ПРОГРАММИРОВАНИЕ

Нижний Новгород
НГТУ им.Р.Е. Алексеева
2016г.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Р.Е. АЛЕКСЕЕВА»

Институт радиоэлектроники и информационных технологий

Кафедра «Вычислительные системы и технологии»

ПРОГРАММИРОВАНИЕ

Учебное пособие для студентов дневной формы обучения
по направлению подготовки 09.03.01
"Информатика и вычислительная техника"

Профиль подготовки
“Вычислительные машины, комплексы, системы и сети”

Квалификация (степень)
Бакалавр

Нижний Новгород
2016 год

Составитель Мартынов Д.С.

УДК 004.432.2

Рецензент – доцент, кандидат технических наук Д.В. Жевнерчук

Программирование: учеб.пособие/ Д.С. Мартынов; Нижегородский государственный технический университет им. Р.Е. Алексеева, 2016. – 00 с.

Данное пособие содержит учебный материал по основам алгоритмизации и программирования, задания для практических и лабораторных работ, контрольные вопросы по дисциплине "Программирование" в соответствии с учебным планом для направлению подготовки 09.03.01 "Информатика и вычислительная техника". Данная часть курса изучается в первом учебном семестре и ориентирована на изучение студентами языка Си в его спецификации *ANSI C11*. Программа дисциплины «Программирование» построена на изучение языков программирование Си и С++. На изучение каждого из языков программирования отводится один учебный семестр. Для закрепления навыков программирования на языке Си во втором учебном семестре студенты выполняют курсовую работу.

В приложении приведены примеры программ, использующие средства управления консольного форматного вывода в спецификации *ANSII C11*.

Редактор _____

Подп. к печ. _____Формат 60x84 $^{1}/_{16}$. Бумага офсетная. Печать офсетная. Печ. л. _6__. Уч.-изд. л. ____. Тираж ____ экз. Заказ ____.

Нижегородский государственный технический университет
имени Р.Е. Алексеева.
Типография НГТУ.
Адрес университета и полиграфического предприятия:
603950, г. Н.Новгород, ул. К.Минина, 24.

ОГЛАВЛЕНИЕ

СПИСОК СОКРАЩЕНИЙ

АЛУ – арифметико-логическое устройство
БУ – блок управления
ВС – вычислительная система
ВТ – вычислительная техника
ОЗУ – оперативное запоминающее устройство
ПЗУ – постоянное запоминающее устройство
ПО – программное обеспечение
С – язык программирования С (читается Си)
С++ – язык программирования С++ (читается Си плюс плюс)
УВВ – устройства ввода/вывода
УУ – устройство управления
ЦП – центральный процессор
ЭВМ – электронная вычислительная машина

ANSI – американский национальный институт стандартизации
ANSI C – стандартный язык программирования С
ASCI
CPU – центральное процессорное устройство
STDIN – стандартный поток ввода
STDOUT – стандартный поток вывода

ВВЕДЕНИЕ

Целью изучения дисциплины «Программирование» является формирование у студентов базовых знаний, умений и профессиональных компетенций, определенных требованиями Федерального образовательного стандарта (ФОС) для направления подготовки 09.03.01. Информатика и вычислительная техника». Предполагается, что студент имеет начальные сведения о работе с компьютером и владеет знаниями в объеме школьного курса информатики. В качестве языка для изучения программирования выбран Си – язык программирования общего назначения.

Данное пособие содержит базовый теоретический материал, задачи, задания к выполнению лабораторных работ по дисциплине «Программирование», выполняемые в первом учебном семестре. Даны рекомендации по составлению и описанию алгоритмов, по организации структуры программы, по оформлению сопроводительной документации и составлению отчетов к лабораторным работам.

Настоящее пособие не может заменить собой полноценный классический университетский учебник по программированию на языке Си, оно может лишь дополнить его, став путеводителем в удивительный мир программного обеспечения.

ЧАСТЬ 1. ВВЕДЕНИЕ В ДИСЦИПЛИНУ ПРОГРАММИРОВАНИЕ

1. Решение задач на ЭВМ

Компьютер – это устройство, способное производить вычисления, выполнять действия по принятию логических решений. Компьютеры обрабатывают данные, используя наборы инструкций, называемые компьютерными программами, которые заставляют компьютер выполнять определенные действия в заранее установленном порядке.

Компьютер, как вычислительная система, состоит из различных физических устройств, которые называются аппаратной частью, аппаратным обеспечением (*hardware*) ЭВМ. Программы, выполняющиеся на компьютере, называются программным обеспечением (*software*) или математическим обеспечением ЭВМ.

Отрасль, связанная с разработкой программного обеспечения (ПО) для ЭВМ, называется программирование. Процесс создания и использования программ получил обобщенное название – решение задач на ЭВМ.

При решении задач на ЭВМ программист решает несколько задач:

- определяет, что именно должна делать разрабатываемая программа;
- какие исходные данные должны быть ведены, и в какой форме должен быть получен результат;
- каким именно путем должен быть получен искомый результат, какие методы и алгоритмы следует использовать;
- какая структура вычислительного процесса будет порождена при выполнении программы на вычислительной системе, будет ли достаточно одной ЭВМ или потребуется вычислительная сеть, объединяющая несколько ЭВМ;
- достоверный ли получен результата и правильно ли работает созданная программа, то есть выполняет тестирование созданной программной системы.

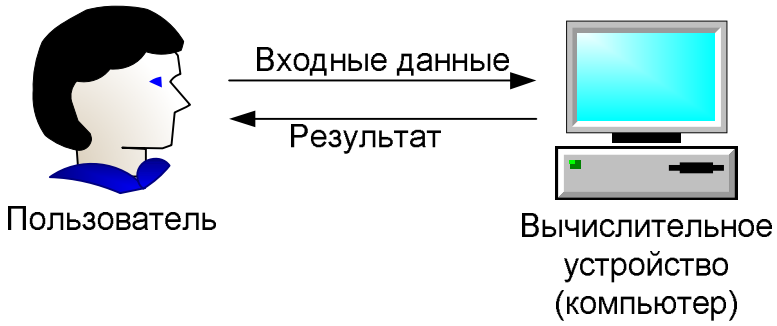


Рис.1. Упрощенная схема решения задачи на ЭВМ

Так или иначе, но именно программист определяет процесс решения задачи на ЭВМ, то есть создания компьютерной программы (рис.1.).

Очевидно, что одну и ту же задачу программист может решить разными способами, используя различные программные средства. Данный факт послужил причиной создания целой отрасли, получившей название технология программирования. Технология программирования определяет как перечень требований, предъявляемых к разрабатываемому программному обеспечению, так и определяет набор действий для программиста, регламентирующих процесс создания ПО. Процесс разработки и использования ПО получил название решение задач на ЭВМ.

Этапы решения задач на ЭВМ включают в себя:

1. Математическая формулировка задачи (формализация задачи), определяется, ***что нужно сделать***, то есть выполняется описание процесса обработки и/или преобразования информации. Любая задача подразумевает наличие входных данных, которые в процессе её решения преобразуются в выходные данные, в результат.
2. Выбор численного метода решения задачи данного класса, определяется, ***как это можно сделать***. После математической постановки задачи необходимо найти или заново разработать метод решения задач данного класса, то есть способ получения результата из исходных данных.
3. Разработка алгоритма решения задач данного класса (алгоритмизация), определяется, ***каким образом это должно быть сделано***.
4. Программирование разработанного алгоритма или метода, то есть запись конкретного алгоритма на языке конкретной ЭВМ.
5. Отладка программы.
6. Запуск программы с заданными входными данными.
7. Анализ полученного результата.

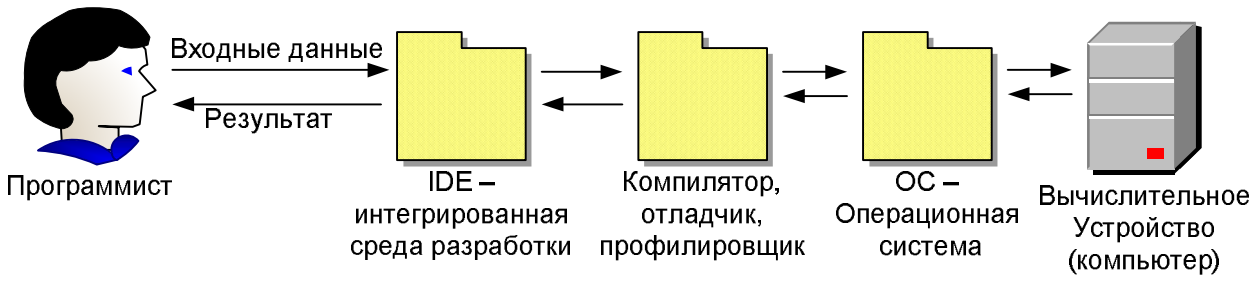


Рисунок 2. Схема взаимодействия программиста с компонентами вычислительной системы при разработке программного обеспечения

Самым важным фактором, определяющим качественные показатели решения задач на ЭВМ, является уровень квалификации программиста, именно он определяет возможность использования современных

инструментальных средств для проектирования, разработки, отладки и сопровождения программных систем. Человек – программист, пользователь, именно от него зависит, а будет ли решена задача, создана необходимая программа. Может быть создана уникальная по своим возможностям программа, но, если ее конечный пользователь не умеет ей грамотно, корректно пользоваться, то результат использования программы будет оставлять желать лучшего. В данном случае вполне уместно вспомнить ситуацию и рассказа Марка Твена «Принц и нищий», когда мальчик из бедной семьи, живя во дворце, использовал большую королевскую печать для разбивания орехов, так как не видел для нее другого применения.

Низкоуровневые языки программирования предполагают разработку программ, которые взаимодействующих с программными объектами, непосредственно отображаемыми в память ЭВМ. Это приводит к тому, что программа формируется в виде последовательности команд, которые выполняются центральным процессором (ЦП). Команда ЦП состоит из кода операции и одного или нескольких операндов. Именно такая организация программ характерна для большинства ассемблеров. Язык программирования *C* разрабатывался как язык общего назначения, ориентированный на максимально доступную производительность создаваемых программ. Это наложило существенный отпечаток на сам язык *C*, так и на другие языки, такие, как *Pascal*, *C++* и другие. Но в этих языках были реализованы соответствующие программные средства, позволяющие избежать неверных манипуляций с адресами переменных, что и привело к снижению быстродействия создаваемых на них программ.

Рассмотри задачу, определяющую процесс нахождения суммы двух целочисленных переменных. Псевдокод для решения данной задачи:

```
Дано: А: Целое, Б: Целое, Сумма: Целое
Начало
    Сумма ← 0
    Вывод «Введите значение для переменной А : »
    Ввод А
    Вывод «Введите значение для переменной Б : »
    Ввод Б
    Сумма ← А + Б
    Вывод «Сумма равна : » , Сумма
Конец
```

В данном примере отчетливо просматривается организационная структура кода: выделены блок объявления программных переменных и блок, отвечающий за вычисление. Действия, связанные с взаимодействием программы с пользователем, тоже являются частью вычислительного

процесса. Границы блока, определяющего вычислительный процесс, заданы метками «*Начало*» (*Begin*) и «*Конец*» (*End*). Каждая строка данного блока соответствует одно инструкции, то есть содержит в себе и команду, и ее операнды.

Команда ***Вывод*** предназначена для вывода сообщения на ***стандартное устройство вывода*** (***STDOUT***), она может быть выполнена как минимум с одним параметром, определяющим то, что именно нужно вывести. Команда ***Ввод*** предназначена для чтения данных со ***стандартного устройства ввода*** (***STDIN***), в ряде случаев подразумевается, что рассматривается ввод данных пользователем с клавиатуры, хотя это может быть и другое устройство ввода.

Псевдокод позволяет освободиться от ряда низкоуровневых абстракций, необходимых для описания процесса обработки информации. Например, процесс нахождения значения суммы для двух переменных на псевдокоде полностью описывается в виде математического выражения: «Параметр_1 + Параметр_2», в то время, как для уровня, близкого ассемблеру, потребуется запись вида:

«СЛОЖИТЬ» «Парметр_1» «Параметр_2».

Псевдокод является текстовой нотацией, позволяющей описывать программы и алгоритмы. Грамотное составление псевдокода позволяет детально описать структуры проектируемой программы и значительно повышает технологичность создаваемого ПО. Очень многие начинающие программисты недооценивают значимость проектного этапа программирования и сразу, без определения метода и средств решения задачи, пытаются писать программу, хотя им еще и не известно, что именно и как им придется запрограммировать…

Для большинства задач псевдокод состоит из трех секций: 1) заголовок, описывающий название решаемой задачи, список входных и выходных параметров и их типы; 2) секция, описывающая подготовку данных, в которой выполняется объявление используемых в задаче переменных; 3) секция инструкций, определяющих процесс нахождения решения.

```
1: Алгоритм «Название алгоритма» (Аргумент <Описание входных дан-
   |     ных>, Результат <Описание выходных данных>)
2: |     Дано <Описание исходных данных к решению задача> Данный блок
   |     содержит инструкции, связанные с выделением памяти для
   |     программных переменных
   |     Надо <Формальное описание модели решения задачи>
3: Начало Начало блока инструкций
   |     <Команды>
   Конец Конец блока инструкций
```

Такая организация псевдокода практически однозначно соответствует структуре организации кода программ, с которыми непосредственно взаимодействуют операционные системы и среды исполнения: сегмент кода (текст программы), сегмент данных (глобальные программные переменные), сегмент стека («свободная» динамическая память).

Язык программирования является инструментом, с помощью которого программист определяет набор инструкций для ЭВМ для решения определенной задачи. Одним из ключевых аспектов, определяющих возможности языка, является множество типов данных, которые поддерживаются данным языком.

Тип данных определяет множество возможных значений, которые могут быть представлены переменной данного типа, и множество операций, которые можно выполнять в отношении переменных данного типа. В языках программирования типы данных могут быть как встроенными, так и определяемыми пользователем. Также язык программирования определяет механизмы преобразования значений из одного типа в другой (операции приведения типа) и средства контроля за правомерность данных преобразований. Язык программирования Си относится к категории языков со слабой типизацией: он реализует ***слабый контроль*** за преобразованием программистом данных из одного типа в другой. В этом смысле язык Си предоставляет программисту практически неограниченную свободу при выполнении операций преобразования типа. В шутку можно сказать, что в этом смысле название языка Си созвучно сокращению от слова свобода.

2. Краткие сведения о языке Си

Язык программирования *C* (произносится “Си”) и был разработан в период с 1969 по 1973 года Денисом Ритчи и Кеном Томпсоном в фирме *Bell Labs*. История рождения языка слегка комична: по воспоминаниям Кена Томпсона во дворе университета они вместе с Денисом подобрали вычислительную машину PDP-7, отнесли в лабораторию и решили изменить имеющуюся на ней программу, моделирующую космическое путешествие. Они начали работать над созданием игрового автомата, а в результате произвели настоящую революцию в мире вычислительной техники.

Язык программирования *C* относится к категории компилируемых (исполняемых) языков программирования. Это значит, что результатом программирования является программа, то есть исполняемый программный модуль, который может быть запущен на вычислительном устройстве (ВУ), например, на персональном компьютере (ПК), под управлением операционной системы (ОС). Он предназначен для создания высокоэффективных программных систем, таких как системные утилиты, драйвера устройств, операционные системы и их подсистемы, и специализированных проблемно ориентированных программных систем. Благодаря использованию низкоуровневых абстракций и ряда других особенностей, язык программирования *C* позволяет создавать относительно небольшие программные системы. Принято считать, что объём кода для *C*-программ не должен превышать 100 тысяч строк инструкций на языке программирования. Это не значит, что *C* не позволяет создавать более крупные программные системы, просто поддержка очень больших проектов на языке *C* – это очень сложная задача, затраты на решение которой не сопоставим с получаемыми результатами.

Ввиду того, что язык программирования *C* относится к компилируемым языкам программирования, существует возможность использовать средства оптимизации и повышения эффективности кода, то есть исполняемый программный код будет создаваться на основании инструкций, написанных программистом, но он может отличаться от исходного варианта, хотя и выполнять возложенную на него вычислительную задачу.

Процесс создания программы и использование её для решения прикладной задачи принято называть решение задачи на ЭВМ.

