная функция, вычисляющая с точностью eps_1 (константа) корень уравнения $f(x) = g(x)$ на отрезке $[a, b]$. Если используется метод касательных или комбинированный метод, то у функции `root` должны быть еще параметры $f1$ и $g1$ — производные функций f и g.

   б) `integral(f,a,b)` — вещественная функция, вычисляющая с точностью eps_2 (константа) величину определенного интеграла от функции $f(x)$ на отрезке $[a, b]$.

3. В основной программе предусмотреть печать абсцисс точек пересечения кривых, количество отрезков разбиения, которое получилось при вычислении интегралов, и печать площади криволинейного треугольника. Печатать только точные цифры результата — используя форматный вывод напечатать только цифры, обеспечиваемые заданной точностью вычислений.

4. Процедуры `root` и `integral` следует предварительно протестировать. Для каждой процедуры нужно составить набор тестовых функций, достаточный для проверки всех существенных вариантов работы `root` и `integral`; написать отдельные программы, содержащие применение `root` и `integral` к тестовым функциям.

5. По итогам работы составить отчёт, который следует предъявить при сдаче задания. Содержание отчета:

1) Постановка задачи — конкретный вариант.
2) Схематичное изображение графиков функций с обозначенными точками пересечения графиков и выделенным криволинейным треугольником; формула выражения площади треугольника через определённые интегралы.
3) Отрезки для поиска точек пересечения графиков, обоснование применимости метода на каждом отрезке.
4) Обоснование выбора eps_1 и eps_2.
5) Набор тестов для функции `root`.
6) Набор тестов для функции `integral`.
7) Результаты выполнения основной программы.

Методические указания

Численное решение уравнений

1. Вместо исходного уравнения $f(x) = g(x)$ будем решать уравнение $F(x) = 0$, где $F(x) = f(x) - g(x)$. Пусть x_0 — точное решение уравнения. Требуется найти точку x', которая близка к точному решению x_0, а именно, $|x' - x_0| < eps_1$. Суть численных методов решения уравнения заключается в построении последовательности точек, сходящейся к точному решению. На каждом шаге вычисляется очередная точка последовательности — очередное приближение. Процесс завершается, когда расстояние от приближения до точного решения стало меньше заданной точности eps_1.

Для корректного применения предложенных методов приближенного решения уравнения $F(x) = 0$ необходимо найти отрезок $[a, b]$, на котором уравнение имеет ровно один корень. Достаточное

условие для этого таково: на концах отрезка функция $F(x)$ имеет разные знаки и на всем отрезке производная функции не меняет знак. Кроме того, для методов хорд и касательных, а также комбинированного метода обязательно требуется, чтобы на данном отрезке первая и вторая производные функции не меняли свой знак (не обращались в нуль).

2. В ***методе деления отрезка пополам*** в качестве первого приближения к корню берётся средняя точка c отрезка $[a, b]$. Из двух отрезков $[a, c]$ и $[c, b]$ выбирается тот, на концах которого функция $F(x)$ имеет разные знаки. К выбранному отрезку применяется та же процедура. Процесс деления отрезков прекращается, когда длина очередного отрезка станет меньше требуемой точности eps_1. В качестве приближённого решения можно взять середину отрезка.

В остальных трёх методах следует различать два случая:

- случай 1 — первая и вторая производные функции $F(x)$ имеют одинаковый знак ($F'(x)F''(x) > 0$);
- случай 2 — эти производные имеют разные знаки.

В ***методе хорд*** концы $(a, F(a))$ и $(b, F(b))$ кривой $y = F(x)$ соединяются прямой линией и определяется точка пересечения этой линии с осью абсцисс:

$$c = \frac{aF(b) - bF(a)}{F(b) - F(a)}.$$

Точка c и есть первое приближение к корню. Дальнейшие действия зависят от случая, который имеет место.

Случай 1. Продолжаем вычисления на отрезке $[c, b]$, т. е. полагаем новое значение a равным c, $a_1 = c$. Опять проводим хорду на отрезке $[a_1, b]$, определяем точку пересечения хорды с осью абсцисс (a_2) и т. д. Таким образом строим возрастающую последовательность левых концов отрезков $(a = a_1, a_2, \ldots)$ Процесс завершается, когда точное решение отстоит от a_i не дальше, чем на eps_1. Это означает, что $F(x)$ меняет знак на отрезке $[a, a + eps_1]$, т. е. $F(a)F(a + eps_1) < 0$.

Случай 2. Вычисления продолжаются на отрезке $[a, c]$, полагаем теперь $b_1 = c$. Проводим хорду на отрезке $[a, b_1]$, получаем координату пересечения хорды с осью абсцисс — новое значение b. Таким образом получаем убывающую последовательность правых концов отрезков $(b = b_1, b_2, \ldots)$ Процесс завершается, когда на отрезке $[b - eps_1, b]$ функция $F(x)$ меняет знак ($F(b - eps_1)F(b) < 0$).

В ***методе касательных***, в зависимости от имеющего место случая, выбирается точка d, проводится касательная к кривой $y = F(x)$ в точке $(d, F(d))$. Точка пересечения касательной с осью абсцисс

$$d_1 = d - \frac{F(d)}{F'(d)}$$

является первым приближением решения уравнения. Далее процесс повторяется: проводится касательная в точке $(d_1, F(d_1))$, вычислив координату пересечения касательной с осью абсцисс, получаем очередное приближение к корню d_2 и т. д. Вычисления заканчиваются, когда приближённое решение оказывается достаточно близко к точному корню.

Случай 1. Касательные проводятся в правом конце отрезка, $d = b$. Последовательность точек $\{d_i\}$ является убывающей, приближение к корню происходит справа; условие окончания вычислений то же, что и в методе хорд: $F(d - eps_1)F(d) < 0$.

Случай 2. Касательные проводятся в левом конце отрезка, $d = a$. Последовательность точек $\{d_i\}$ возрастает, приближение к корню происходит слева; условие окончания вычислений: $F(d)F(d + eps_1) < 0$.