При решении задач на ЭВМ программист решает несколько подзадач:

- определяет, что именно должна делать разрабатываемая программа;

- какие исходные данные должны быть ведены, и в какой форме должен быть получен результат;
- каким именно путем должен быть получен искомый результат, какие методы и алгоритмы следует использовать;
- какая структура вычислительного процесса будет порождена при выполнении программы на вычислительной системе, будет ли достаточно одной ЭВМ или потребуется вычислительная сеть, объединяющая несколько ЭВМ;
- достоверный ли получен результата и правильно ли работает созданная программа, то есть выполняет тестирование созданной программной системы.

Язык программирования *C* поддерживает процедурную парадигму программирования. Можно сказать, что программа, написанная на языке *C* представляет собой набор вызовов процедур (функций). Язык программирования *C* не поддерживает процедуры, с которыми, возможно, Вам уже приходилось знакомиться при изучении языков, подобных *Pascal*`ю. Функции могут быть как библиотечные, поставляемые вместе со средой программирования, компилятором, приобретаемые у поставщиков кода, так и разработанные сами программистом. Работа программы начинается с вызова, то есть активации, главной функции программы, которая называется *main* (главная функция). На некоторых платформах вместо функции `main` используются близкие по названию функции, которые также выполняют роль «главной функции», например, функция `_tmain` в языке *MS Visual C++*, либо функция `loop` в версии языка C++ для платформы *Arduino IDE*.

Программа, написанная на языке *C*, может содержать несколько программных модулей, то есть состоять из нескольких исходных файлов, но функция `main` в ней будет только одна. Функция `main` определяет «точку входа» в программу. С запуска функции `main` начинается работа программы.

Следует учесть, что одну и ту же задачу программист может решить разными способами, используя различные программные средства. Это послужило причиной создания целой отрасли, получившей название технология программирования. Технология программирования определяет как перечень требований, предъявляемых к разрабатываемому программному обеспечению (ПО), так и последовательность действий для программиста, определяющих процесс разработки этого ПО. Процесс разработки ПО получил название решение задач на ЭВМ.

Этапы решения задач на ЭВМ:

Принцип программного управления, реализуемый в низкоуровневых программно-аппаратных системах и языках программирования,

базируется на создании программ в виде последовательности команд, которые выполняет вычислительное устройство. Команда состоит их кода операции и одного или нескольких операндов. Именно такая организация программ характерна для большинства ассемблеров.

Принцип программного управления был сформулирован Норбертом Винером в 1947 году. Согласно ему процесс исполнения программы на ЭВМ описывается следующим образом:

Пункт 1. Загрузить программу из ВЗУ в ОЗУ.

Пункт 2. Вычислить адрес первой исполняемой инструкции. Поместить адрес инструкции в регистр адреса ЦП.

Пункт 3. Загрузить команду.

Пункт 4. Декодировать команду, вычислить количество операндов и их адреса.

Пункт 5. Загрузить операнды в регистры данных ЦП

Пункт 6. Выполнить машинную инструкции.

Пункт 7. Вычислить адрес следующей исполняемой инструкции и перейти к П.3.

С некоторым допущением можно сказать, что данное описание соответствует пошаговому исполнению программы, написанной на языке *C*. Большинство современных процессоров используют механизмы конвейерного исполнения команд, когда, согласно стратегии прогнозирования хода программы, исполняется определенная последовательность команд. Данный факт накладывает дополнительные ограничения на проектируемые программы особенно при использовании нелинейных алгоритмов, работе со структурами, массивами и вызове функций.

Дополнительно хочется отметить тот факт, что программы, написанные на *C*, как правило, имею прямой доступ к оборудованию вычислительного устройства, работают с реальными адресами в памяти ЭВМ. Поэтому ошибки, связанные с формирование неверных значений адресов программных переменных, приводят, как минимум, к сбоям в работе программной системы.

Язык программирования *C* достаточно прост и лаконичен, он поддерживает только процедурную парадигму программирования. Он дает программисту средства для прямого доступа к памяти, которые значительно повышают эффективность создаваемых программ. Но именно эти средства прямого доступа к памяти при неаккуратном использовании и приводят к сбоям в работе программ. Цена ошибки программиста, пишущего на *C* может привести к более тяжелым последствиям в работе программных систем, нежели ошибки программиста, использующего современные языки программирования высокого уровня, использующие

высокоуровневые абстракции и механизмы контроля среду исполнения кода программ.

Язык программирования включает алфавит, константы, ключевые слова, идентификаторы и комментарии. Компилятор языка *C* воспринимает исходный файл, содержащий программу на языке *C*, как последовательность текстовых строк. Каждая строка завершена символом новой строки.

Компилятор языка *C* последовательно считывает строки программы и разбивает каждую строку на группы символов, называемых лексемами. Эти строки программы и определяют машинные инструкции, которые сформирует компилятор после обработки исходного файла.

Процесс создания исполняемого модуля на языке *C*, соответствующего простой консольной программе, состоит из следующих операций:

1. создание исходного текста программы в виде текстового файла с расширение **“.c”**;
2. обработка исходного текста программы препроцессором. В настоящее время большинство компиляторов самостоятельно вызывают препроцессор без участия программиста. В результате работы препроцессора создается «образ» программы, файл с расширением **“.i”** (*image file*), представляющий исходный текст программы, дополненный необходимой для компиляции информацией.
3. Компиляция исходного текста программы. Данный процесс получил название трансляции, то есть перевода программы с языка программирования на язык машинных инструкций. В результате компиляции создается объектный файл, который не может быть непосредственно исполнен, он имеет расширение **“.o”** (*object file*). Для создания исполняемого файла объектный файл должен быть “собран» (слинкован).
4. Линковка объектного файла или набора файлов. Большинство функций стандартной библиотеки языка *C* поставляются в виде объектных файлов, которые подключаются к объектному файлу создаваемой программы (клиентской программе). Линковщик, программа, выполняющая функции линковки программы, может быть вызван либо непосредственно, либо через опции компилятора.

Программы, созданные на языке *C*, могут поставляться как в виде исполняемых модулей, так и в виде исходных файлов. В частности, в виде исходных файлов поставляются большинство *UNIX/Linux* систем.

ЧАСТЬ II. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Вперед, мой друг. Нас ждут великие дела.

Цель лабораторного практикума состоит в закреплении теоретических основ первой части курса "Программирование", получении практических навыков разработки алгоритмов при решении задач на ЭВМ создании программ на языке высокого уровня, к изучению предлагается «стандартный Си» (*ANSI C11*).

Исходя из требования российского законодательства в области защиты авторского и смежного права, студентам рекомендуется выполнять задания лабораторного практикума на программной платформе, работающей под управлением операционной системы (ОС) *UNIX/Linux* и использовать в своей работе так называемые *Open Source* программное обеспечение. В тоже время, не возбраняется использовать «свободные версии» коммерческого ПО, предоставляемые для учебных целей по академической лицензии, например *MS Visual C++* 10 *Express* и другое.

Все занятия лабораторного практикума для студентов дневного отделения проводятся в учебных классах вычислительного центра НГТУ им. Р.Е. Алексеева, оборудованных персональными компьютерами с установленной на них ОС *Linux*. Для студентов, не имеющих начальных знания по работе с *UNIX/Linux* системами, предусмотрены варианты «альтернативной загрузки» ОС. В качестве альтернативной ОС на ПК используется ОС *MS Windows*, с установленными пакетами интегрированной среды разработки ПО *MS Visual Studio* и офисного пакета *MS Office*, распространяющимися по академической лицензии.

Студенты, впервые приступающие к работе в учебном компьютерном классе, обязаны пройти инструктаж по технике безопасности и расписаться в соответствующем журнале.

1. Порядок выполнения лабораторных работ

1.1. Состав и содержание заданий к лабораторному практикуму

Первая часть курса "Программирование" посвящена изучению основ алгоритмизации и программирования, освоению языка программирования *C* – на это отводится первый учебный семестр. Лабораторный практикум в первом семестре предусматривает выполнение студентами пяти лабораторных работ по материалам изучаемых разделов. Лабораторные работы включают в себя несколько компонент. Первый компонент – это исследование предметной области, содержащее задания для углубленного изучению материала по заданной теме и выполнение заданий на ПК.

Второй компонент включает в себя непосредственно составление алгоритма и программы на языке для решения конкретной практической задачи. Созданные программы проверяется на работоспособность и технологичность, для этого студент должен разработать несколько тестовых примеров, несколько тестовых наборов входных данных, на которых будет проверена корректность созданного программного решения.

1.2. Подготовка к выполнению лабораторной работы по программированию

До начала занятия в компьютерном классе студент должен ознакомиться с заданием на лабораторную работу, выбрать алгоритм для решения задачи, описать алгоритм при помощи псевдокода, составить соответствующую ему блок-схему и создать программу на языке *C* для решения задачи соответствующего варианта (номер варианта задается преподавателем). Программа должна удовлетворять следующим основным требованиям:

а) массовость, т.е. должна быть работоспособной, без изменения текста программы для любых значений параметров задачи, удовлетворяющих заданным ограничениям (например, на размеры матриц);

б) дискретность, т.е. должна состоять из блоков, в каждом из которых решается самостоятельная "подзадача" (например, ввод исходного значения переменной, ввод массива с консоли, выдача матрицы на терминал). Для обеспечения читаемости программы всячески поощряется использование имен переменных, отражающих их функциональное назначение в программе (использование т.н. "венгерской нотации", например, `line` -- для номера строки матрицы, `matr` -- для двумерного массива и т.п.), дополнительных пробелов, табуляции, пустых строк, комментариев, при необходимости, и т.п. (для разделения функционально самостоятельных блоков программы), выделения вложенных циклов и т.д. (для улучшения читаемости текста программы);

в) должна быть защищена от неправильного ввода исходных данных. Для обеспечения этого, вводу параметров с консоли должны предшествовать текстовые приглашения, а после их ввода, для контроля правильности, они должны выводиться на консоль. Рекомендуется включать в программу блоки контроля значений вводимых параметров на их соответствие "физике" решаемой задачи, допустимому интервалу значений данного параметра, с выдачей соответствующих диагностических сообщений;

г) вывод на экран должен сопровождаться соответствующими пояснениями: цифровую информацию необходимо располагать в удобном для восприятия виде (матрицы, например, распечатывать в виде таблиц с соответствующим числом столбцов и строк);

д) технологичность разрабатываемого программного обеспечения (учебных программ). С точки зрения технологичности хорошим считается стиль оформления программ, облегчающий её понимание, как самим автором, так и другими программистами, которым, возможно, придётся её проверять или модифицировать.

Стиль оформления программы включает:

- правила именования программных объектов (переменных, функций, типов данных и пр.);
- правила оформления модулей;
- стиль оформления текстов модулей.

1.3. Правила именования программных объектов

При выборе имен для программных объектов (идентификаторов) следует придерживаться следующих правил:

– имя (идентификатор) должно соответствовать цели использования данного программного объекта, например:

MaxItem – максимальный элемент;

MinCount – минимальное значение;

NextItem – следующий элемент;

NextPtr – указатель на следующий элемент;

PrintTable – функция, выводящая значения элементов таблицы.

– если позволяет язык программирования, можно использовать символ нижнего подчеркивания «_» для визуального разделения компонент-лексем, составляющих идентификатор, например:

Max_Item, *Next_Item*;

– необходимо избегать близких по написанию имен, например:

Index и *InDex*,

такие идентификаторы запутывают и затрудняют понимание кода;

– исторический опыт программирования на *C* и *C++* сформировал две рекомендации по составлению имен программных объектов, это так называема «венгерская» нотация и *POSIX*-нотация:

maxItem – имена переменных скалярных типов начинаются со строчной буквы, если имя состоит из нескольких компонент, то каждая компонента, начиная со второй, пишется с прописной буквы.

max_item, *bad_opt* – имена, составляющие идентификатор, соединяются при помощи символа нижнее подчеркивание «_».

1.4. Правила оформления модулей

В настоящем, под словом модуль понимается отдельная функция, разрабатываемая программистом, в том числе и главная функция программы *main*(). В ряде случаев под модулем также может подразумеваться отдельный файл, библиотека функций.

Каждый программный модуль (функция) должен начинаться заголовком, который, как минимум, содержит:

- название модуля;
- краткое описание его назначения;
- краткое описание вводных и выходных параметров с указанием единиц измерения;
- список используемых (вызываемых) модулей;
- краткое описание алгоритма (метода) и/или ограничений;
- ФИО автора программы;
- идентифицирующую модуль информацию (номер версии и/или дату последней корректировки).

Пример оформления комментария к программному модулю:

```
/*************************************************************/
/*    Функция: Length_Path (n: int; L: double) : double      */
/*    Цель:  определение суммарной длины отрезков            */
/*    Исходные данные:                                       */
/*         n – количество отрезков                           */
/*         L – массив длин отрезков (в метрах)               */
/*    Результат:    длина (в метрах)                         */
/*    Вызываемые модули:  нет                                */
/*    Описание алгоритма:                                    */
/*      длины отрезков суммируются методом накопления, n ≥ 0 */
/*    Дата:  2016 / 09 / 01    Версия 1.01                   */
/*    Автор:  Сидоров П.В.                                   */
/*    Исправления:  нет                                      */
/*************************************************************/
```

1.5. Стиль оформления текстов модулей

Стиль оформления текстов модулей определяет использование отступов, пропусков строк и комментариев, облегчающих понимание программы. Большинство современных средств автоматизированной разработки программного обеспечения (*IDE* – интегрированные среды разработки) сами выполняют «грязную работу» по горизонтальному

выравниванию блоков кода, однако такие программные системы изначально ориентированы на высокопроизводительные рабочие станции, оснащенные дисплеем с высокой разрешающей способностью.

Начинающий программист должен научиться самостоятельно составлять хорошо читаемый исходный текст, это позволит ему глубже изучить возможности изучаемого языка программирования, так как использование пробельных символов, пустых строк и комментариев регламентируется правилами языка программирования.

В сообществе Си-программистов были выработаны две нотации формирования горизонтальных пробельных отступов - это нотации «2 и 2» и «4 и 4». Нотация «2 и 2» регламентирует использование 2 пробельных позиций каждый раз при входе в «блок операций». Нотация «4 и 4» -соответственно 4 символьных позиций. Между блоками кода рекомендуется вставлять пустые строки и строки комментарии, поясняющие назначение этих блоков.

Использование *IDE* освобождает программиста от рутинной работы, связанной с разметкой текста. Многие *IDE* оснащены встроенными семантическими анализаторами.

Пример 1

```
/* Блок поиска наименьшего значения в массиве целочисленных
значений. Размер массива: 10 элементов, Результат: minCount –
искомое минимальное значение. */
for(i=1, minCount = Arr[0]; i<10; i++) {
/*      Сравнение значения «текущего» элемента массива с   */
/*      найденным на предыдущем шаге цикла минимальным     */
/*      значением. Если условие «истинно», то формируется  */
/*      новое минимальное значение.                        */
  if(minCount>Arr[i]) {
      minCount=Arr[i];
}
```

Пример 2

```
/* Проверка на принадлежность контролируемого значения */
/* заданному интервалу                                 */
if(curCount<minCount || curCount>maxCount)
  {
     printf("Not the correct value.");
     /* блок обработки не корректного значения */
    . . .
 }
```

При составлении комментариев к исходному тексту программы (листингу) следует придерживаться следующих правил:

– комментарии должны помогать понять ход обработки данных в программе;

- комментарии должны описывать ограничения, налагаемые на используемые в программе данные;
- комментарии могут использоваться как средство визуального разделения блоков кода;
- не стоит комментировать очевидное.

Специфические требования к программам указаны в заданиях.

Пример 3. Оформления исходного текста программы на языке Си.

```
/* lab_01.c – Листинг программы для 1-ой лабораторной работы
*****************************************************************
* Filename : lab_01.c
* Abstract : This is sample C-program
* Description :
* Creation Date : 2013 / 09 / 10
* Author : Ivanov Ivan
 * Notes / Platform / Copyrights : OS Linux, FreeWare
*****************************************************************/
#include <stdio.h>

#define myPi 3.14159265358979323846
char msg1[] = "Nizhniy Novgorod Technical University\n"
              "Study work number 1. Task number 1.\n"
              "Performed student Ivanov Ivan , 13-IVT-3."

/* *****************   Main function   ***************************/
int main(void)
  {  /* *** BEGIN *** */
/******************   Local variables   ************************/
    int intVar1=10, intVar2=5;
/******************     Code Segment    ************************/
     printf(msg1);
     printf("\n constant PI = %lf", myPi);
     printf("\n%d + %d = %d\n", intVar1, intVar2,
(intVar1+intVar2));
       *      *      *      *      *      *
     printf("\n ========== End of program ==========\n");
     return 0;
  }  /* **** END **** */
```

Старайтесь проектировать и писать код таким образом, чтобы исходный текст содержал всю сопроводительную информацию в должном объеме.

Пример 4.

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

```
/* Функция main – “точка входа” в программу */
int main() {
    int num1;    /* первое число, вводимое пользователем   */
    int num2;    /* второе число, вводимое пользователем   */
    int Sum=0;   /* сумма введенных пользователем значений */
    printf(“Enter two integers: ”);
    /* считать из stdin два целых числа */
    scanf(“%d%d”, &num1, &num2);
    Sum=num1+num2;
    /* вывод результата */
    printf(“%d + %d = %d\n”, num1, num2, Sum);
    return 0;
}  /* Конец функции main */
```

Замечание. Начиная со стандарта *ANSI C*11 в языке *C* допускаются однострочные комментарии:

```
// Это однострочный комментарий в стиле C++.
```

1.6. Программирование и отладка учебных программ

На начальном этапе обучения студенту предлагается ознакомиться с возможностями компилятора, поэтому предполагается работа в консоли.

При разработке программы студент должен предусмотреть организацию программного диалога с пользователем таким образом, чтобы обеспечить работу с программой и в так называемом пакетном режиме. Для запуска программы в пакетном режиме необходимо подготовить текстовый файл, содержащий данные, подлежащие к вводу в программу через стандартный поток ввода (*STDIN*). Данные программе можно передать с использованием операторов *SHELL* "<<" и ">>". Для этого в консоли составляется команда для запуска созданного исполняемого программного модуля, вводится необходимый оператор перенаправления потока и указывается имя текстового файла.

Например, организовать перенаправление экранного вывода программы *Lab1_1* в файл *dest_1.txt* можно следующим образом:

"*./Lab_1 > dest_1.txt* [*Enter*]"

1.7. Оформление отчета о выполненной лабораторной работе

Отчеты следует оформлять либо в тонкой тетради, либо в папке на печатных листах формата *A*4. Содержание отчета, включая задание, описание алгоритма, должно быть оформлено в рукописном виде! Допускается вставка распечатанных исходных текстов программ и скриншотов «окон» работы программ. Данная мера обоснована использованием отдельными студентами чужих отчетов.

Каждый отчет должен содержать:

1) информацию о лице, выполнившем данную работу;
2) задание к лабораторной работе;
3) подробное описание метода решения задачи и используемых алгоритмов;
4) блок-схемы использованных алгоритмов;
5) исходные данные для контрольных примеров, на которых проверялась корректность работы разработанных учебных программ;
6) распечатку файлов исходных данных, текста программы и результатов ее работы (приклеить в сложенном виде).

После выполнения работы отчет должен быть сдан на проверку преподавателю. В случае выявления ошибок, как в самой работе, так и в оформлении отчета, студент должен исправить их и оформить отчет заново. Правильно и в полном объеме отчет визируется преподавателем, о чем ставится соответствующая запись в журнал текущей успеваемости.

1.8. Концепция проектирования программного обеспечения

При решении задачи проектирования программного обеспечения (ПО) необходимо исходить из того, что программа выполняется под управлением операционной системы (ОС) на конкретной аппаратной платформе: ЭВМ. Программа реализует заложенный в нее алгоритм либо метод решения задачи обработки данных.

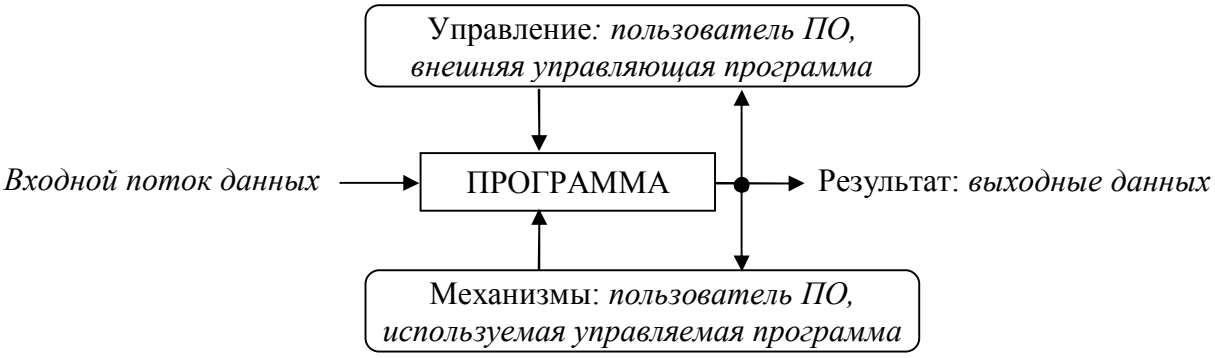


Рисунок 1. Общая схема функционирования программного обеспечения на ЭВМ

Общая схема функционирования ПО на ЭВМ под управлением ОС представлена на рис.1. Данная схема является базовой для большинства многозадачных ОС. Программа запускается пользователем, либо другой управляющей программной. Работая под управлением ОС программа использует ресурсы конкретной ЭВМ и, при необходимости, посредством вызовов сервисных функций, обращается к другим программам, которые предназначены для решения конкретных задач. Так в языке *C* функции стандартной библиотеки выделены в отдельные модули, поставляемые в виде объектных файлов. А «обращение» к этим функциям выполняется

посредством подключения библиотечных *header*–файлов. Правильнее сказать, что в *header*-файлах содержится информация для компилятора, каким именно образом можно вызывать эти функции, но подробно об этом будет рассказано чуть позже…

2. Задания для лабораторных работ

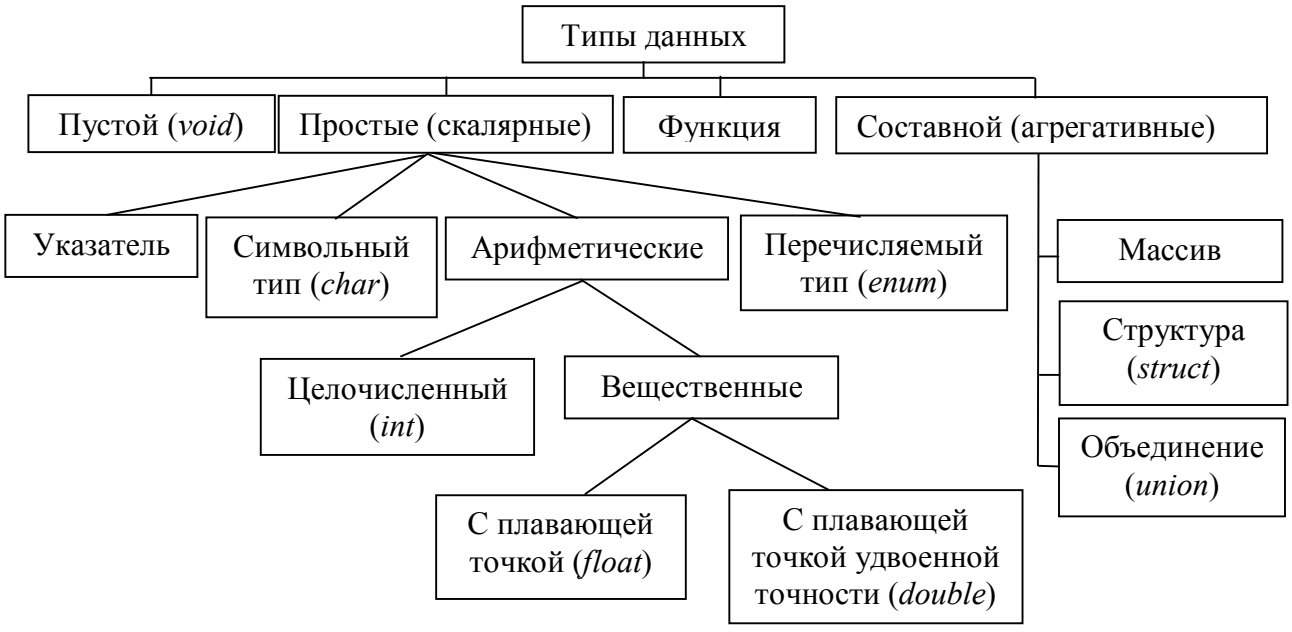
2.1. Лабораторная работа №1. Основы программирования на языке *ANSI C* в ОС *Linux*. Использование линейных алгоритмов.

Цели работы:

- Знакомство с возможностями программное оболочки ОС *Linux*.
- Изучение опций компилятора *GCC*.
- Освоение основных принципов разработки консольных приложений на языке *C*.
- Ознакомление со структурой базовых типов данных языка *C*.
- Использование линейных алгоритмов для решения простых вычислительных задач на ЭВМ.

Типы данных в языке *C*

Как уже было сказано ранее, тип данных – это базовая составляющая языка, которая определяет как множество значений, которое может быть представлено при помощи данного типа, так и множество операций, которые можно выполнять над программными переменными данного типа. Язык программирования *C* имеет набор базовых типов данных и средства для определения новых пользовательских типов данных.



Линейный алгоритм

Любой алгоритм может быть описан при помощи трех алгоритмических конструкций: следование, ветвление и цикл. Следование определяет процесс последовательного выполнения инструкций. Инструкции как бы выстроены в линию, именно поэтому данная организационная структура получила название линейной.

Рассмотрим структуру псевдокода, используемую для описания линейных алгоритмов.

```
Алгоритм «Название алгоритма» (Аргумент <Описание входных данных>,
|   Результат <Описание  выходных данных>)
|   Дано <Описание исходных данных к решению задачи>
|   Надо <Формальное описание модули решения задачи>
Начало <Подготовительные действия>
|   <Операция 1>
|   *     *     *
|   <Операция N>
Конец
```

Псевдокод для задачи нахождения суммы двух целочисленных значений/переменных.

```
Алгоритм Нахождение суммы двух целочисленных значений (Аргумент
Целое
| integer1, integer2, Результат Целое sum)
|   Дано Пользователь вводит значения для двух целочисленных
переменных
|   Надо Найти значение суммы: sum := integer1 + integer2
Начало sum := 0 // обнуление «аккумулятора»
|   Вывод «Input firs integer :»
|   Ввод integer1
|   Вывод «Input second integer :»
|   Ввод integer2
|   sum := integer1 + integer2
|   Вывод "Sun : %d + %d = %d", integer1, integer2, sum
Конец
```

Составьте псевдокод для решения следующих задач:

1. Вывести на экран значения числа Пи с точностью до десятитысячных.
2. Вывести на экран число, вводимого с клавиатуры. Выводимому числу должно предшествовать сообщение «Вы ввели число : ».
3. Вывести на одной строке числа 1, 7, 13, 23, 29, 31 с одним пробелом между ними.
4. Вывести столбиком четырех любых числе. Разрешается использовать не более одной инструкции вывода.
5. Вывести на экран следующей информации:

а)	б)	в)
* * * * * *	**********	* * *
* * * * *	* *	** ** *
* * * *	* ******** *	* * * * *
* * *	* * * *	* * * *
* *	* * * *	* * * *
*	* * * *	* * * * *
	* * *	** * * ***
	**********	* * *

г)	д)	е)
**********	****	***
**********	* **	** **
****	* **	*********
****	* **	** * * * * *
****	**********	**********
**********	* *** **	**********
**********	** * * **	~~~~~~~~~~~~~~~~~~~~

Домашнее задание – исследуйте логотипы программ, запускавшихся под управлением MS-DOS и других ОС, предоставлявших доступ к окну текстового терминала. Создайте логотип для соей программы.

Рекомендации по организации вывода логотипа в STDOUT

```
char logo[] =
“ *** ***  * *****\n”
“  *  *  * *   *\n”
“  *  ***  *   *\n”
“ *** *  * *   *\n”;
   .   .   .
printf(“%s”, logo);
```

Исследования возможностей платформы ОС *Linux* и средств компилятора *GCC* для разработки программ на языке *C*

Процесс ручной компиляции С-программы из командной строки Linux.

```
>> gcc –o <Имя_исполняемого_модуля> <Имя_исходного_модуля>
```

где *Исходный_модуль* – имя файла с расширение “.c”, содержащего исходный текст программы, *Исполняемый_модуль* – имя создаваемого исполняемого файла.

Например,

```
>> gcc –o A1 prog1.c
```

Из файла *prog1.c* будет создан исполняемый файл *C* – программы *A1*, запуск которого выполняется по схеме:

```
>> ./A1
```

где пара символов “./” обозначает текущую директорию.

Подробную информацию о возможностях компилятора можно получить, воспользовавшись встроенной справочной системой «*man*» (*manual* - руководство). Для получения справки о компиляторе в консоли нужно ввести команду: “*>> man gcc*”. Выход из справочной системы осуществляется нажатием клавиши *'q'*, которая соответствует команде *“quit”* (выход).

Задание:

– Изучить возможности командной строки (консоль *Linux* - *konsole*).

– Научиться использовать файловый менеджер (*mc – Midnight Commander*).
– В текстовом редакторе (*kate*, *Emacs* или другом) создать файл с расширением **“.c”**, содержащий исходный текст программы на языке *C* в.
– Скомпилировать созданную программу средствами компилятора командной строки *gcc*, изучить сообщений компилятора в случае наличия ошибок в исходном тексте программы.

Для получения навыков отладки и компиляции предлагается изучить несколько простых примеров программ, некоторые из которых заведомо содержат программные «дефекты». Наберите в текстовом редакторе исходный текст нижеприведенных примеров – листингов, сохраните их в виде файлов с расширением **“.c”**, скомпилируйте и запустите на исполнение.