В ***комбинированном методе*** одновременно применяются метод хорд и метод касательных к разным концам отрезка. В случае 1 точка a изменяется по методу хорд, точка b — по методу касательных, в случае 2 — наоборот. Приближение к корню происходит с двух сторон. Критерий окончания — длина очередного отрезка меньше eps_1.

3. При использовании метода хорд, метода касательных или комбинированного метода функция `root` должна самостоятельно распознавать, какой из двух случаев, указанных в п. 2., имеет место при текущем обращении к ней. Это можно сделать проверкой следующих двух условий:

- функция возрастает или убывает;
- график функции расположен выше хорды, соединяющей концы графика, или ниже.

Поскольку производные $F'(x)$ и $F''(x)$ на отрезке $[a, b]$ не меняют знак, для проверки первого условия достаточно сравнить $F(a)$ с 0 (при $F(a) < 0$ функция возрастает). Для проверки же второго условия надо сравнить в какой-то внутренней точке отрезка значения

функции и хорды; например, если взять среднюю точку $(a+b)/2$ отрезка, то соотношение

$$F\left(\frac{a+b}{2}\right) > \frac{F(a)+F(b)}{2}$$

означает, что график функции расположен выше хорды (т. е. функция вогнутая). Если функция возрастает и её график расположен ниже хорды или если функция убывает и её график расположен выше хорды, то имеет место случай 1, иначе — случай 2.

4. Для того, чтобы повысить наглядность процедуры `root` и сделать короче выражения, реализующие численный метод, можно описать вложенную функцию, например, `fg(x)`, вычисляющую разность $f(x) - g(x)$.

Численное интегрирование

1. Суть численного интегрирования функции $f(x)$ на отрезке $[a, b]$ состоит в следующем. Отрезок $[a, b]$ делится на n одинаковых отрезков. На каждом из отрезков разбиения вычисление интеграла исходной функции $f(x)$ заменяется на вычисление интеграла более простой функции; приближённое значение интеграла $f(x)$ на отрезке $[a, b]$ получается как сумма интегралов на всех отрезках разбиения. В зависимости от метода, исходная функция $f(x)$ заменяется на константу (метод прямоугольников), прямую (метод трапеций) или параболу (метод Симпсона). Соответствующие квадратурные формулы для приближенного вычисления интеграла I от функции $f(x)$ на отрезке $[a, b]$ имеют следующий вид (n — число отрезков разбиения $[a, b]$):

Формула прямоугольников:

$$I \approx I_n = h(f_0 + f_1 + \cdots + f_{n-1}),$$

где $f_i = f(a + (i + 0.5)h)$, $h = \frac{b-a}{n}$

Формула трапеций:

$$I \approx I_n = h(0.5f_0 + f_1 + f_2 + \cdots + f_{n-1} + 0.5f_n),$$

где $f_i = f(a + ih)$, $h = \frac{b-a}{n}$

Формула Симпсона (n — чётное):

$$I \approx I_n = \frac{h}{3}(f_0 + 4f_1 + 2f_2 + 4f_3 + \cdots + 4f_{n-3} + 2f_{n-2} + 4f_{n-1} + f_n),$$

где $f_i = f(a + ih)$, $h = \frac{b-a}{n}$

2. Для достижения требуемой точности обычно используется следующий метод: берётся некоторое начальное число разбиений n_0 (1, 2 или 3) и последовательно вычисляются значения I_n при n, равном $2n_0$, $4n_0$, $8n_0$ и т. д. Известно *правило Рунге*

$$|I - I_n| \approx p|I_n - I_{2n}|$$

(для формул прямоугольников и трапеций $p = 1/3$, для формулы Симпсона $p = 1/15$). Согласно этому правилу, когда на очередном шаге величина $p|I_n - I_{2n}|$ окажется меньше eps_2, в качестве приближенного значения для I можно взять I_n или, что лучше, I_{2n}.

3. При реализации функции `integral` следует учитывать, что в формулах трапеций и Симпсона в сумму I_{2n} входят значения f_i, вычисленные ранее для суммы I_n, поэтому их не следует перевычислять заново.

Отладка функций `root` и `integral`

1. В общем случае процесс отладки программы заключается в тестировании программы с целью выявления ошибок и в исправлении найденных ошибок. Подготавливается набор тестовых данных, таких, что правильный результат известен заранее. Программа применяется к тестовым данным, получившийся результат сравнивается с правильным ответом. Тестовые данные должны быть настолько простые, чтобы несложно было вручную получить ответ. Данных в тестовом наборе должно быть достаточное количество для проверки всех существенных вариантов работы тестируемой программы.

В задании требуется отладить не программу, а две процедуры. Для отладки процедуры нужно написать вспомогательную программу, которая вызывает процедуру для тестовых данных и печатает результат. Тестовыми данными являются функция и отрезок (параметры процедуры).

При отладке реализации численного метода нужно проверить, как точность метода влияет на вычисленный результат. Для этого нужно прогнать отладочную программу с разными значениями точности для одних и тех же тестовых данных и сравнить, на сколько вычисленные результаты отличаются от точного решения.

2. При отладке `root(f,g,a,b)` тест состоит из пары функций f и g и отрезка для нахождения корня. Удобно взять $g \equiv 0$. Отрезок

должен быть таким, чтобы разность функций на нём удовлетворяла условиям применимости численного метода (см. п. 1. раздела Численное решение уравнений). Набор тестов должен проверять все существенные случаи работы процедуры `root`. Для метода деления отрезка пополам по крайней мере один тест должен проверять случай, когда середина отрезка является корнем. Для остальных методов набор должен содержать тесты, соответствующие четырём случаям — всем возможным комбинациям возрастания/убывания и выпуклости/вогнутости функции. При отладке методов хорд и касательных процедура должна печатать, с какой стороны идёт приближение к корню; для комбинированного метода нужно печатать, к какому концу отрезка применяется метод хорд, а к какому — метод касательных. Для сравнения результатов вычислений, выполненных с разной точностью, во время отладки нужно печатать все знаки результата, без форматирования.