Листинг 1.
```
int main(void) { }
```

Листинг 2.
```
/* Самая короткая программа на Си, которая «ничего не делает». */
int main(void) { return 0;}
```

Листинг 3.
```
main(void) {      /* Использование типа в ANSI C11 по умолчанию */
        return 0; /* приводит к непереносимости кода.           */
}
```

Листинг 4.
```
int main(void) { return 0;}
```

Листинг 5.
```
int main() {
    int A:=15;             /* Синтаксическая ошибка!       */
    return EXIT_SUCCESS;   /* Использование неопределенной */
}                          /* программной переменной       */
```

Листинг 6.
```
int main(){ /* Пример попытки «срыва стека». Большинство ОС   */
/* имеют «защиту» от подобного рода «программных дефектов».   */
    int *ptr = 0;    /* Создание переменной-указателя          */
    *ptr = 12345;    /* Попытка записи в пространство ядра…    */
    return 0;
}
```

Листинг 7.
```
int main(){
```

```
    int A;
    A = B + 3; /* Попытка обращения к необъявленной переменной */
    return 0;
}
```

Листинг 8.

```
int main(void) {
    printf("\n\tHello, World!\a");
    return 0;
}
```

Листинг 9.

```
/* Самая первая программа, с которой, как утверждают создатели */
/* языка Си, следует изучать программирование.                  */
#include <stdio.h>
int main(void)
    {
        printf("\n\tHello, World!\n");
        return 0;
    }

/* «Старые» версии компиляторов требовали, чтобы в конце листинга
обязательно была вставлена «пустая строка» */
```

Листинг 10. Простая программа “Hello, World!” на ANSI C99 (с использованием комментариев на русском языке)

```
/* figure: progr01                                                */
/****************************************************************/
/* Filename: progr01.c                                            */
/* Abstract: This is a sample C-program                           */
/* Description: Выводит две строки сообщения в STDOUT             */
/* Create Date: 2015 / 09 / 24                                    */
/* Author:                                                        */
/* Notes / Platform / Copyright   UNIX/Linux, FreeWare            */
/****************************************************************/

#include <stdio.h>          /* Подключение библиотеки <stdio.h> */

/* Работа программы начинается с вызова функции main()           */
int main(void)
  {  /* BEGIN: Начало кода главной функции модуля                */
     printf(“Hello, World\n”);
     printf(“Welcom to C!\n”);
     return 0; /* Возврат управления ОС в случае успешной работы */
  }  /* END: Конец кода главной функции модуля                   */
```

Листинг 11. Простая программа “Hello, World!”

```
/* figure: progr01                                                    */
/*********************************************************************/
/* Filename: progr01.c                                               */
/* Abstract: This is a sample C-program                              */
/* Description: Printing two lines with a single print               */
/* Create Date: 2015 / 09 / 24                                       */
/* Author:                                                           */
/* Notes / Platform / Copyright   UNIX/Linux, FreeWare               */
/*********************************************************************/

#include <stdio.h>     /* A directive to the C preprocessor          */
                       /* function main begins program execution     */
int main(void)
  {  /* BEGIN: Begin function main                                   */
     printf("Hello, World\n");
     printf("Welcom to C!\n");
     return 0;  /* indicate that program ended successfully         */
  }  /* END: End function main                                       */
```

Листинг 12. Простая программа "Hello, World!" на *ANSI C*11 (с использованием комментариев на русском языке) Замечание, по умолчанию для компиляции программ на *C* используется стандарт ANSI *C*99.

```
// figure: progr01
//*************************************************************
// Filename: progr01.c
// Abstract: This is a sample C-program
// Description: Выводит две стоки сообщения в STDOUT
// Create Date: 2015 / 09 / 24
// Author: Студент ИРИТ НГТУ Умнов А.В. 15-ИВТ-3
// Notes / Platform / Copyright   IRIT NNTU/ UNIX/Linux / FreeWare
//*************************************************************

#include <stdio.h>   // Подключение библиотеки <stdio.h>

// Работа программы начинается с вызова функции main()
int main(void)
  {  // BEGIN: Начало кода главной функции модуля
     printf("Hello, World\n");
     printf("Welcom to C!\n");
     return 0;  // Возврат управления ОС в случае успешной работы
  }  // END: Конец кода главной функции модуля
```

Листинг 13. Программа сложения двух целых чисел.

```
/* figure: progr02                                                    */
/*********************************************************************/
```

```
/* Filename: progr02.c                                                */
/* Abstract: This is a sample C-program                               */
/* Description:                                                       */
/* Create Date: 2015 / 09 / 24                                        */
/* Author:                                                            */
/* Notes / Platform / Copyright   UNIX/Linux, FreeWare                */
/********************************************************************/
#include <stdio.h>                  /* Подключение библиотеки
<stdio.h>  */
/* Работа программы начинается с вызова функции main() */
int main(void)
  {  /* BEGIN: Начало кода главной функции модуля*/
     int integer1; /* Первое слагаемое   */
     int integer2; /* Второе слагаемое   */
     int sum=0;    /* Результат сложения */
     printf("================================================\n");
     printf("= Enter first integer: ");
     scanf("%d", &integer1);
     printf("= Enter second integer: ");
     scanf("%d", &integer2);
     sum=integer1+integer2;
     printf("= Sum: %d + %d = %d\n", integer1, integer2, sum);
     printf("================================================\n");
     printf("== The program has completed its work. ==========\n");
     return 0; /* Возврат управления ОС в случае успешной работы */
  }  /* END: Конец кода главной функции модуля */
```

Листинг 14.

```
#include <stdio.h>
int globalValue=15; /* This is a global integer variable */
int main(void) {     /* Begin */
    int localValue=9;
    printf("\n globalValue = %d\n localValue = %d");
    return 0;
}  /* End */
```

Листинг 15.

```
#include <stdio.h>
int main(void) {
    int Var1, Var2, Dif, Sum;
    Var1=15, Var2=27;
    Sum=Var1+Var2;
    Dif=Var1-Var2;
    printf("\n\tVar1 = %d , Var2 = %d", Var1, Var2);
    printf("\n\t%d + %d = %d", Var1, Var2, Sum);
    printf("\n\t%d - %d = %d", Var1, Var2, Dif);
    printf("\n\t%d * %d = %d",)
    return 0;
```

```
}
```

Листинг 16.

```
#include <stdio.h>
/* ********************************************************** */
char msg1[40]="\n\tHello, World!\n";
char msg2[40]="\n\tThis is a sample of C-program.\n";
char msg3[40]="\n\tPress ane key to continue.\n";
void ShowMsg(char* );
void DrawLine(int);
/* ********************************************************** */
int main(void) { /* BEGIN */
    ShowMsg(msg1);
    DrawLine(45);
    ShowMsg(msg2);
    DrawLine(45);
    ShowMsg(msg3);
    getchar();
    getchar();
    return 0;
}  /* END */
/* ********************************************************** */
void ShowMsg(char* msg) {
    puts(msg);
}
void DrawLine(int len) {
    int i;
    for(i=0; i<len; i++)
           putchar('*');
}
```

Листинг 17.

```
//пример использования функции gets, считывания одной строка
#include <stdio.h>
int main() {
    char string [256];
    printf("Введите свой полный адрес: ");
    gets(string); // считать строку из стандартного потока ввода
    printf("Ваш адрес: ", string);
    return 0;
}
```

Листинг 18.

```
int main(){
    int A = 2, B = 3;
    double Div = A/B;
    printf(“A / B = %lf\n”, (A/B));
    printf(“Div = %lf\n”, Div);
```

```
    return 0;
}
```

Листинг 19.

```
/* Результат работы данной программы зависит от программной платформы, на которой разрабатывается и запускается программа */
int main(){
    double A;
    /*  */
    A = 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 +0.1 + 0.1 + 0.1 + 0.1;
    printf("A = %lf\n", A);

    return 0;
}
```

Листинг 20.

```
/* Исследование механизмов передачи параметров функции printf() */
int main(){
    int A = 2, B = 3, Sum = A+B;
    printf("A  = %d, B = %d\n", A, B);
    printf("%d + %d = %d", A, B);
    return 0;
}
```

Создайте программы на основании кода вышеприведенных примеров, откомпилируйте и запустите их. Данная работа выполняется в компьютерном классе. Подробно опишите в отчете процесс компилирования и отладки созданных программ, объясните результат работы.

Все представленные программные примеры описываются при помощи линейных алгоритмов и могут быть представлены с помощь блок-схемы следующего вида (рис.3).

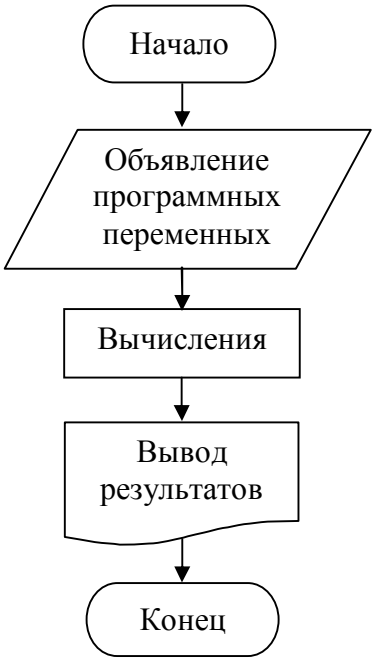


Рисунок 3. Упрощенная блок-схема линейного алгоритма

Разработка простых консольных программ на языке С

Задание

1. Изучить опции запуска компиляторов *GCC*.
2. Создать, отладить и запустить на исполнение простые консольные программы, написанные на языке *ANSI C*.

   Рекомендации по компиляции учебных программ:

   1. Запустить консоль (терминал)
      Меню« ВЫПОЛНИТЬ»команда «*konsole*»
   2. В текущей директории проекта выполнить команду:
      “*gcc –o Axx task_xx.c*“<*Enter*>

где *Axx* – имя компилируемого модуля; *task_xx.c* – имя исходного файла программы

3. Создать программы согласно следующим заданиям:
   1. Напишите программу, запрашивающую у пользователя фамилию, имя и отчество (например: "What is your name?", или "Здравствуй, как тебя зовут?") и выводящую полученные данные в приветствии (например: "*Hello, Ivanov Andrei Petrovich*", или "Здравствуй, Иванов Иван Иванович.").
   2. Напишите программу, печатающую целые числа в различных форматах (десятичном, восьмеричном, шестнадцатеричном) с различными флагами, с различной шириной поля. Вывод значения на экран должен быть пояснен предварительным сообщением, например: «10: 11 | 8: 011 | 16: 0xB»

3. Напишите программу, печатающую значения с плавающей точкой в разных форматах (обычная десятичная и экспоненциальная форма записи) с разными флагами, с разной шириной поля и точностью.
4. Напишите программу, вычисляющую площадь прямоугольника. Значения длин сторон должны вводиться с клавиатуры.
5. Напишите программу, переводящую значения веса из фунтов в килограммы.
6. Напишите программу, вычисляющую сумму вклада (исходные данные: сумма вклада, процентная ставка и время размещения вклада). Вычисление значения конечной суммы вклада осуществляется по формуле суммы членов геометрической прогрессии.
7. Напишите программу, вычисляющую площадь круга. Значение константы Пи определить с помощью директивы препроцессора #define.
8. Напишите программу, вычисляющую площадь кольца. Значение константы Пи определить с помощью директивы препроцессора #define.
10. Индивидуальное задание, выполняемое по вариантам, указанным в табл.1. Необходимо составить блок-схему алгоритма и написать программу на языке Си для вычисления значений переменных Y и Z по заданным формулам.

Требования, предъявляемые к разрабатываемым программам:

1) массовость: программа должна быть работоспособной без изменения текста программы, для любых значений параметров задачи, удовлетворяющих заданным ограничениям;
2) дискретность: программа должна состоять из блоков, в каждом из которых решается самостоятельная "подзадача". Для обеспечения читаемости программы рекомендуется использовать имена переменных, отражающие их функциональную нагрузку;
3) программа должна быть защищена от неправильного ввода данных, то есть, в логику работы программы должен быть заложен механизм анализа допустимости вводимых данных;
4) в начале работы программа должна выводить служебное сообщение, описывающее ее назначение (логотип программы), например:

```
************************************************************
*  Нижегородский государственный технический университет  *
*  Лабораторная работа №1. Задание 1.                      *
*  Выполнил студент группы _____  .________________________ *
************************************************************
```

или

```
*********************************************************
*  Nizhniy Novgorod Technical University                *
*  Study work number 1. Task number 1.                  *
*  Performed student _________  ____________________    *
*********************************************************
```

5) вывод информации, равно как и запрос на ввод данных, должен сопровождаться соответствующими пояснениями.

Программу необходимо реализовать с использованием команд форматного ввода-вывода *IO* (*Input/Output*) *printf*(), *scanf*(). Программа должна выполнять следующие действия: 1) вывести приглашение для ввода значений переменных; 2) произвести проверку корректности вводимых значений; 3) вывести результаты вычислений.

Для вычислений можно использовать математические функции, входящие в состав стандартной библиотеки языка Си, описываемые в файле *math.h*.

`sin(x)`, `cos(x)`, `tan(x)` – тригонометрические функции вычисления значений синуса, косинуса и тангенса от заданного значения угла, соответственно (значение ***x*** задается в радианах);

`asin(x)`, `acos(x)`, `atan(x)` – обратные тригонометрические функции `arcsin x`, `acrcos x`, `arctan x` соответственно;

`exp(x)`, `sinh(x)`, `cosh(x)` – экспонента и гиперболические функции;

`log(x)` – логарифм натуральный от *x*, `log10(x)` – логарифм десятичный от *x*;

`sqrt(x)` –корень квадратный из *x*, причем $x \geq 0$, `fabs(x)` – абсолютное значение *x*;

`ceil(x)` – наименьшее целое, большее чем *x*;

`floor(x)` – наибольшее целое, меньшее чем *x*;

`pow(x,y)` – возведение *x* в степень *y*.

Аргументы и значения, возвращаемые всеми математическими функциями, имеют тип `double`. Дополнительную информацию о функциях, входящих в стандартную библиотеку языка *ANSI C11* можно получить из встроенной в ОС системы помощи «manual», для этого в окне терминала нужно ввести строку запроса вида:

```
$man 3 func_name<Enter>,
```

где `func_name` – имя функции, по которой необходимо получить справочную информацию. Для выхода из текстового редактора, в котором загружается для просмотра файл встроенной документации, необходимо

начать клавишу ‘*q*’, что соответствует команде *Quit*. Например, для того чтобы получить справку по функции *cos()*, необходимо в командной строке ввести:

```
$man 3 cos<Enter>
```

Сигнатура функции *soc()* – *double cos(double)* обозначает, что функция принимает агрумент типа *double* и возвращает значение типа *double*. Необходимо учесть, что тригонометрические функции принимают значения аргумента из диапазона значений, для которого они определены. Это связано с тем, что при реализации библиотечных функций были использованы методы приближенных вычислений, основанные на разложении функций в функциональный степенной ряд.

Замечание 1. Компилятору *GCC* необходимо указывать опцию *-lm* (*linking math*) для подключения библиотеки *<math.h>*, поэтому, для компиляции программы, составленной по заданию №10 можно использовать компилятор *C++*:

```
$C++ -o task10 task10.c <Enter>
```

Замечание 2. Каждая программная задача, решаемая в рамках лабораторного практикума, должна быть описана при помощи псевдокода. Ниже приведен пример псевдокода для задачи нахождения суммы трех целочисленных значений.

Таблица1. Индивидуальные задания по вариантам

1. $Y=\sqrt{x^2+a}$ $Z=\arcsin(x^3-a)$	при a=0.01 x=0.12	9. $Y=\ln 2x^3+a^{3/2}$ $Z=3.7\mathrm{tg}^2 2x$	при a=2.53 x=0.7
2. $Y=\frac{\sqrt{x^{1.5}}}{a^2}$ $Z=\cos(3.56(x+a))$	при a= -5.1 x=4.78	10. $Y=\arccos x+a^2$ $Z=\ln(x^3+\cos a)$	при a=0.75 x=0.14
3. $Y=2e^{4x}+\mathrm{arctg}(\frac{x}{a})$ $Z=\cos x^3+\sin^2 x$	при a=2.8 x=1.29	11. $Y=\lvert\cos(x-a^3)\rvert$ $Z=e^{2a}-\mathrm{arctg}(2x)$	при a=1.5 x=3.3
4. $Y=\ln\lvert\sin(x+a)\rvert$ $Z=\mathrm{tg}(xe^x)^2$	при a= -3.4 x=2.75	12. $Y=2e^{4x}+\arccos\left(\frac{x}{a}\right)$ $Z=\cos x^3+\sin^2 a$	при a=0.6 x=0.3
5. $Y=\left(\frac{x}{2}\right)+a^3$ $Z=tg\left(e^x+\cos a\right)$	при a= 0.34 x=0.02	13. $Y=\ln\lvert\cos(x+a)\rvert$ $Z=\mathrm{arcctg}\left(ae^x\right)^3$	при a=-3 x=0.6
6. $Y=\ln(1.5x)+a^4$	при a=2.50	14. $Y=\frac{\sqrt[3]{x}}{a^2}+\sin(a)$	при a=0.35 x=-0.78

$Z = \text{arctg}\left(\cos\frac{a}{x^2}\right)$	x=3.11	$Z = \arccos(e^x a)^5$	
7. $Y = \cos\left(x^3 + a^3\right)$ $Z = \text{arctg}\sqrt{2x + a}$	при a=2.48 x=0.21	15. $Y = -\sqrt{\frac{4x^2 + a^2}{3}}$ $Z = \frac{5\cos^3(xa) - 1}{\sqrt[3]{xa} + 2{,}5}$	при x=1 a=2.3
8. $Y = \left\|\sin\left(x - a^2\right)\right\|^4$ $Z = e^{2x} + \arccos(2x + a)$	при a=0.35 x=0.21	16. Y=2a ^{x}+ ln|x+a^3| $Z = \sqrt[5]{\frac{xa^2}{1 - tg(x - a^2)}}$	При x=2.1 a= 0.2

Контрольные вопросы для собеседования

1. Функциональная схема Фон-неймановской вычислительной машины.
2. Принцип программного управления, сформулированный Норбертом Винером.
3. Структурная схема исполняемого программного модуля, работающего под управлением ОС *UNIX/Linux*.
4. Структура простой консольной программы на языке *C* (рекомендуемая и для *C*++). Роль разметки исходного текста программы. Рекомендации по разметке исходного текста программы. Роль и место комментариев в исходном тексте программы на языке *ANSI C11*.
5. Этапы решения задач на ЭВМ. Понятие алгоритма и его свойства. Процесс создание исполняемой программы на языке *ANSI C11*.
6. Структура и состав языка *C*. Ключевые слова в языке *C*, что они определят и какую смысловую нагрузку несут.
7. Базовые типы данных в языке *C* и машинное представление данных. Модификаторы типов данных.
8. Идентификаторы. Объявление и инициализация переменных. Венгерская нотация, *POSIX*- нотация именования программных объектов.
9. Область видимости переменных: глобальные, локальные, автоматические, регистровые, внешние. Каким образом область видимости связана с понятием время жизни программной переменной.
10. Стандартные потоки: *STDIN*, *STDOUT*, *STDERR*. Использование функций `printf()`, `scanf()` для организации пользовательского ввода/вывода в консольных программах. Спецификации форматов ввода-вывода функций `printf()`, `scanf()`.

Таблица 2. Использование формата вывода для функции `printf()`

prefix	6d	6o	8x	10.2e	10.2f
%-+#0	+555	01053	0x22b	+5.50e+00	+5.50

%-+#	+555	01053	0x22b	+5.50e+00	+5.50
%-+0	+555	1053	22b	+5.50e+00	+5.50
%-+	+555	1053	22b	+5.50e+00	+5.50
%-#0	555	01053	0x22b	5.50e+00	5.50
%-#	555	01053	0x22b	5.50e+00	5.50
%-0	555	1053	22b	5.50e+00	5.50
%-	555	1053	22b	5.50e+00	5.50
%+#0	+00555	001053	0x00022b	+005.50e+00	+0000005.50
%+#	+555	01053	0x22b	+5.50e+00	+5.50
%+0	+00555	001053	0000022b	+005.50e+00	+0000005.50
%+	+555	1053	22b	+5.50e+00	+5.50
%#0	000555	001053	0x00022b	005.50e+00	0000005.50
%#	555	01053	0x22b	5.50e+00	5.50
%0	000555	001053	0000022b	005.50e+00	0000005.50
%	555	1053	22b	5.50e+00	5.50

Домашнее задание

Запишите по правилам языка С следующие выражения.

2.2. Лабораторная работа № 2. Нелинейные и циклические алгоритмы

Использование условного оператора.

Алгоритмическая конструкция ветвления определяет ситуацию, соответствующую ситуации выбора пути дальнейшего следования на основании результатов проверки определенного условия, именно поэтому данную конструкция называют условием или условным оператором

Условный оператор подразумевает наличие двух альтернатив. Общая форма записи алгоритма условного оператора

Если <Условное выражение> :
| **То** <Действие1>
| **Иначе** <Действие2>
Все-если

Рассмотрим в контексте условного оператора изречение Рене Декарта: “*Ego cogito, ergo sum*” (лат. — «*Я мыслю, следовательно, существую*»).

Если «Я мыслю»
| **То** «Я существую»
| **Иначе**
Все-если

Получили вариант условного оператора с «пустой» альтернативной ветвью. Можно опускать «пустые» альтернативы, то есть использовать сокращенный условный оператор в формате:

Если <Условное-выражение>
| **То** <Действие>
Все-если

Синтаксис условного оператора в языке Си описывается следующим образом:

```
if ( <Условное_выражение> )
     <Выражение1>
else
     <Выражение2>
```

Блок-схема, соответствующая операции выбора (ветвление), изображена на рисунке 4.

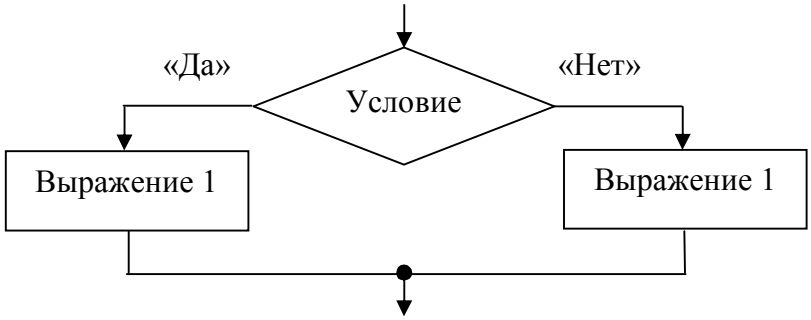


Рисунок 4. Блок-схема оператора выбора

Часто условие в операторе выбора называется управляющим выражением. Результат вычисления управляющего выражения оценивается в двоичной системе: «ноль» и «не ноль». В ранних версиях языка программирования С нет логического типа, вместо него используются следующие эквиваленты: «ложь» - целочисленный ноль, «истина» («не ноль») – целочисленное значение, отличное от нуля.

Задание. Необходимо составить алгоритм и соответствующую ему программу на языке Си для решения следующих задач:

1. Если целое число *M* делится нацело на целое *N*, то вывести на экран частное от деления, в противном случае вывести сообщение «M на N не делится без остатка».
2. Определить, является ли число *A* делителем для *B*.
3. Дано целое число. Определить, является ли оно четным и оканчивается ли оно цифрой 4.

4. Дано целое число. Определить, является ли оно нечетным и оканчивается ли оно цифрой 7.
5. Дано целое число. Определить, оканчивается ли оно четной цифрой (составное условие не использовать).
6. Известны год и месяц рождения человека, а также год и номер месяца сегодняшнего дня. Определить возраст человека, число полных лет. В случае совпадения указанных месяцев считать, что прошел полный год.
7. Известны два расстояния: одно в километрах, другое – в футах (1 фут = 0,45м). Какое расстояние меньше?
8. Известны две скорости: одна в километрах в час, другая – в метрах в секунду. Какая из скоростей больше?
9. Даны радиус круга и сторона квадрата. У какой фигуры площадь больше?
10. Известны площади круга и квадрата. Определить, уместится ли круг в квадрате.
11. Дано двузначное число. Определить, какая из его цифр больше, первая или вторая.
12. Дано двузначное число. Определить, одинаковы ли его цифры.
13. Дано двузначное число. Определить, равен ли квадрат этого числа учетверенной сумме кубов его цифр.
14. Дано двузначное число. Определить, является ли сумма его цифр двузначным числом
15. Дано двузначное число. Определить, кратна ли трем сумма его цифр.
16. Дано трехзначное число. Выяснить, является ли оно палиндромом («перевертышем»), то есть числом, десятичная запись которого читается одинаково слева направо и справа налево.
17. Дано трехзначное число. Определить, равен ли квадрат этого числа сумме кубов его цифр.
18. Дано трехзначное число. Определить, является ли сумма его цифр двузначным числом.
19. Дано трехзначное число. Определить, является ли произведение его цифр трехзначным числом.
20. Дано трехзначное число. Определить, кратна ли пяти сумма его цифр.
21. Дано трехзначное число. Верно ли, что все его цифры одинаковые?
22. Дано трехзначное число. Определить, есть ли среди его цифр одинаковые.
23. Дано четырехзначное число. Определить, равна ли сумма двух первых его цифр сумме двух его последних цифр.
24. Дано четырехзначное число. Определить, кратно ли четырем произведение его цифр.

25. Дано натуральное число. Верно ли, что оно заканчивается нечетной цифрой?
26. Дано натуральное число. Верно ли, что оно заканчивается четной цифрой?
27. Дано вещественное число. Верно ли, что его целая часть заканчивается четной цифрой?
28. Дано вещественное число. Верно ли, что его целая часть кратна трем?
29. Даны три вещественные числа. Возвести в квадрат те из них, значения которых неотрицательные, и в четвертую степень – отрицательные.
30. Даны две точки: $A(x_1, y_1)$ и $B(x_2, y_2)$. Составить алгоритм для определения того, какая из точек находится ближе к началу координат.
31. Даны вещественные числа x и y, не равные друг другу. Меньшее из этих двух чисел заменить половиной их суммы, а большее – их удвоенным произведением.
32. На плоскости $X0Y$ задана своими координатами точка A. Указать, где она расположена (на какой оси или в каком квадранте).
33. Даны целые числа n, m. Если числа не равны, то заменить каждое из них значением их суммы, а если равны, то числа заменить нулями.
34. Подсчитать количество отрицательных чисел среди a, b, c.
35. Подсчитать количество положительных чисел среди a, b, c.
36. Подсчитать количество четных чисел среди a, b, c. Условный оператор не использовать.
37. Подсчитать количество нечетных чисел среди a, b, c. Условный оператор не использовать.
38. Заданы три числа, a, b, c. Есть ли среди них ровно два четных числа.
39. Заданы три числа, a, b, c. Есть ли среди них ровно два нечетных числа.
40. Перераспределить значения переменных A и B таким образом, чтобы в A оказалось большее значение, а в B – меньшее.
41. Определить правильность даты, введенной с клавиатуры (число – от 1 до 31, месяц - от 1 до 12). При формировании ответа необходимо учитывать реальное количество дней в месяце. Для февраля учитывать количество дней в високосном году.
42. Для введенного значения даты подсчитать количество дней, прошедших с начала эпохи (1 января 1970г.).
43. Для введенного значения даты определить номер квартала, которому принадлежит данная дата и число дней, прошедших с начала этого квартала.
44. Для введенного значения времени (ЧЧ – час, ММ – минут, СС-секунд) определить количество полных минут и количество секунд,

прошедших сначала суток. Также определить количество полных минут, прошедших с начала четверти часа, которой соответствует указанный момент времени.

45. Написать программу, которая анализирует данные о возрасте и относит человека к одной из четырех возрастных групп: дошкольник, ученик, работник, пенсионер. Значение возраста вводится с клавиатуры.

46. Составить программу, определяющую, пройдет ли график функции $y=Ax^2+Bx+C$ через заданную точку с координатами (M,N).

47. Даны значения длин сторон кирпича A, B, C. Определить, можно ли переместить через отверстие прямоугольного сечения с длинами сторон D и E переместить кирпич указанного размера.

48. Заданы три положительные числа A, B и C. Определить, являются ли они последовательно стоящими элементами арифметической или геометрической прогрессии.

49. С клавиатуры вводятся длины отрезков A, B, C и D. Определить их на возможность построения треугольников.

50. Даны три точки: $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$. Определить, расположены ли они на одной прямой. Если нет, то вычислить угол B.

2.2.2. Использование циклического оператора.

Язык программирования *C* поддерживает три типа циклов: итерационный цикл, цикл с предусловием и цикл с постусловием.

Начало-цикла **Для** <параметр_цикла> **от** <начальное значение>
до <конечное значение>, **шаг** <величина цикла>
| <Тело цикла>
Конец-цикла

Язык псевдокода допускает использование следующих сокращений:

Начало-цикла – **НЦ**, **Конец-цикла** – **КЦ**, таким образом пример, приведенный выше, может быть записан:

НЦ **Для** <Параметр цикла> **от** <начальное значение>
до <конечное значение>, **шаг** <величина шага>
| <Тело оператора цикла
КЦ

Разрешается и другая нотация для описания счётного цикла:

Для <индекс> = <N> <K>, <H>
| <Тело оператора цикла>
Все-цикл

Где переменные *N* – начальное значение, *K* – конечное значение, а *H* – шаг изменения счётчика цикла.