3. При отладке `integral(f,a,b)` тест состоит из функции f и отрезка интегрирования. Процедура должна печатать количество точек разбиения. Вообще говоря, чем лучше точность, тем больше отрезков разбиения получится. Однако, если интегрируемая функция в силу метода заменяется на саму себя, количество отрезков разбиения не зависит от точности (т. к. приближённое значение интеграла равно точному). В тестовый набор, помимо других тестов, должен входить тест, проверяющий эту ситуацию.

Функциональные параметры в среде Турбо Паскаль

В функциях `root` и `integral` используются параметры-функции (f, g и др.). В языке Турбо Паскаль такие параметры передаются следующим образом.

В разделе типов необходимо описать так называемый функциональный тип:

```
type TF = function (x: Real): Real;
```

(вместо `TF` и `x` можно использовать любые другие имена). Все вещественные функции от одного вещественного аргумента, передаваемого по значению, описанные в программе, автоматически включаются в функциональный тип `TF`. Имя данного типа нужно указать в спецификации формального параметра-функции в заголовках `root` и `integral`:

```
function root(f,g: TF; a,b: Real): Real;
```

При обращении к функциям `root` и `integral` указываются имена фактических параметров-функций, например:

```
root(f1, f2, -0.1, 3.5)
```

Замечание: стандартные функции нельзя передавать в качестве фактических параметров.

Перед описанием функций, передаваемых как фактические параметры (`f1`, `f2` и др.), необходимо разместить директиву транслятору `{$F+}`, например:

```
{$F+}
function f1(x:Real):Real; begin f1:=ln(x)  end;
function f2(x:Real):Real; begin f2:=2*x+14 end;
```

Директива `{$F+}` делает вызовы описанных вслед за ней процедур и функций дальними (по умолчанию действует директива `{$F-}` и все описываемые процедуры и функции объявляются ближними). Описание функционального типа также следует производить после директивы `{$F+}`, в противном случае дальние функции нельзя будет передавать в качестве аргументов такого типа.

Задание 4. Методы сортировки
Язык Паскаль

Постановка задачи

Требуется ввести последовательность данных в массив и упорядочить (отсортировать) массив одним из методов сортировки. Метод сортировки и тип данных определяется вариантом задания. В процессе сортировки нужно посчитать количество выполненных операций сравнения и перемещения элементов массива. Необходимо предусмотреть возможность демонстрации процесса упорядочения, т. е. возможность отображения состояния последовательности на разных этапах сортировки.

Более точно: программа должна

1. ввести
   а) длину упорядочиваемой последовательности ($\leqslant 100$);
   б) последовательность данных для сортировки;
   в) режим работы программы (обычный или демонстрационный);
2. упорядочить исходную последовательность;
3. напечатать полученную последовательность, количество выполненных сравнений и перемещений элементов.

В демонстрационном режиме программа должна печатать последовательность после каждого этапа (шага) сортировки. Элементы последовательности следует печатать в том же формате, в котором они вводились.

Смысл понятий «перемещение элементов» и «шаг сортировки» зависит от метода сортировки.

Варианты задания

1. Методы сортировки

№	метод сортировки	перемещение	шаг сортировки
1	простые вставки	$d_1 := d_2$	установка на место очередного элемента
2	бинарные вставки	$d_1 := d_2$	установка на место очередного элемента
3	пузырек	$d_1 \leftrightarrow d_2$	один просмотр массива с перестановками
4	челночная	$d_1 \leftrightarrow d_2$	перестановка на окончательное место элемента, нарушающего порядок
5	простой выбор	$d_1 \leftrightarrow d_2$	установка на место очередного элемента
6	Шелла	в зависимости от метода сортировки подмассива	упорядочен подмассив $x_i, x_{i+k}, x_{i+2k}, \ldots$
7	быстрая	$d_1 \leftrightarrow d_2$	перестановка элементов подмассива относительно элемента
8	простое слияние	$d_1 := d_2$	описан в методе
9	естественное слияние	$d_1 := d_2$	описан в методе

2. Виды упорядочиваемых данных

1. Информация о студентах: фамилия, имя, номер группы, дата рождения в формате **`dd.mm.yyyy`** (день.месяц.год). Упорядочить список студентов по убыванию возраста (по возрастанию дат рождения).
2. Ассортимент шоколада: марка, производитель, цена в формате **`P руб. K коп.`**; цена может выражаться только в рублях или только в копейках, тогда другое поле отсутствует (например, `68 руб. 50 коп.` или `80 руб.`) Упорядочить шоколад по возрастанию цены.
3. Ассортимент шоколада: марка, производитель, цена в формате **`nnn.nn S`**, где **`n`** — цифра, **`S`** — это знак `$`, `E` или `P`, обозначающий

валюту (доллар, евро или рубль); пример цены: `105.50 Р`. Упорядочить шоколад по возрастанию цены.

4. Участники выставки кошек: имя животного, порода, пол, вес в формате **К кг Г гр** (например, `5 кг 600 гр`). Упорядочить список участников по возрастанию веса.

5. Результаты выставки кошек: имя животного, порода, пять оценок, выставленных животному членами жюри. Каждая оценка — число от 0 до 5. Упорядочить результаты по возрастанию суммы оценок.

6. Спортсмены сборной команды по баскетболу: фамилия, имя, рост в формате **М м С см** (например, `2 м 3 см`). Упорядочить список спортсменов по увеличению роста игроков.

7. Список баскетболистов NBA: фамилия, имя, рост в футах и дюймах в формате **Ф f Д "** (например, `6 f 10 "`). Упорядочить список по увеличению роста игроков.

8. Информация о погоде в разных городах: город, среднегодовая температура в формате **±nn.n S**, где **n** — цифра, **S** — символ `C` или `F`, система измерения температуры (по Цельсию или по Фаренгейту), знак перед числом может отсутствовать; пример: `+14.2 C` или `71.7 F`. Расположить города в списке в порядке возрастания среднегодовой температуры.

9. Информация об автомобилях: марка автомобиля, год выпуска, объём топливного бака в литрах или в галлонах в формате **Л л** (литры) или **Г г** (галлоны) (например, `60 л` или `16 г`). Упорядочить список автомобилей по возрастанию объёмов топливных баков.