Согласно основной теореме алгоритмизации, любая алгоритмическая конструкция может быть представлена в виде комбинации трех алгоритмических структур: следование, ветвление и цикл. Условный оператор является неотъемлемой частью циклического оператора. Как видно из представленных описаний в виде псевдокода, циклические операторы отличаются условиями поверки выхода из цикла. Цикли являются неотъемлемой частью нашей жизни. Любая программа, предназначенная для продолжительной работы, имеет не менее одного цикла.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    FILE *fp=NULL;
    int ch;
    char file_name[40]; /* Имя файла */
    printf("Enter the file name: ");
    gets(file_name); /* Чтение имени файла из STDIN */
    fp=fopen(file_name, "r"); /* Открытия файла для чтения */
    /* Если попытка открытия файла выполнена неудачно */
    if(fp==NULL) {
        printf("Can`t open file: %s", file_name);
        return EXIT_FAILURE;
    }
    /* Посимвольное стение данных из файла не эффективно */
    /* Чтение сомвола из файлового потока */
    while((ch=fgetc(fp))!=EOF) {
         fputc(ch, stdout);
    }
    fclose(fp); /* Закрытие файлового потока */
    fp=NULL;    /* "Защита" указателя на файловый поток */
    return EXIT_SUCCESS;
}
```

Задания для закрепления темы

Формализуйте, то есть опишите при помощи псевдокода, следующие процессы/задания:

1. Пробежать десять кругов на стадионе.
2. Съесть три котлеты.
3. Вымыть всю посуду.
4. Вручную, не в посудомоечной машине, вымыть одну тарелку.

5. Упаковать багаж в чемодан.
6. Сделать уроки.
7. Выполнить новостной обзор на заданную тему.
8. Почистить зубы.

Задачи для практической работы в классе

1. Найти сумму вводимых с клавиатуры положительных чисел (пользователь может вводить и отрицательные значения). Сигналом к вычислению результата является ввод нуля, символизирующего конец последовательности.
2. Найти сумму всех целых чисел, больших -50 и меньших 200, которые кратны 5 и 8 и заканчиваются на 5 или 0.
3. Дано натуральное число. Определить количество единиц в записи данного числа в двоичной системе счисления. Например, в двоичной записи числа 5 содержится 2 единицы (5(10) = 101(2)).
4. Даны натуральные числа *M* и *N*. Определить их наименьшее общее кратное.
5. Для заданного числа найти все его делители.
6. В заданном натуральном числе поменять порядок цифр на обратный и сравнить полученное число с исходным.
7. Напечатать числа следующим образом:

а)
10 10.4
11 11.4
. . . . . .
25 25.4

б)
21 20.4
22 21.4
. . . . . .
35 34.4

в)
25 25.5 24.88
26 26.5 25.88
. . . .

г) выявить закономерность формирования значения для ниже представленного набора и, используя полученное правило, вывести не столбиком не менее 8 значений.

01 00001
02 00005
03 00025
04 00125

05 00625
06 03125

Задачи для практической работы, выполняемой самостоятельно

Данный блок заданий выполняется студентами по вариантам.

Задание 1. Используя циклический оператор решить следующие задачи, составить алгоритм решения задачи и соответствующую ему программу на языке *C*.

1. Дано натуральное число *n*. Вычислить $S = 1 - \frac{1}{2} + \frac{1}{4} - \frac{1}{8} + ... + (-1)^n \frac{1}{2^n}$.
2. Дано натуральное число *n*. Вычислить произведение первых n сомножителей $P = \frac{2}{3} \cdot \frac{4}{5} \cdot \frac{6}{7} \cdot ... \cdot \frac{2n}{2n+1}$.
3. Дано вещественное число *x*. Вычислить $x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \frac{x^{13}}{13!}$
4. Даны натуральное число *n* и вещественное число *x*. Вычислить $S = \sin x + \sin\sin x + ... + \underbrace{\sin\sin...\sin x}_{n \text{ раз}}$
5. Даны вещественное число *a* и натуральное число *n*. Вычислить $P = a(a+1)(a+2)...(a+n-1)$.
6. Даны вещественное число *a* и натуральное число *n*. Вычислить $P = a(a-n)(a-2n)...(a-n^2)$
7. Даны вещественное число *a* и натуральное число *n*. Вычислить $P = \frac{1}{a} + \frac{1}{a^2} + \frac{1}{a^4} + ... + \frac{1}{a^{2n-2}}$
8. Дано вещественное число *x*. Вычислить $D = \frac{(x-1)(x-3)(x-7)...(x-63)}{(x-2)(x-4)(x-8)...(x-64)}$
9. Вычислить $P = (1+\sin 0{,}1)(1+\sin 0{,}2)...(1+\sin 10)$
10. Даны натуральное число *n* и вещественное число *x*. Вычислить $S = \sin x + \sin x^2 + ...\sin x^n$
11. Дано натуральное число *n*. Вычислить $S = 1 \cdot 2 + 2 \cdot 3 \cdot 4 + ... + n \cdot (n+1)..2n$

12. Дано натуральное число n (n>2). Вычислить $P = (1-\frac{1}{2^2})(1-\frac{1}{3^2})...(1-\frac{1}{n})$

13. Дано натуральное число n. Вычислить $S = \frac{1}{3^3} + \frac{1}{5^2} + \frac{1}{7^2} + ... + \frac{1}{(2n+1)^2}$

14. Для данного вещественного числа x вычислить по схеме Горнера $y = x^{10} + 2x^9 + 3x^8 + ... + 10x + 11$

15. Дано натуральное число n. Вычислить $y = 1 \cdot 3 \cdot 5 \cdot ... \cdot (2n-1)$

16. Дано натуральное число n. Вычислить $y = 2 \cdot 4 \cdot 6 \cdot ... \cdot (2n)$

17. Даны натуральное число n и вещественное число x. Вычислить $y = \cos x + \cos x^2 + \cos x^3 + ... + \cos x^n$

18. Вычислить $y = \sin 1 + \sin 1{,}01 + \sin 1{,}02 + ... + \sin 2$

19. Даны натуральные числа n и k. Вычислить $\sqrt{k + \sqrt{2k + ... + \sqrt{k(n-1) + \sqrt{kn}}}}$

20. Дано натуральное число n. Вычислить $S = \frac{2}{1} + \frac{3}{2} + \frac{4}{3} + ... + \frac{n+1}{n}$

21. Дано натуральное число n. Найти сумму $S = n^2 + (n+1)^2 + ... + (2n)^2$

22. Найти сумму: $-1^2 + 2^2 - 3^2 + 4^2 + ... + 10^2$. Условную инструкцию не использовать.

23. Вычислить сумму: $2^2 + 2^3 + 2^4 + ... + 2^{10}$ без возведения в степень.

24. Вычислить сумму: $1 + \frac{1}{3} + \frac{1}{3^2} + ... + \frac{1}{3^{10}}$ без возведения в степень.

25. Вычислить сумму: $1 - \frac{1}{2} + \frac{1}{3} - ... + (-1)^{n+1}\frac{1}{n}$. Условную инструкцию не использовать.

Задание 2. Построить таблицу значений функции $y=f(x)$ на интервале $x \in [0.05;1.55]$ с шагом $\Delta x=0.05$. Найти минимальное и максимальное значение, приминаемое функцией на указанном интервале, вычислить среднее арифметическое значение для сформированной таблицы значений функции $f(x_i)$.

1. $y = 2x^5 - \frac{4}{x^3} + \frac{1}{x} + 3\sqrt{x}$;
2. $y = \frac{3}{x} + \sqrt[5]{x^2} - 4x^3 + \frac{2}{x^4}$;
3. $y = 3x^4 + \sqrt[3]{x^5} - \frac{2}{x} - \frac{4}{x^2}$;
4. $y = 7\sqrt{x} - \frac{2}{x^5} - 3x^3 + \frac{4}{x^2}$;
5. $y = 7x + \frac{5}{x^2} - \sqrt[7]{x^4} + \frac{6}{x}$;
6. $y = 5x^2 - \sqrt[3]{x^4} + \frac{4}{x^3} - \frac{5}{x}$;

7. $y = 3x^5 - \frac{3}{x} - \sqrt{x^3} + \frac{10}{x^5}$;

8. $y = \sqrt[3]{x^7} + \frac{3}{x} - 4x^6 + \frac{4}{x^5}$;

9. $y = 8x^2 + \sqrt[3]{x^4} - \frac{4}{x} - \frac{2}{x^3}$;

10. $y = 4x^6 + \frac{5}{x} - \sqrt[3]{x^7} - \frac{7}{x^4}$;

11. $y = 2\sqrt{x^3} - \frac{7}{x} + 3x^2 - \frac{2}{x^5}$;

12. $y = 4x^3 - \frac{3}{x} - \sqrt[5]{x^2} + \frac{6}{x^2}$;

13. $y = 5x^3 - \frac{8}{x^2} + 4\sqrt{x} + \frac{1}{x}$;

14. $y = \frac{9}{x^3} + \sqrt[3]{x^4} - \frac{2}{x} + 5x^4$;

15. $y = \frac{4}{x^5} - \frac{9}{x} + \sqrt[5]{x^2} - 7x^3$;

16. $y = \frac{8}{x^3} + \frac{3}{x} - 4\sqrt{x^3} + 2x^7$;

17. $y = 5x^2 + \frac{4}{x} - \sqrt[3]{x^7} - 2x^6$;

18. $y = 10x^2 + 3\sqrt{x^5} - \frac{4}{x} - \frac{5}{x^4}$;

19. $y = \sqrt{x^5} - \frac{3}{x} + \frac{4}{x^3} - 3x^3$;

20. $y = 9x^3 + \frac{5}{x} - \frac{7}{x^4} - 3x^3$;

21. $y = 3\sqrt{x} + \frac{4}{x^5} + \sqrt[3]{x^2} - \frac{7}{x}$;

22. $y = \sqrt{x^3} + \frac{2}{x} - \frac{4}{x^5} - 5x^3$;

23. $y = 7x^2 + \frac{3}{x} - \sqrt[5]{x^4} + \frac{8}{x^3}$;

24. $y = 8x^3 - \frac{4}{x} - \frac{7}{x^4} + \sqrt[7]{x^2}$;

25. $y = 8x - \frac{5}{x^4} + \frac{1}{x} - \sqrt[5]{x^4}$.

Вопросы для собеседования

1) Программная переменная: объявление, инициализация, область видимости, правила именования программных переменных.
2) Внутренняя структура целочисленных переменных в памяти программы, написанной на языке *C*. Место знакового регистра. Как получить размер программной переменной? От чего зависит размер программных переменных? Приведите примеры.
3) Определение выражения в программе на языке *C*. Что влияет на порядок выполнения инструкций, входящих в состав выражения?
4) Что такое оператор управления? Какие операторы управления определены в языке *C* как они используются?
5) Какие операции определены в *C*? В чем заключается отличие логических и побитовых операций? Приоритет и ассоциативность выполнения операций в выражениях на языке Си, на что они влияют? Приведите примеры.
6) Область видимости программной переменной. Каким образом область видимости связана с «временем жизни» программной переменной.
7) Неявное приведение типа в языке *C*. Правило, на основании которого определяется тип результата выражения.
8) Что такое "выражение" и "оператор", как эти понятия определяются в языке *C*.

9) Как определяется тип переменной и тип выражения? Операции явного и неявного приведения типа.
10) Структура простого Си-приложения, исполняемого под управлением ОС *Linux*: какие сегменты выделяются программе и для чего они используются.

2.3. Лабораторная работа № 3 Векторная память. Использование статических массивов в языке Си

“Ах, как время-то летит. Брысь. Вот ты уже и до мелких пакостей дорос.”

(из м/ф «Шиворот-навыворот», СССР, ТО «Экран», 1981г.)

В языке ANSI C массив представляет собой совокупность данных одного типа, собранных под одним именем, располагающихся в одном сплошном блоке памяти. Доступ к элементам массива осуществляется по индексу, определяющему смещение до требуемого элемента от начала массива. Данное смещение определяется в элементах. Имя массива является указателем, содержащим адрес первого элемента массива, то есть элемента с нулевым значением индекса. Язык *C* поддерживает статические массивы как базовый тип данных. Индексная форма доступа к элементу массива транслируется в адресное выражение. Данный механизм используется для организации динамических массивов. Динамические массивы «похожи» на статические по способу доступа к элементам и по организации адресного пространства за исключением того, что память для динамических массивов выделяется из «кучи» (Heap). Размер статических массивов остается неизменным, он задается до этапа компиляции программы. Размер динамического массива может задаваться во время работы программы. Сервисы операционной системы, связанные с поддержкой службы памяти, предоставляют прикладной интерфейс для управления адресным пространством динамических массивов. В рамках данного раздела изучаются задачи обработки одномерных статических массивов.

Замечание. В языке ANSI C нет механизмов, контролирующих корректность значения индекса элемента массива. Контроль за доступом к элементам массива целиком и полностью возлагается на программиста.

Объявление массива:

```
[М_К_П] <Тип> <Список_массивов>;
где <Список_массиво> представляет собой запись вида
<Имя_массива>[P1][…[PN]] [={<Список_инициализаторов>}
```

Например:

```
const static int Keys[8] = {5, 7, 11, 13, 17, 19, 23, 29};
```

```
float Array_01[10][12];
char Msg1[6] = { ‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\n’ },
     Msg2[6] = { ‘W’, ‘o’, ‘r’, ‘l’, ‘d’, ‘\n’ };
int intArr_1[10], intArr_2[12];
```

Доступ к элементу массива может осуществляться:

– через индексное выражение:

<Идентификатор_массива>[*Индекс*1][…[*Индекс*N]]

– через указатель:

*(<Идентификатор_массива> + <Значение_приведённого_индекса>)

Например:

```
intArr_1[2]=15, intArr_1[3]=-27;
*(Array_01  +  5 * 12 + 7)=5.25F;
```

Пример 1. Определение массива и использование цикла для инициализации его элементов.

```
/* figure: array01                                                    */
/*********************************************************************/
/* Filename: progr01.c                                               */
/* Abstract: This is a sample C-program                              */
/* Description: Пример работы с массивами в языке C                  */
/* Create Date: 2015 / 10 / 14                                       */
/* Author:                                                           */
/* Notes / Platform / Copyright   UNIX/Linux, FreeWare               */
/*********************************************************************/
#include <stdio.h>           /* Подключение библиотеки <stdio.h> */
/* Работа программы начинается с вызова функции main()               */
int main(void)
  {  /* BEGIN: Начало кода главной функции модуля                    */
     int Arr[10]; /* Arr – массив из 10 целых значений */
     int i; /* счётчик */
     /* Инициализировать элементы массива Arr нулями */
     for(i=0; i<10; i++) { /* Цикл для i от 0 до 9 с шагом 1 */
         Arr[i]=0; /* Arr[i] := 0 */
     }  /* Конец цикла */
     /* Печать заголовка таблицы */
     printf(“%s%13s%7s\n”, “Element”, “Value”, “Address”);
     /* Вывод на печать содержимого массива в виде таблицы */
     for(i=0; i<10; i++) { /* Цикл для i от 0 до 9 с шагом 1 */
         printf(“%7d%13d%7d\n”, i, Arr[i]), &Arr[i];
     }  /* Конец цикла */
     printf(“Press <Enter>-Key to exit.\n”);
     getcahr();  /* Ожидание действия пользователя */
     return 0;  /* Возврат управления ОС в случае успешной работы /
  }  /* END: Конец кода главной функции модуля                       */
```

Пример 2. Инициализация массива в определении с помощью списка инициализаторов.

```
/* figure: array02                                                    */
#include <stdio.h>           /* Подключение библиотеки <stdio.h> */
/* Работа программы начинается с вызова функции main()              */
int main(void) { /* BEGIN: Начало кода главной функции модуля     */
    int Arr[10] = {10, 12, 15, 76, 47, 11, 23, 33, 54, 81};
    int i; /* счётчик */
    /* Печать заголовка таблицы */
    printf("%s%13s%7s\n", "Element", "Value", "Address");
    /* Вывод на печать содержимого массива в виде таблицы */
    for(i=0; i<10; i++) { /* Цикл для i от 0 до 9 с шагом 1 */
        printf("%7d%13d%7d\n", i, Arr[i]), &Arr[i];
    }  /* Конец цикла */
    printf("Press <Enter>-Key to exit.\n");
    getcahr(); /* Ожидание действия пользователя */
    return 0;  /* Возврат управления ОС в случае успешной работы */
}  /* END: Конец кода главной функции модуля                        */
```

Типичные ошибки программирования массивов

1. Не учитывается факт, что элементы массива в *C* автоматически не инициализируются нулевыми значениями.

`int Array[10] = {0};` - нулевым значением инициализируется только первый элемент массива

2. Некорректная инициализация элементов массива.

`int Array[5] = {0, 1, 2, 3, 4, 5};` - количество инициализаторов превышает количество элементов массива

3. Выход за пределы массива.

`int Array[10];` - объявлен массива из 10 целочисленных элементов

`Array[10] = 10;` - попытка обратиться к элементу за пределами массива

Пример 3. Инициализация массива с помощью псевдо-случайного значения, возвращенного в результате вызова функции `rand()`.

```
/* figure: array03                                                    */
/*********************************************************************/
/* Filename: array03.c                                               */
/* Abstract: This is a sample C-program                              */
/* Description: Заполнение целочисленного массива псевдо-            */
/* случайными значениями и вывод содержимого массива на экран        */
/* (в STDOUT)                                                        */
/* Create Date: 2015 / 10 / 14                                       */
/* Author:                                                           */
/* Notes / Platform / Copyright   UNIX/Linux, FreeWare               */
/*********************************************************************/
#include <stdlib.h>       /* Для функций rand() и srand()            */
```

```
#include <time.h>          /* Для функции time()                    */
#include <stdio.h>         /* Для функций printf() getchar()        */
#define SIZE 10            /* Определение символической константы */
/* Работа программы начинается с вызова функции main()             */
int main(void) {    /* BEGIN: Начало кода главной функции модуля */
    int Arr[SIZE];  /* Статический массив целочисленных значений */
    int i;           /* счётчик */
    srand((unsigned)time(NULL));  /* “Сброс” функции генератора  */
    /*  Инициализировать элементы массива случайными значениями  */
    for(i=0; i<10; i++) { /* Цикл для i от 0 до SIZE-1 с шагом 1 */
        Arr[i]=rand()%1000; /* Arr[i] := x| x ∈ [0, 999]         */
    }  /* Конец цикла */
    /* Печать заголовка таблицы */
    printf(“%s%13s%7s\n”, “Element”, “Value”, “Address”);
    /* Вывод на печать содержимого массива в виде таблицы       */
    /* Цикл для i от 0 до SIZE-1 с шагом 1 */
    for(i=0; i<SIZE; i++) {
        printf(“%7d%13d%7d\n”, i, Arr[i]), &Arr[i];
    }  /* Конец цикла */
    printf(“Press <Enter>-Key to exit.\n”);
    getcahr(); /* Ожидание действия пользователя */
    return 0;  /* Возврат управления ОС в случае успешной работы */
}  /* END: Конец кода главной функции модуля                     */
```

Типовые задачи, связанные с обработкой элементов массивов

1. Нахождение суммы всех элементов массива.

Дано: **Целое** N, **Массив** A[N], Сумма := 0
. . . // Подготовка данных – инициализация элементов массива
НЦ **Для** i **от** 0 **до** N-1
| Сумма := Сумма + A[i]
КЦ
Вывод результата или его использование в дальнейших расчетах

Программная реализация данного алгоритма на языке С:

```
const int N = 20; /* объявление константы – размера массива */
float Arr[N], Sum=0F;
int i;
for(i=0; i<N; i++) {
    Arr[i] = (float)(i * i + 1) / i;
}
 . . .
for(i=0; i<N; i++) {
  Sum+=Arr[i];
}
```

. . .

2. Нахождение суммы элементов массива с заданными свойствами

Дано: **Целое** N, **Массив** A[N]
Сумма := 0
. . . // Подготовка данных – инициализация элементов массива
НЦ Для i **от** 0 **до** N-1
| **Если** <Условие>
| | **То** Сумма := Сумма + A[i]
| **Все-если**
КЦ
Вывод результата или его использование в дальнейших расчетах

3. Нахождение количества элементов массива с заданными свойствами

4. Нахождение среднего арифметического элементов массива с заданными свойствами

5. Изменение значений элементов массива с заданными свойствами

6. Вывод на экран элементов массива с заданными свойствами

7. Нахождение номеров (значений индексов) элементов массива с заданными свойствами

8. Определение значения индекса элемента массива, равного заданному числу

Задачи для практической работы в классе

Задания для лабораторной работы (выполняемые по номеру варианта)

Задание. Сформировать двумерный статический массив значений вещественного типа, соответствующий таблице значений функции на заданном интервале $[a,b]$ с заданным шагом дискретизации Δx. По сформированным данным найти наибольше и наименьшее значение функции $y=f(x)$ на заданном интервале.

1. $y = \ln(x^2 - 2x + 2)$, [0; 3], Δx=0.1;
2. $y = 3x/(x^2 + 1)$, [0; 5], Δx=0.1;
3. $y = (2x - 1)/(x - 1)^2$, [-0.5; 0], Δx=0.1;

4. $y=(x+2)e^{1-x}$, [-2; 2]
5. $y=\ln(x^2-2x+4)$, [-1; 1.5], Δx=0.1;
6. $y=x^3/(x^2-x+1)$, [-1; 1], Δx=0.1;
7. $y=((x+1)/x)^3$, [1; 2] , Δx=0.1;
8. $y=\sqrt{x-x^3}$, [-2; 2] , Δx=0.1;
9. $y=4-e^{-x^2}$, [0; 1] , Δx=0.1;
10. $y=(x^3+4)/x^2$, [1; 2] , Δx=0.1;
11. $y=xe^x$, [-2; 0] , Δx=0.1;
12. $y=(x-2)e^x$, [-2; 1] , Δx=0.1;
13. $y=(x-1)e^{-x}$, [0; 3] , Δx=0.1;
14. $y=x/(9-x^2)$, [-2; 2] , Δx=0.1;
15. $y=(1+\ln x)/x$, [1/*e*; *e*] , Δx=1/(3*e*);
16. $y=e^{4x-x^2}$, [1; 3] , Δx=0.1;
17. $(x^5-8)/x^4$, [-3; -1], Δx=0.1;
18. $y=(e^{2x}+1)/e^x$, [-1; 2], Δx=0.1;
19. $y=x\ln x$, [1/e^2; 1], Δx=0.1;
20. $y=x^3e^{x+1}$, [-4; 0], Δx=0.1;
21. $y=x^2-2x+2/(x-1)$, [-1; 3], Δx=0.1;
22. $y=(x+1)\sqrt[3]{x^2}$, [-4/5; 3], Δx=0.1;
23. $y=e^{6x-x^2}$, [-3; 3], Δx=0.1;
24. $y=(\ln x)/x$, [1; 4], Δx=0.1;
25. $y=3x^4-16x^3+2$, [-3; 1], Δx=0.1;
26. $y=x^5-5x^4+5x^3+1$, [-1; 2], Δx=0.1;
27. $y=(3-x)/e^x$, [0; 5], Δx=0.1;
28. $y=\sqrt{3}/2+\cos x$, [0; π/2] , Δx=π/100;
29. $y=108x-x^4$, [-1; 4], Δx=0.1;
30. $y=x^4/4-6x^3+7$, [16; 20], Δx=0.1;

Определить необходимый для хранения данных размер статического двумерного массива. Использую операторы цикла решить задачу присвоения значений элементам двумерного массива согласно заданию. Обеспечить процесс отображения сформированного массива значений на экран, то есть в поле терминала. Использовать спецификаторы форматного вывода данных таким образом, чтобы обеспечить максимальную наглядность выводимой таблицы данных.

```
#include <stdio.h>
#define SIZE 4
void DrawLine(FILE*, int);
int main() {
    FILE* fPtr=NULL;
    int a[SIZE]={1, 2, 3, 4};
    int i, *p;
    p=a;
    fPtr=fopen("con", "w");
    DrawLine(fPtr, 63);
    fprintf(fPtr, " a = %p &a = %p *a = %d\n", a, &a, *a);
    DrawLine(fPtr, 63);
    if(fPtr==NULL){
        return 0;
    }
    fprintf(fPtr, "   i  p + i    a + i      *(p+i)  p[i]  i[p]
    *(a+i)  a[i]  i[a]\n");
    DrawLine(fPtr, 63);
```

```
    for(i=0; i<SIZE; i++) {
        fprintf(fPtr, "%4d%9p%9p%7d%7d%6d%8d%7d%6d\n",
            i, p+i, a+i, *(p+i), p[i], i[p], *(a+i), a[i], i[a]);
    }
    DrawLine(fPtr, 63);
    fclose(fPtr);
    fPtr=NULL;
    printf("Press any key to contineu\n");
    getchar();
    return 0;
}
void DrawLine(FILE* fp, int n) {
    int i;
    fputc(' ', fp);
    for(i=0; i<n; i++) {
        fputc('-', fp);
    }
    fputc('\n', fp);
}
```

Вопросы для собеседования

1. Массивы в языке Си: объявление, начальная инициализация, доступ к элементам массива. Индексное выражение.
2. Особенности организации одномерных и многомерных массивов в языке Си. Понятие приведенного индекса массива.
3. Определение алгоритма и его свойства.
4. Организация адресного пространства приложения. Влияние области видимости переменной на выбор сегмента для ее размещения в программе. Влияние области локализации программной переменной на присваиваемое ей значение по умолчанию.
5. Какие спецификации форматов ввода-вывода данных имеют функции *scanf*() и *printf*()? Каким образом организовывать ввод данных для поддержки программной обработка неправильно введенных данных?
6. Перенаправление потоков ввода-вывода. Как можно сформировать файл с исходными данными? Потоковые функции для работы с текстовыми файлами *fprintf*() и *fscanf*(), их использование для файлового ввода-вывода.
7. Как вывести результат работы программы в файл?

Задачи для самостоятельной работы

2.4 лабораторная работа N 4. Синтез и использование функций, указатели, динамическое управление памятью

Благими намерениями вымощена дорога в Ад.
(Самуэль Джонсон)

Функции

Функция – это совокупность объявлений и операторов, предназначенных для решения определенной задачи. С функцией в *C* связано три понятия: объявление, определение и вызов. Объявление функции декларирует компилятору, с какими параметрами будет вызываться функция, то есть, как нужно интерпретировать список фактических параметров. Определение «определяет» код, реализуемый функцией. В ряде случаев допускается совмещать объявление и определение функции. Вызов функции используется непосредственно в коде программы. Функции, как часть кода, участвуют в процессе внутри программного взаимодействия (рис.1). Особо стоит отметить тот факт, что код программы на языке *C* оформлен в виде вызова функции `main()` – главной функции программы, совмещённого с ее определением.

Внутри кода одной функции может содержаться и, как правило, содержится вызов других функций. В ряде случаев, это чрезмерно запутывает сложность кода, и программа может стать «похожей на тарелку со спагетти».

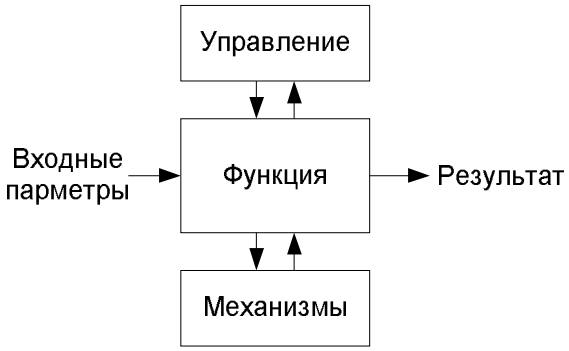


Рисунок 1. Обобщенная модель взаимодействия функции с программной средой

Можно сказать , что программа на языке *C* представляет собой совокупность объявлений программных переменных, операторов и вызовов функций. Код функций, то есть их определение, на уровне бинарного модуля хранится в сегменте команд (*CS – Code Segment*) непосредственно «после» команд тела функции *main*() (рис.2).

Рисунок 2. Обобщенная структура исполняемого модуля

В языке программирования *C* запрещаются вложенные объявления и определения функций. Язык программирования *C* не поддерживает вызова процедур, используемых в других языках программирования, таких, как *Pascal*. Возможна опосредованная реализация процедур средствами макроподстановок, но она может приводить к трудно выявляемым ошибкам времени исполнения.

Функции в языке *C* предоставляют поддержку структурной парадигмы программирования, так как позволяют выделять код отдельных задач непосредственно в виде кода функций. Но, как следует из изречения Самуэля Джонсон, сформулированном им еще в XVIII веке, мы не всегда правильно используем возможности функций и вместо улучшения в программу вносим скрытые дефекты, которые могут приводить к проявлению ошибок. Самая безобидная ошибка, возникающая при разработке функций, это создание функций, выполняющих более одной задачи. Кроме того, начинающие программисты пытаются определить в функции, реализующей не очень простую задачу, более одной «точки выхода», то есть использую более одного оператора `return`, что приводит к снижению технологичности ПО, то снижают надежность кода.

Формат объявления функции:

```
[М.1] <Тип> <М.2> <Имя>(<список_типов_форм._параметров>);
```

Формат определения функции:

```
[М.1] <Тип> <М.2> <Имя>(<список_форм._параметров>) {
    <Код_тела_функции>
}
```

Формат вызова функции:

<Операнд1>(*<Операнд2>*)

где *M.1* – модификатор класса памяти для типа возвращаемого из функции значения; *M.2* – модификатор стека, определяющий порядок выборки фактических параметров из стека, выполняемый при вызове функции; Тип – определяет тип возвращаемого из функции значения; *Операнд1* – это имя функции или указатель на функцию; *Операнд2* – список типов фактических параметров.

Типом возвращаемого значения для функции может быть любой тип, кроме массива и функции, но из функции можно вернуть указатель на массив, либо указатель на функцию. Это значит, что из функции можно вернуть только одну программную переменную, один «объект», но этот «объект» может быть переменной агрегативного типа: структурой либо объединением. Если функция в качестве результата работы должна возвращать более одного параметра, то эти параметры передаются ей в качестве параметров – указателей.

Следует обратить внимание на то, что объявление функции является выражением, поэтому оканчивается «пустым оператором» **‘;’**.

В теории программирования определены два механизма передачи параметров в функцию: по значению и по ссылке. В *C* поддерживается только механизм передачи по значению, иногда называемый механизмом передачи параметров по имени, так как в вызове фигурируют «имена» фактических параметров функции.

Функция предназначена для решения одной задачи, настоятельно рекомендуется не нарушать данное правило, так как это приводит к искусственному увеличению сложности кода и, как следствие, снижению технологичности программного обеспечения (ПО).

Например, функция для нахождения суммы двух целочисленных аргументов, может быть реализована следующим образом.

```
int SumIntVar(int, int);
```

Ее определение в программе:

```
int SumIntVar(int var1, int var2) {
     return (var1 + var2);
}
```

Вызов данной функции может быть выполнен следующим образом:

```
int a=12, b=97;
printf(“Sum is %d\n”, SumIntVar(a, b));
```

Кроме того, следует учитывать особенности механизма передачи параметров в функцию, она заключается в том, что в стек помещаются

копии фактических параметров и все манипуляции внутри вызова функции выполняются именно в отношении этих копий.

В ряде случаев это может привести к «ошибкам проектирования» кода, особенно на начальном этапе работы с языком *C*, которые проявляются в том, что функции «делают» не то, что от них ожидают.

Ниже приводится пример кода, демонстрирующий данный факт.

```
#include <stdio.h>
void IncrInt(int);
int main() {
     int k=10;
     print(“Before call IncrInt(): %d\n”, k);
     IncrInt(k);
     print(“After call IncrInt(): \n”, k);
     return 0;
}
void IncrInt(int val) {
     printf(“BEGIN: Call IncrInt()\n”);
     printf(“--> StartUp Parameter: %d\n”, val);
     val=val+1;
     printf(“--> The resulting value: %d\n”, val);
     printf(“END: Exit the function”);
     return;
}
```

Результат работы программы:

```
Before call IncrInt(): 10
BEGIN: Call IncrInt()
--> StartUp Parameter: 10
--> The resulting value: 11
END: Exit the function
After call IncrInt(): 10
```

То есть значение переменной после вызова функции не изменилось. В данном случае для того, чтобы вызов функции привел к изменению значения локальной переменной, в функцию необходимо передавать значение указателя.

```
#include <stdio.h>
void IncrInt(int*);
int main() {
     int k=10;
     print(“Before call IncrInt(): %d\n”, k);
     IncrInt(&k);
     print(“After call IncrInt(): \n”, k);
     return 0;
}
```

```
void IncrInt(int* val) {
     printf(“BEGIN: Call IncrInt()\n”);
     printf(“--> StartUp Parameter: %d\n”, *val);
     *val=*val+1;
     printf(“--> The resulting value: %d\n”, *val);
     printf(“END: Exit the function”);
     return;
}
```

Результат работы изменённой программы:

```
Before call IncrInt(): 10
BEGIN: Call IncrInt()
--> StartUp Parameter: 10
--> The resulting value: 11
END: Exit the function
After call IncrInt(): 11
```

Рассмотрим ещё несколько примеров реализации функций в языке С.

```
void swapInt(int* a, int* b) {
     int c = *a; *a = *b; *b = c;
}

double MaxReal(double a, double b) {
     return (a>b ? a: b);
}

double Cube(double val) {
     return (val*val*val);
}
```

Иногда программисты позволяют себе некоторые вольности по искусственному усложнению кода программы, но, как правило, это ничем не оправдано и приводит к снижению качества ПО, делая его не способным к дальнейшей поддержке и модификации.

```
#include <stdio.h>
void func1(void), func2(void);
int main(void) {
    func1();
    return 0;
}
void func1(void) {
     int i;
     for(i=0; i<10; i++)
         func2();
}
void func2(void) {
     int i;
     for(i=0; i<10; i++)
```

```
        printf(“%d”, i);
}
```

В данном примере функция *func1*() используется лишь как «обертка» для вызова функции *func2*(), фактически, никакой «полезной» нагрузки она не несет. Но, если детально разобраться в этом примере, то вызов функции оборачивается лишними машинными командами, загрузкой параметров в стек, передачей управления и, тем самым, существенно снижают производительность программы.

Иногда для повышения производительности в программах на *C* используются макроподстановки, выполненные наподобие процедур. Например, директива препроцессора

```
#define ABS((X)) ((X)>0 ? : (X) : (-(X)))
```

определяет макроподстановку, выполняющую роль функций *abs*() и *fabs*(), входящих в состав библиотеки <*math.h*>. Такие макроподстановки вызываются наподобие процедур, то есть их код непосредственно подставляется в точку вызова, но их использование в ряде случаев может приводить к искажению результата. Например,

```
#define CUBE((X)) ((X)*(X)*(X))
  .   .   .
int var1=10;
int var2=CUBE(var1++);
```

Данная макроподстановка разворачивается в следующее выражение:

```
int var2=(var1++)*(var1++)*(var1++);
```

что эквивалентно

```
int var2=(10)*(11)*(12);
```

Таким образом, значение переменной *var*2 не будет равным 1000. То есть, результат вызова макроса может отличаться от ожидаемого.

Массивы в качестве параметров функций

Массивы в качестве параметров могут быть переданы в функцию только через указатель. Для этого объявление функции должно содержать **указатель на массив** и информацию о его размерности, это может быть сделано следующим образом:

```
void func1(int A[n], int n);
void func2(int* A, int n);
```

Запись *int A*[*n*] в объявлении функции эквивалентна *int* A*. Важно, что одномерные массивы занимают сплошной блок памяти, в функцию

передается адрес первого элемента массива и количество элементов в массиве.

Следующий пример демонстрирует базовые приемы работы с одномерными массивами.