10. Информация о высоте городов над уровнем моря: город, высота над уровнем моря в метрах или в футах в формате **М м** (метры) или **Ф ф** (футы) (например, `1547 м` или `5074 ф`). Упорядочить список городов по возрастанию высот над уровнем моря.

11. Информация о географическом положении городов: город, широта и долгота в формате **nn с.ш. nnn в.д.** (например, `56 с.ш. 38 в.д.`), Широта может быть северная (`с.ш.`) или южная (`ю.ш.`), долгота — восточная (`в.д.`) или западная (`з.ш.`). Расположить города в списке в порядке с севера на юг.

Требования к программе

1. Метод сортировки реализовать в виде процедуры; последовательность и её реальная длина — параметры процедуры.

2. Ввод и печать последовательности оформить в виде соответствующих процедур.

3. Работу с элементами последовательности: чтение элемента, печать элемента, сравнение двух элементов, перемещение элемента (обмен) — оформить в виде процедур.

4. В программе должны быть описаны переменные — количество сравнений и количество перемещений. Процедуры сравнения и перемещения при каждом обращении к ним должны увеличивать значения этих глобальных счётчиков сравнений и перемещений.

5. Реализацию метода сортировки и процедуры обработки данных отлаживать отдельно.

Метод сортировки отлаживать на массиве целых чисел. Для этого разделить программу на модули.

1) *Основной модуль*. Содержит обрабатываемый массив, процедуры сортировки, ввода и печати последовательности. Содержит программу, которая реализует сценарий, данный в постановке задачи.

2) *Модуль данных*. Информация об элементах последовательности, соответствующих варианту задания. Модуль включает в себя описание типа элементов, а также все процедуры обработки элементов — чтение, печать, сравнение двух элементов, перемещение элементов (обмен).

3) *Отладочный модуль*. Содержит информацию для отладки метода сортировки: описание типа элементов и процедур работы для элементов — целых чисел. Имена типов и процедур этого модуля должны совпадать с соответствующими именами типов и процедур модуля данных.

В процессе отладки метода сортировки к основному модулю нужно подключить отладочный модуль. В дальнейшем, когда процедура сортировки будет отлажена, к основному модулю следует подключить модуль данных.

Для отладки процедур работы с данными потребуется написать программу, которая вызывает процедуры модуля данных.

6. Операторы перехода любого вида, включая `halt`, `exit`, `break`, не использовать.

7. По итогам работы составить отчёт, который следует предъявить при сдаче задания. Содержание отчета:

1) словесное описание метода сортировки;
2) наглядный пример пошагового применения метода сортировки к некоторому числовому массиву;
3) примеры худшего и лучшего массивов (т. е. массивов, на которых достигается максимальное и минимальное количество операций) с подсчётом количества операций для этих массивов.

Краткое описание методов сортировки[1]

Упорядочивается по возрастанию массив $X_1, \dots, X_n$.

Сортировки вставками

Общая идея методов сортировки вставками: в ранее упорядоченную подпоследовательность $X_1, X_2, \dots, X_{k-1}$ вставляется элемент X_k так, чтобы упорядоченными оказались k первых элементов исходной последовательности. Для этого

1. в подмассиве $X_1, X_2, \dots, X_{k-1}$ отыскивается место для элемента X_k, т. е. находится номер i такой, что $X_j < X_k$ для $j \in [1, i]$ и $X_k < X_j$ для $j \in [i+1, k-1]$;
2. элементы $X_{i+1}, \dots, X_{k-1}$ сдвигаются на одну позицию вправо, элемент X_k записывается на место элемента X_{i+1}.

Процесс заканчивается, когда поставлен на своё место элемент X_n.

В зависимости от способа поиска места для элемента X_k различаются следующие методы.

а) ***Метод простых вставок.*** Величина X_k по очереди сравнивается с элементами $X_{k-1}, X_{k-2}, X_{k-3}$ и т. д. до тех пор, пока не будет найдено место для элемента X_k.

[1] С небольшими изменениями здесь дан материал книги Н. П. Трифонов, В. Н. Пильщиков «Задания практикума на ЭВМ (1 курс)»

б) ***Метод бинарных вставок.*** Величина X_k сравнивается со средним элементом подпоследовательности $X_1, X_2, \ldots, X_{k-1}$. Если X_k меньше этого элемента, то место для него ищется тем же способом в левой половине подпоследовательности, а если больше — в правой половине.

Метод пузырька

По очереди просматриваются пары соседних элементов X_1 и X_2, X_2 и X_3, X_3 и X_4 и т. д.; в неупорядоченных парах ($X_i > X_{i+1}$) переставляются элементы. По окончании просмотра и перестановок максимальный элемент станет последним элементом последовательности. Далее аналогичная процедура применяется ко всем элементам, кроме последнего. Если при очередном просмотре последовательности не было произведено ни одной перестановки элементов, то последовательность уже упорядочена и следует прекратить сортировку.

Челночная сортировка

В этой сортировке элементы массива просматриваются попеременно то слева направо, то справа налево, как движется челнок в ткацком производстве. Сначала массив просматривается слева направо, сравниваются последовательные пары X_1 и X_2, X_2 и X_3, X_3 и X_4 и т. д. до обнаружения неупорядоченной пары: $X_{k-1} > X_k$. Далее начинается движение справа налево: «неупорядоченный» элемент X_k сравнивается и меняется местами с предыдущими элементами, пока не окажется на своём месте. Таким образом получается упорядоченный подмассив $X_1, X_2, \ldots, X_k$. Затем продолжается движение слева направо: начиная с $(k+1)$-ой позиции возобновляется просмотр массива X и поиск неупорядоченной пары. Если неупорядоченная пара не найдётся, процесс заканчивается.

Сортировка посредством простого выбора

В массиве $X_1, \ldots, X_n$ отыскивается минимальный элемент. Найденный элемент меняется местами с первым элементом массива. Затем так же обрабатывается подмассив $X_2, \ldots, X_n$. Когда обработан подмассив X_{n-1}, X_n, сортировка оканчивается.

Метод Шелла

Пусть k — целое от 1 до $n/2$. Независимо друг от друга упорядочиваются (одним из описанных выше методов) k подпоследовательностей из элементов, отстоящих друг от друга на k позиций: $X_i, X_{i+k}, X_{i+2k}, \ldots$ $(i = 1, 2, \ldots, k)$.

Затем k уменьшается, и процесс повторяется заново. Последний шаг должен выполняться при $k = 1$.

Значение k можно менять по разным законам. В отчёте следует объяснить выбранный способ изменения k. Метод сортировки подмассивов следует описать в виде вложенной процедуры, параметры — индекс начала подмассива и шаг k.

Быстрая сортировка (метод Хоара)

Суть метода заключается в следующем. Выбирается q — произвольный элемент массива. Элементы массива $X_1, \ldots, X_n$ переставляются таким образом, чтобы в начале массива оказались элементы, меньшие или равные q, а в конце — большие или равные q. Таким образом массив делится на две части.

Требуемую перестановку элементов можно выполнить следующим образом. Последовательность просматривается слева направо, пока не встретится элемент, больший или равный q; затем массив просматривается справа налево до элемента, меньшего или равного q. Найденные элементы меняются местами, после чего просмотры с обоих концов последовательности возобновляются со следующих элементов.

После разбиения массива на две части описанный выше процесс рекурсивно применяется к полученным подмассивам.

Когда длина обрабатываемого подмассива станет меньше трёх, его надо упорядочить более простым методом.

Сортировка слиянием

Основная идея такой сортировки — разделить последовательность на уже упорядоченные подпоследовательности (назовем их «отрезками») и затем объединять эти отрезки во всё более длинные упорядоченные отрезки, пока не получится единая упорядоченная последовательность. Отметим, что при этом необходима дополнительная память (массив $Y[1..n]$).

Различаются следующие варианты сортировки слиянием.

а) ***Простое слияние.*** Считается, что вначале отрезки состоят только из одного элемента, они сливаются в отрезки из двух элементов (из X_1 и X_2, из X_3 и X_4, …), которые переносятся в массив Y. На втором этапе соседние двухэлементные отрезки (Y_1, Y_2 и Y_3, Y_4; Y_5, Y_6 и Y_7, Y_8; …) объединяются в отрезки из 4 элементов, которые записываются в массив X. На третьем этапе строятся отрезки из 8 элементов, которые заносятся в массив Y, и т. д.

б) ***Естественное слияние.*** Массив X просматривается с начала, определяется наиболее длинный упорядоченный по неубыванию отрезок; затем массив X просматривается с конца и выбирается наиболее длинный упорядоченный (также по неубыванию), отрезок в конце массива; полученные отрезки сливаются в один отрезок, который записывается в начало массива Y. Затем сливаются следующие максимально длинные отрезки с обоих концов, и полученный отрезок записывается (справа налево) в конец массива Y. Третьи по порядку отрезки после слияния записываются снова в начало Y (вслед за первым объединенным отрезком), четвертые — в конец Y (перед вторым объединенным отрезком) и т. д. Первый этап сортировки оканчивается, когда все элементы из X будут перенесены в массив Y. На втором этапе применяется та же процедура, только массивы X и Y меняются ролями. И так далее, пока весь массив не образует один упорядоченный отрезок.

Замечание: в обоих вариантах следует учитывать, что в конце концов упорядоченные элементы должны оказаться в массиве X.

Задание 5. Арифметические вычисления Язык ассемблер

Постановка задачи

Требуется написать на ассемблере программу для решения следующей задачи. Дано число N, затем последовательность из N чисел. Ввести последовательность и вычислить значение выражения, определяемого вариантом задания. Считать, что входные данные корректны, т. е. вводится ровно N чисел. Программа должна напечатать приглашение для ввода длины последовательности и приглашение для ввода самой последовательности. Если вычислить значение выражения невозможно (знаменатель равен нулю), нужно напечатать сообщение об этом.

Варианты задания

1. Вычислить частное от деления первого числа последовательности на минимальный элемент, числа без знака.
2. Вычислить частное от деления первого числа последовательности на минимальный элемент, числа со знаком.
3. Вычислить остаток от деления первого числа последовательности на минимальный элемент, числа без знака.
4. Вычислить остаток от деления первого числа последовательности на минимальный элемент, числа со знаком.
5. Вычислить частное от деления последнего числа последовательности на минимальный элемент, числа без знака.
6. Вычислить частное от деления последнего числа последовательности на минимальный элемент, числа со знаком.
7. Вычислить остаток от деления последнего числа последовательности на минимальный элемент, числа без знака.
8. Вычислить остаток от деления последнего числа последовательности на минимальный элемент, числа со знаком.
9. Вычислить частное от деления максимального элемента на первое число последовательности, числа без знака.

10. Вычислить частное от деления максимального элемента на первое число последовательности, числа со знаком.
11. Вычислить остаток от деления максимального элемента на первое число последовательности, числа без знака.
12. Вычислить остаток от деления максимального элемента на первое число последовательности, числа со знаком.
13. Вычислить частное от деления максимального элемента на последнее число последовательности, числа без знака.
14. Вычислить частное от деления максимального элемента на последнее число последовательности, числа со знаком.
15. Вычислить остаток от деления максимального элемента на последнее число последовательности, числа без знака.
16. Вычислить остаток от деления максимального элемента на последнее число последовательности, числа со знаком.

Методические указания

1. Для составления программы нужно использовать шаблон — схему программы из раздела «Материалов по ассемблеру» сайта `http://al.cs.msu.ru`. В шаблоне указано, в каком месте программы нужно разместить данные и куда вписывать команды.

2. Во время обработки не требуется хранить всю последовательность чисел; запрещено использовать массив для хранения всех чисел.

3. Для печати сообщений использовать команду `OUTSTR`.

Задание 6. Обработка текстов
Язык ассемблер

Постановка задачи

Написать программу, которая вводит два текста, более длинный текст преобразует по первому правилу, более короткий — по второму правилу, затем печатает получившиеся тексты. Правила преобразования текстов определяются вариантом. Под текстом понимается непустая последовательность не длиннее 100 символов. Признак конца ввода — точка.

Программа должна печатать приглашение для ввода текста, печатать сообщение, по какому правилу (первому или второму) преобразуется каждый текст.

Если хотя бы одна из введённых последовательностей не является текстом (пустая либо длиннее 100 символов), правила преобразования не выполнять, а напечатать сообщение об ошибке ввода.

Варианты задания

1. Первое правило преобразования

1) Заменить каждую ненулевую цифру соответствующей ей строчной буквой латинского алфавита (1 → a, 2 → b и т. д.).
2) Заменить каждую строчную латинскую букву цифрой N mod 10, где N — порядковый номер буквы в алфавите.
3) Заменить каждую прописную русскую букву цифрой N mod 10, где N — порядковый номер буквы в алфавите.
4) Заменить каждую прописную латинскую букву следующей за ней по алфавиту, букву Z менять на A.
5) Заменить каждую прописную русскую букву следующей за ней по алфавиту, букву Я менять на А.
6) Заменить каждую строчную латинскую букву соответствующей прописной буквой, а прописную — строчной.
7) Заменить каждую латинскую букву буквой, симметричной ей в алфавите (A ↔ Z, b ↔ y, …).
8) Заменить каждую прописную русскую букву буквой, симметричной ей в алфавите (А ↔ Я, Б ↔ Ю, …).

2. Второе правило преобразования

1) Перевернуть текст, не используя дополнительную память.
2) Удвоить каждый символ текста.
3) Удвоить каждую строчную латинскую букву текста.
4) Удвоить каждую цифру, входящую в текст.
5) Удвоить каждую прописную русскую букву текста.
6) Циклически сдвинуть текст на три позиции влево. (с. к.)
7) Циклически сдвинуть текст на четыре позиции вправо. (с. к.)
8) Циклически сдвинуть текст на *K* (константа) позиций влево без использования дополнительной памяти, реализовав следующий алгоритм: перевернуть подмассив из первых *K* символов; перевернуть оставшийся подмассив; перевернуть весь текст.
9) Циклически сдвинуть текст на *K* (константа) позиций вправо без использования дополнительной памяти, реализовав следующий алгоритм: перевернуть подмассив из последних *K* символов; перевернуть оставшийся подмассив; перевернуть весь текст.
10) Заменить на `'*'` все элементы, идущие за первым вхождением `'a'` в текст. (с. к.)
11) Заменить на `'*'` все элементы, идущие за последним вхождением `'a'` в текст. (с. к.)
12) Заменить на `'*'` все элементы, идущие перед первым вхождением `'a'` в текст. (с. к.)
13) Заменить на `'*'` все элементы, идущие перед последним вхождением `'a'` в текст. (с. к.)
14) Удалить из текста все повторные вхождения его первого символа.
15) Удвоить каждую строчную латинскую букву текста.
16) Переставить все цифры в начало текста, сохранив их взаимный порядок.

Требования к программе

1. Значение 100 описать как константу. Для хранения текстов описать два массива.

2. Ввод текста описать в виде функции; функция возвращает значение **true** (0FFh), если введённая последовательность является текстом, и **false** (0) в противном случае. Правила преобразования текста описать в виде процедур. Интерфейс процедур (соответствующий заголовок на Паскале) нужно написать в виде комментария перед директивный PROC. Процедуры должны удовлетворять стандартным соглашениям о связях, параметры передавать в стеке.

3. Печать приглашения для ввода и печать пояснения, какое правило применяется к тексту, расположить ведущей части программы.

4. Нельзя преобразовывать текст во время печати, требуется сначала преобразовать текст в массиве, а затем печатать.

5. Нельзя использовать дополнительную память (дополнительный массив или стек) для построения преобразованного текста. Если преобразование предполагает увеличение длины текста, нужно сразу описать массив с запасом.

6. При реализации вариантов, помеченных «с. к.», требуется использовать строковые команды с префиксами повторения.

Методические указания

1. Фактически, в задании реализуется работа со строками переменной длины. Для представления строки удобно использовать две переменные — массив, где хранится содержание строки, и переменная, в которой хранится текущая длина строки.

2. Печать приглашения для ввода и печать пояснения, какое правило применяется к тексту, расположить ведущей части программы.

3. В процедуре преобразования текста может быть полезным иметь два указателя на позиции в тексте: указатель на обрабатываемый символ (источник) и указатель на позицию, куда происходит запись (получатель).

4. Считать, что прописные русские буквы упорядочены по алфавиту и расположены подряд, маленькие русские буквы упорядочены, но идут не подряд (кроме букв Ё, ё; считать, что этих букв не может быть в тексте).

5. Печатать преобразованные тексты командой OUTSTR; считать, что символ '$' не встречается в тексте.

Задание 7. Работа с упакованными данными
Языки Паскаль, ассемблер

Постановка задачи

Требуется модифицировать программу, реализованную в задании № 4 «Методы сортировки». Назовём ключом набор полей, по которым проводилась сортировка. Требуется упаковать ключ в слово или байт. Изменения касаются только модуля данных. Необходимо переопределить тип элементов сортируемой последовательности, сгруппировав данные, являющиеся ключом, в одно поле типа `word` или `byte` и написать на ассемблере процедуру упаковки данных, составляющих ключ, процедуру сравнения ключей и процедуру распаковки ключа. Процедура упаковки ключа будет использоваться в процедуре ввода элемента последовательности, процедура распаковки — в процедуре печати элемента.

Нужно проанализировать данные, являющиеся ключом, и разработать машинное представление ключа. Машинное представление должно быть, с одной стороны, компактным, без избыточности и, с другой стороны, удобным для эффективной реализации основных операций с ключами.

Варианты ключей

1. Дата **`dd.mm.yyyy`** (день.месяц.год) упаковывается в слово. В поле года хранить две последние цифры.
2. Цена плитки шоколада в виде **`P руб. K коп.`** упаковывается в слово. Считать, что цена не превосходит 500 руб.
3. Цена плитки шоколада в виде **`nnn.nn S`**, где **`n`** — цифра, **`S`** — это знак $, E или P, обозначающий валюту (доллар, евро или рубль), упаковывается в слово. Считать, что величина **`nnn.nn`** не превосходит 500.
4. Вес кошки **`K кг Г гр`** упаковывается в слово. Считать, что вес не превосходит 64 кг.
5. Ключ — массив из пяти оценок, каждая оценка лежит в интервале от 0 до 5. Упаковывается в слово.

6. Рост спортсмена-баскетболиста в виде **М м С см** упаковывается в байт. Поле метры хранить в виде смещения относительно 1 м.

7. Рост спортсмена в футах и дюймах в формате **Ф f Д "** упаковывается в байт.

8. Температура в формате ±**nn.n S**, где **n** — цифра, **S** — символ C или F, система измерения температуры (по Цельсию или по Фаренгейту) упаковывается в слово.

9. Объём топливного бака в литрах или в галлонах в формате **Л л** (литры) или **Г г** (галлоны) упаковывается в байт. Считать, что объём бака не больше 100 литров.

10. Высота городов над уровнем моря в метрах или в футах в формате **М м** (метры) или **Ф ф** (футы) упаковывается в слово.

11. Географические координаты широта и долгота (в формате **nn с.ш. nnn в.д.**) упаковываются отдельно. Широта упаковывается в байт, долгота — в слово.

Требования к программе

1. Для описания ключа использовать стандартный тип Турбо Паскаля `word` или `byte`.

2. Процедуры работы с ключами описать в части **`implementation`** модуля данных, используя директиву **`assembler`** (см. методические указания).

3. Для отладки ассемблерных процедур и демонстрации их корректности написать программу, которая вводит два элемента сортируемого массива, сравнивает их, печатая полученный результат, и печатает введённые элементы. Программа использует соответствующие процедуры, описанные в модуле данных.

4. Для демонстрации корректности работы процедуры упаковки в процедуре чтения элемента последовательности после ввода данных предусмотреть печать сгенерированного ключа (в виде одного числа).

5. После отладки ассемблерных процедур модуля данных подключить его к основной программе сортировки.

6. Отчёт должен включать анализ возможных величин ключей, описание машинного представления ключа и обоснование выбора машинного представления.

Методические указания

Ассемблерные процедуры в системе Турбо Паскаль

Турбо Паскаль позволяет писать тело процедуры на ассемблере. Для этого используется директива `assembler` и так называемый `asm`-оператор. Описание ассемблерной процедуры выглядит так:

```
⟨заголовок процедуры или функции⟩; assembler;
[⟨описание локальных переменных⟩]
asm
  ⟨последовательность команд и директив ассемблера⟩
end;
```

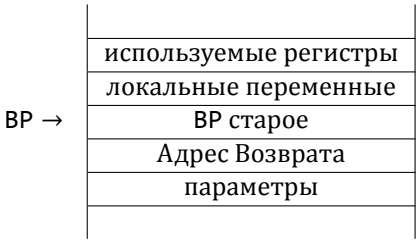
Комментарии в теле процедуры оформляются с учетом синтаксиса Турбо Паскаль и ассемблера: текст комментария записывается в фигурных скобках; комментарий может располагаться либо в отдельной строке, либо в одной строке с командой (директивой) после самой команды.

Например,

```
procedure ex(x:integer; var z:boolean); assembler;
var y:integer;
asm          {тело процедуры на ассемблере}
  MOV AX, X  {AX содержит значение X}
  MOV Y, 10  {использование локальной переменной}
  ADD Y, AX
  …
end;
```

При написании процедуры на ассемблере необходимо учитывать паскалевские соглашения о связях между программой и процедурой.

1. Параметры и локальные переменные располагаются в стеке. Ниже приведена структура стека.



Локальные переменные адресуются как [BP − константа], параметры — как [BP + константа].

2. При вызове процедуры параметры помещаются в стек в прямом порядке, сначала первый параметр, затем второй и т. д.

3. Параметры передаются по значению и по ссылке. В первом случае в стек помещается значение фактического параметра (см. таблицу).

тип параметра	передаётся в стеке
integer	word
char	byte
boolean	byte

Поскольку команды работы со стеком требуют операнда-слова, при передаче символа или логического значения в стек помещается слово, в младшем байте которого хранится параметр, старший байт слова не используется.

При передаче параметра по ссылке в стек помещается полный адрес фактического параметра, сегментная часть и смещение.

4. В конце работы процедура удаляет параметры из стека.

5. Если заголовок процедуры указан в интерфейсной части модуля (т. е. она доступна из других модулей), процедура транслируется как дальняя, FAR. Процедура, описанная только в реализационной части модуля (она будет вызываться только в этом модуле), транслируется как близкая, NEAR.

6. Ассемблерная процедура может изменять любые регистры, кроме BP, SP, SS, DS. Значение этих регистров следует сохранить в стеке в начале процедуры и восстановить из стека перед выходом из процедуры.

7. Функция возвращает значение в регистрах AL, AX или DX:AX в зависимости от типа результата: char, boolean, byte возвращается в AL, integer — в AX, указатель — в DX:AX.

Система Турбо Паскаль, транслируя текст ассемблерной процедуры, сама включает в начало процедуры, перед первой командой, стандартные входные действия (пролог), а в конец процедуры, за последней командой тела, — стандартный выходные действия (эпилог):

Пролог	Эпилог
PUSH BP	MOV BP, SP
MOV BP, SP	POP BP
SUB BP, Locals	RET Papams

Здесь Locals — размер локальных переменных, то есть количество байтов, занимаемых всеми локальными переменными; Params — размер параметров. Таким образом, программировать входные и выходные действия не нужно.

В теле ассемблерной процедуры разрешено использовать имена параметров и имена локальных переменных. Эти имена транслируются так, как будто они описаны директивами EQU. Например, для приведенного в начале параграфа фрагмента (в случае близкой процедуры) имеем

```
Z EQU [BP+4]
X EQU [BP+8]
Y EQU [BP-2]
```

−2	Y
BP →	BP ст
+2	AB
+4	ofs Z
+6	seg Z
+8	X

Кроме адресации относительно BP, с именем параметра (локальной переменной) связан тип, заданный в описании, его следует учитывать: например, команда

```
MOV AL, X
```

неверная, так как тип X — слово, а AL — байт.

Имена параметров, передаваемых по ссылке, имеют тип dword. Для доступа к фактическому параметру нужно написать, например, такие команды

```
MOV DS, word ptr Z+2 {сегментная часть – в старшем слове}
MOV BX, word ptr Z
MOV byte ptr [BX], 0 {z:=false}
```

В ассемблерной процедуре метки должны начинаться с символа @.

Сравнение ключей

В некоторых вариантах требуется сравнивать величины (ключи), выраженные в разных единицах измерения. Рекомендуется сравнивать ключи как рациональные числа по правилу

$m/n < a/b$ тогда и только тогда, когда $mb < na$.

Можно использовать соотношения:
Вариант 3. Курсы валют $1€ = 78$ руб., $1\$ = 73$ руб., $100\$ = 89€$.
Вариант 8. Температуры по Цельсию и по Фаренгейту связаны соотношением $9t_C = 5(t_F - 32)$.
Вариант 9. Перевод галлонов в литры: 1000 галлонов $\approx$ 3785 литров.
Вариант 10. Перевод футов в метры: 1 метр $\approx$ 3 фута.

Задание 8. Модули и макросы
Язык ассемблер

Постановка задачи

Требуется модифицировать программу, реализованную в рамках задания № 6, как описано ниже.

1. Разделить программу на два модуля, вынести процедуры в отдельный модуль.
2. Оформить в виде макросов действия по вызову процедур.
3. Описать в виде макроса проверку принадлежности символа заданному диапазону (например, проверка, является ли символ цифрой).

Методические указания

1. Процедуры могут быть дальними или близкими. Для описания близких процедур необходимо объединить сегменты кода обоих модулей. Для этого нужно дать сегментам одинаковые имена и в директивах `SEGMENT` указать параметр `PUBLIC`.

2. Директиву `INCLUDE IO.ASM` нужно указать в обоих модулях. Файл `IO.ASM` содержит макроопределения команд ввода-вывода; в основном модуле используются команды печати, а во вспомогательном модуле используются команды ввода символа.

3. Для упрощения процесса сборки программы, рекомендуется сохранить оба файла с исходным текстом модулей на ассемблере в каталог с `MASM`. Этот каталог должен содержать ассемблер (`MASM.EXE`), компоновщик (`LINK.EXE`), макросы ввода-вывода (`IO.ASM`), объектный код процедур ввода-вывода (`IOPROC.OBJ`).

Процесс сборки исполняемого модуля состоит из двух этапов:

1) ассемблирование (трансляция в объектный код) модулей, решающих задачу;
2) компоновка полученных объектных модулей и модуля с процедурами ввода-вывода в загрузочный (исполняемый) модуль.

Ассемблирование осуществляется запуском ассемблера (MASM.EXE) с передачей в качестве параметра имени файла, содержащего исходный код на ассемблере:

```
MASM.EXE ⟨файл⟩.ASM
```

При этом MASM задаст несколько вопросов по поводу именования выходных файлов. Почти на все вопросы можно оставлять умолчательные ответы, нажимая **Enter**. Если ассемблирование пройдёт успешно, то будет создан файл с объектным кодом модуля ⟨файл⟩.OBJ.

Описанную процедуру ассемблирования нужно произвести для каждого программного модуля.

Процесс компоновки заключается в однократном запуске редактора связей LINK.EXE, на вход которому нужно передать все объектные модули, из которых состоит программа (включая объектный модуль с процедурами ввода-вывода), разделяя их знаком плюс:

```
LINK.EXE ⟨файл1⟩.OBJ+⟨файл2⟩.OBJ+IOPROC.OBJ
```

В процессе компоновки также будут запрошены имена генерируемых файлов. Как и раньше, на все вопросы можно оставлять умолчательные ответы, нажимая **Enter**. Успешное завершение компоновки будет ознаменовано созданием исполняемого файла (загрузочного модуля) ⟨файл⟩.EXE. Исполняемый файл можно запустить на счёт.

Рекомендуемая литература

1. В. Г. Абрамов, Н. П. Трифонов, Г. Н. Трифонова. Введение в язык Паскаль. – М.: КНОРУС, 2011. – 384 с.
2. В. Н. Пильщиков. Язык Паскаль: упражнения и задачи. – М.: Научный мир, 2003. – 224 с.
3. Н. Вирт. Алгоритмы и структуры данных.: Пер. с англ. – 2-е изд., испр. – СПб.: Невский Диалект, 2001. – 352 с.: ил.
4. В. Н. Пильщиков. Assembler. Программирование на языке ассемблера IBM PC. – М.: Далог-МИФИ, 2005. – 288 с.: ил.

Содержание

Предисловие 3
Введение 5
Задание 1. Обработка многочленов 8
Задание 2. Частотный словарь 11
Задание 3. Вычисление корней уравнений и определенных интегралов 15
Задание 4. Методы сортировки 24
Задание 5. Арифметические вычисления 32
Задание 6. Обработка текстов 34
Задание 7. Работа с упакованными данными 37
Задание 8. Модули и макросы 43
Рекомендуемая литература 45

Учебное издание

БОРДАЧЕНКОВА Елена Анатольевна
ПАНФЁРОВ Антон Александрович

ЗАДАНИЯ ПРАКТИКУМА. 1 КУРС

Учебное пособие

Электронный вариант книги лежит на сайте *al.cs.msu.su*

Издательский отдел
Факультета вычислительной математики и кибернетики МГУ
имени М.В. Ломоносова
Лицензия ИД N 05899 от 24.09.01 г.

119992, ГСП-2, Москва, Ленинские горы,
МГУ имени М.В. Ломоносова,
2-й учебный корпус

Напечатано с готового оригинал-макета
в издательстве ООО “МАКС Пресс”.
Лицензия ИД N 00510 от 01.12.99 г.

Подписано в печать 30.03.2016 г.
Формат 60x90 1/16. Усл.печ.л. 3,0. Тираж 300 экз. Заказ 078.

119992, ГСП-2, Москва, Ленинские горы, МГУ им. М.В. Ломоносова,
2-й учебный корпус, 527 к.
Тел. 8(495)939-3890/91. Тел./Факс 8(495)939-3891