```
#include <stdio.h>
#define SIZE 10

void swap(int* a, int* b) {int c=*a; *a=*b; *b=c;}
void PrintArray(int* arr, int n);
void SortArray(int* arr, int n);

int main() {
    int Arr[SIZE]={123, 3, 15, 45, -2, 0, 234, 12, -1, 10};
    printf("Исходный массив : \n");
    PrintArray(Arr, SIZE);
    printf("Массив после обработки : \n");
    PrintArray(Arr, SIZE);
    return 0;
}

void PrintArray(int* arr, int n) {
     int i;
     for(i=0; i<n; i++)
         printf("%d", arr[i]);
     printf("\n");
}
void SortArray(int* arr, int n) {
     int i, j;
     for(i=0; i<n-1; i++) {
         for(j=i+1; j<n; j++) {
             if(arr[i]>arr[j])
                 swap(&arr[i], &arr[j]);
         }
     }
}
```

Передача многомерных массивов в качестве параметров функции требует несколько иного подхода, так как в функции необходима информация о внутренней организации массива: количество размерностей и значения размерностей – эта информация используется для вычисления адреса необходимого элемента массива. Рассмотрим следующие объявления:

```
void func1(int A[S1][S2], int S1, int S2);
void func1(int A[ ][S2], int S1, int S2);
void func2(int A[S1][S2][S3], int S1, int S2, int S3);
void func2(int A[ ][S2][S3], int S1, int S2, int S3);
```

Допускается опускать значение левой размерности массива, передавая значения всех размерностей в качестве параметров функции.

При проектировании функций, предназначенных для обработки многомерных массивов, начинающие программисты допускают ошибки относительно типа указателя на массив. Это связано с тем, что статические многомерные массивы в *C* реализуются через одномерные массивы, элементы которых являются массивами. Также нужно учесть, что статические и динамические многомерные массивы используют несколько различные механизмы адресации. То есть, параметр функции вида `int A[n][m]` ни коим образом не связан с параметром `int** A`.

Использование динамической памяти

В языке *C* для работы со свободной динамической памятью предназначены функции `malloc()`, `calloc()`, `realloc()` и `free()`. Они являются системными вызовами, реализующими обращение к менеджеру «кучи».

Прототипы функций:

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

Правила чтения сигнатур: `malloc` – «***m***emory ***alloc***ates memory», `calloc` – «***c***lear ***alloc***ates memory», `realloc` – «***re alloc***ates memory».

Функция `malloc()` возвращает указатель на блок памяти в «куче», то есть адрес его первой ячейки, в случае невозможности выделения памяти данная функция возвращает значение `NULL`. В качестве аргумента `size` данная функция принимает размер области, указанный байтах.

Функция `calloc()` возвращает указатель на начало блока памяти в «куче». Данный блок заполняется нулевыми значениями. Данная функция принимает два аргумента типа `size_t`, определенного в библиотеке `<string.h>` и ряде других библиотек. Предполагается, что один из аргументов `nmemb` определяет количество блоков, а второй `size` – размер каждого блока, внутри функции производится вычисление значения произведения данных аргументов. В случае невозможности выделить блок требуемого размера функция `calloc()` возвращает `NULL`-указатель.

Функция `realloc`() позволяет изменить размер ранее выделенного в «куче» блока памяти, причем механизм изменения заключается именно в выделении нового блока памяти. В случае успешного выделения нового

блока памяти большего размера, то производится процедура копирования данных из «старого» блока памяти в «новый».

Функции для работы с памятью

Базовые задачи, связанные с обработкой областей памяти, реализованы функциями семейства `mem`, входящими в состав стандартной библиотеки языка *C* и объявленными в <`string.h`>. Вот некоторые из них:

memset(), memcpy(), memmove(), memchr().

Сигнатуры унций:

```
void* memcpy(void* destptr, const void* srcptr, size_t num);
void* memmove(void* destptr, const void* srcptr, size_t num);
const void* memchr(const void* memptr, int val, size_t num);
void* memchr(void* memptr, int var, size_t num);
void* memset(void* memptr, int var, size_t num);
```

Для выполнения данной лабораторной работы требуется решить одну из предложенных ниже задач, учитывая следующие дополнительные требования:

1) все математические функции, указанные в условиях задач, необходимо синтезировать самостоятельно без привлечения библиотечных модулей;
2) массивы, используемые в программе, должны быть динамическими;
3) программа должна быть работоспособна для любых входных матриц, общий размер которых $N \times M \leq 300$.

Программа должна работать в текстовом режиме, обеспечивать процедуры запроса и ввода данных, сопровождающиеся соответствующими пояснениями. В случае, когда в программе выполняются действия над массивами данных, программный диалог должен начинаться с запроса на ввод размерностей соответствующих массивов с проверкой на корректность введенных данных. Далее должен быть реализован запрос на выбор способа подготовки данных: ручной ввод, либо "автоматическая" программная генерация с использованием аналога функции *`rand()`*, реализованного по алгоритму линейного конгруэнтного генератора (ЛКГ), с последующей "адаптацией" сгенерированных данных к требованиям программы. При генерации вещественных значений ограничиться тремя цифрами после плавающей точки.

Замечание. В заданиях, касающихся действий над матрицами, в качестве представления матриц использовать динамические массивы соответствующих размерностей.

Задания.

1. Ввести массив натуральных чисел $A[N]$ размера $N \le 40$, вывести его на экран монитора. Написать функцию, преобразующую натуральное число в число, равное сумме цифр, представляющих исходное число в M-ичной системе счисления. Вывести на консоль в десятичной системе счисления исходный массив, предварительно заменив в нем четные числа на сумму цифр их четверичного, а нечетные - пятеричного представления.

2. Вычислить и напечатать таблицу значений функции

$$f(x) = \begin{cases} \cos(x) + 1 & при \quad \operatorname{tg}(x) < 1, \\ \sqrt{|\sin(x)|} & при \quad 2 > \operatorname{tg}(x) \ge 1, \\ \sin(x) + \cos(x) & при \quad \operatorname{tg}(x) \ge 2 \end{cases}$$

и ее максимальное значение при $x \in [a,b]$ с шагом h. Вычисление $f(x)$ для одного значения аргумента определить в отдельной функции.

3. Ввести массив натуральных чисел $a[N]$ размера $N \le 40$ (все числа $a[i] < 10000$), вывести его на экран монитора. Написать функцию, преобразующую натуральное число в число, равное произведению цифр, представляющих исходное число в M-ичной системе счисления ($1 < M \le 9$). Вывести на консоль в десятичной системе счисления исходный массив, предварительно заменив в нем числа, все цифры M-ичного представления которых отличны от нуля, на произведение цифр их четверичного представления, а остальные оставив без изменений.

4. Вычислить и напечатать таблицу значений функции

$$f(x) = \begin{cases} \max\{x, tg(x)) + \sqrt{|x|} & при \quad |x - 0{,}5| < 1, \\ e^x \cdot \cos(x) & при \quad |x - 0{,}5| \ge 1 \end{cases}$$

и ее минимальное значение при $x \in [a,b)$ с шагом h. Вычисление $f(x)$ для одного значения аргумента определить в отдельной функции.

5. Вычислить и напечатать таблицу значений функции

$$f(x,y) = \begin{cases} \sin[x \cdot (|x + y| + 1)] & при \quad |x - y| \le 1, \\ \ln|x + e^y| & при \quad 1 < |x - y| \le 2, \\ 4\sin x + \cos x & при \quad |x - y| > 2 \end{cases}$$

и таблицу тех ее значений, которые по модулю не превышают 2, при $x \in [a,b]$ с шагом h_x, $y \in [c,d]$ с шагом h_y. Вычисление $f(x,y)$ для одной пары значений аргументов определить в отдельной функции.

6. Вычислить и напечатать таблицу значений функции

$$f(x,y)=\begin{cases}(1+|y|)^{2x}+x & при \quad x<y,\\ \sqrt[3]{y^2}\,\mathrm{arctg}(x+y) & при \quad x=y,\\ \mathrm{tg}(x+y) & при \quad x>y\end{cases}$$

и сумму ее значений для всех пар аргументов при $x\in[a,b)$ с шагом h_x, $y\in[c,d)$ с шагом h_y. Вычисление $f(x,y)$ для одной пары значений аргументов определить в отдельной функции.

7. Вычислить и напечатать таблицу значений функции

$$f(x)=\begin{cases}\mathrm{arctg}(x) & при \quad x<0,\\ \cos(x) & при \quad 0\le x<1,\\ \min\{\mathrm{arctg}(x),\cos(x)\} & при \quad x\ge 1\end{cases}$$

и определить, является ли она монотонной на интервале $[a, b]$ при дискретном увеличении аргумента от $x=a$ с шагом h. Вычисление $f(x)$ для одного значения аргумента определить в отдельной функции.

8. Вычислить и напечатать таблицу значений функции

$$f(x)=\begin{cases}x^2+\sqrt{|x|} & при \quad x^2<1,\\ x^2\,\mathrm{arctg}(2x+1) & при \quad 1\le x^2\le 2,\\ \sin(\cos(|x|)) & при \quad x^2>2\end{cases}$$

и ее максимальное по модулю значение для $x\in[a, b)$ при дискретном увеличении x с шагом h. Вычисление $f(x)$ для одного значения аргумента определить в отдельной функции.

9. Для матриц a и b размерности $N\times M$ и $M\times L$ соответственно вычислить и напечатать $\{|a|_{\min}, I_a, J_a\}$, $\{|b|_{\min}, I_b, J_b\}$ - значения и индексы минимальных по модулю элементов матриц a и b, а также значения переменных $K=(I_a+I_b)-(J_a+J_b)$, $\beta=\max\{|a|_{\min},|b|_{\min}\}$. Вычисление значения и индексов минимального по модулю элемента матрицы определить в отдельной функции.

10. Заданы матрицы a и b с размерами $N\times M$ и $M\times L$ соответственно. Вычислить и напечатать $\{|a|_{\min}, I_a, J_a\}$, $\{|b|_{\min}, I_b, J_b\}$ - значения и индексы минимальных по модулю элементов матриц a и b, а также значение $\beta=\min\{I_a-I_b,\ J_a-J_b,\ \max\{|a|_{\min},\ |b|_{\min}\}\}$. Вычисление значения и индексов минимального по модулю элемента матрицы определить в отдельной функции.

11. Заданы матрицы A, B и C с размерами $N\times M$, $M\times L$ и $L\times K$ соответственно. Вычислить и напечатать значения их минимальных элементов $a_{\min}$, $b_{\min}$, $c_{\min}$ и переменной $\beta=\max\{|a_{\min}|,\ |b_{\min}|,\ |c_{\min}|\}$. Вычисление минимального элемента матрицы оформить в виде функции.

12. Заданы матрицы A, B и C с размерами $N\times M$, $M\times L$ и $L\times K$ соответственно. Вычислить и напечатать значения их максимальных

элементов $a_{\max}$, $b_{\max}$, $c_{\max}$, и переменной $\beta=\min\{|a_{\max}|, |b_{\max}|, |c_{\max}|\}$. Вычисление минимального элемента матрицы оформить в виде функции.

13. Заданы матрицы A, B и C с размерами $N\times M$, $M\times L$ и $L\times K$ соответственно. Вычислить и напечатать значения переменных

$$S_a=\sum_{i=1}^{N}\left(\prod_{j=1}^{M}a[i][j]\right),\quad S_b=\sum_{i=1}^{M}\left(\prod_{j=1}^{L}b[i][j]\right),\quad S_c=\sum_{i=1}^{L}\left(\prod_{j=1}^{K}c[i][j]\right)$$

Алгоритм вычисления правых частей этих выражений оформить в виде функции.

14. Заданы матрицы A, B и C с размерами $N\times M$, $M\times L$ и $L\times K$ соответственно. Вычислить и напечатать значения переменных

$$P_a=\prod_{i=1}^{N}\left(\sum_{j=1}^{M}A[i][j]\right),\quad P_b=\prod_{i=1}^{M}\left(\sum_{j=1}^{L}B[i][j]\right),\quad P_c=\prod_{i=1}^{L}\left(\sum_{j=1}^{K}C[i][j]\right).$$

Процедуры вычисления правых частей этих выражений оформить в виде функций.

15. Заданы матрицы B и C с размерами $N\times M$ и $M\times M$ соответственно. Написать функции умножения и сложения двух матриц. Вычислить и напечатать матрицу $R=B\times C+B$.

16. Для матрицы A, с размерами $N\times M$ написать функции умножения матрицы на скаляр и вычисления суммы элементов матрицы. Вычислить и напечатать значения *Sa* - суммы элементов матрицы A, и $B=A/Sa$.

17. Заданы матрицы A и B с размерами $N\times M$ и $M\times L$. Написать функции умножения двух матриц и транспонирования матрицы. Вычислить и напечатать матрицу $R=(A\times B)^{\mathrm{T}}$.

18. Для матрицы A, с размерами $N\times M$, и M-мерного вектора $D=\{d_1, d_2, \ldots, d_{\mathrm{M}}\}$ написать функцию формирования $N\times M$-мерной матрицы Q, элементы которой вычисляются по формуле $Q[i][j]=A[i][j]\times \mathrm{D}[j]$. Написать функцию сложения двух матриц. Вычислить и напечатать матрицу $R=A+Q$.

19. Заданы матрицы A, B и C с размерами $N\times M$, $M\times L$ и $L\times K$ соответственно. Написать функции умножения двух матриц и вычисления следа квадратной матрицы. Вычислить и напечатать матрицу $R=A\times B\times C$ и ее след.

20. Заданы матрицы A и C с размерами $N\times M$ и $L\times L$ соответственно. Написать функции умножения матрицы на скаляр и вычисления следа квадратной матрицы. Вычислить и напечатать *Slc* - след матрицы C и матрицу $R=A\times Slc$.

21. Заданы матрицы A, B и C с размерами $N\times M$, $M\times K$ и $K\times N$ соответственно. Написать функции умножения матрицы на скаляр и вычисления минимального значения элемента матрицы. Вычислить и напечатать $A_{\min}$ и $B_{\min}$ – минимальные элементы матриц A и B, а также матрицу $R=C\times|A_{\min} - B_{\min}|$.

22. Написать функцию поиска максимального значения произвольной стандартной функции одного аргумента при дискретном изменении аргумента на интервале $[A,B]$ с шагом h (имя конкретной стандартной функции должно передаваться в созданную функцию как параметр). Определить и вывести соответствующее сообщение, какая из функций: $f_1(x)=\mathrm{arctg}(x)$, $f_2(x)=\cos(x)$, $f_3(x)=\sin(x)$, $f_4(x)=(e^{x}-e^{-x})/2$ имеет наибольшее значение на интервале $x\in[A,B]$ с шагом h.

23. Написать функцию расстановки элементов одномерного массива в порядке убывания их значений. Используя эту функцию, преобразовать исходную $N\times M$ - мерную матрицу A в матрицу B, каждая строка которой состоит из элементов соответствующей строки исходной матрицы, упорядоченных по убыванию их значений.

Методические указания

В условиях задач выражение вида "$x\in[A,B]$ с шагом h" означает, что переменная x изменяется дискретно с шагом h в интервале $[A,B]$, начиная от его левой границы;
$\max\{\alpha, \beta, \ldots, \gamma\}$ - функция, принимающая значение максимальной из перечисленных в скобках переменных.

Следует помнить, что элементы любого массива, в том числе и многомерного, обычно располагаются в оперативном запоминающем устройстве ЭВМ в одном непрерывном блоке ячеек памяти, размер которого вычисляется в ходе выполнения программы из конкретных значений размерностей заданного массива.

Таким образом, любой массив в памяти всегда хранится в виде одномерной последовательности значений, и обеспечение произвольного доступа к его элементам является обязанностью программиста. По соглашению, принятому для Си-программ, элементы многомерных массивов заносятся в память последовательно, начиная с элемента, имеющего нулевые индексы в порядке увеличения на единицу сначала младшего (первого справа) индекса от нуля до его максимального значения при нулевых значениях других индексов, затем элементы, второй индекс справа которых равен единице, а более старшие - нулевые и т.д. Для матрицы это соответствует построчной записи ее элементов в память ЭВМ. Произвольный доступ к элементу трехмерного массива $a[i][j][k]$, $i=0...I-1$, $j=0...J-1$, $k=0...K-1$ возможен с использованием

"приведенного индекса" - величины, равной смещению ячейки памяти, в которой хранится значение данного элемента *a*[*i*][*j*][*k*] относительно начала массива. Для *a*[*i*][*j*][*k*] приведенный индекс равен: *i*×*J*×*K*+*j*×*K*+*k*. Приведенная ниже программа иллюстрирует ввод целочисленных значений матрицы размера *line*×*col* в динамически запрашиваемую область памяти и вывод на монитор ее значений строчками по *col* элементов в каждой, а также отдельного элемента *matr*[*l*][*c*] с использованием приведенного индекса:

```
#include<stdio.h>    /*****************************************/
#include<conio.h>    /*  Данная программа разработана с учетом  */
#include<alloc.h>    /*  особенностей ОС  MS-DOS и MS Windows.  */
#include<process.h>  /*****************************************/
int main(void) {
    int *matr, *matr_ptr, line, col, l=1, c=2;
    printf("Введите кол-во <строк> <столбцов> матрицы:");
    scanf("%d %d",&line,&col);
    int size=line*col;
    char *diagn_mes="Ошибка распределения памяти.\n";
    if((matr=(int *)malloc(size*sizeof(int)))==NULL) {
        printf("%s", diagn_mes); getch(); abort();
    }
    matr_ptr=matr;
    for(int i=0;i<size;i++) {
        printf("Введите элемент matr[%d][%d]: ",i/col, i%col);
        scanf("%d",matr_ptr++);
    }
    printf("Исходная матрица:");
    for(i=0, matr_ptr=matr; i<size; i++) {
        if(!(i%col))
            printf("\n");
        printf(" %5d",*matr_ptr++);
    }
    printf("\nЭлемент matr[l=%d][c=%d]:%d", l, c, matr[l*col+c]);
    return 0;
}
```

Для того, чтобы эта программа считала данные из файла, а не с консоли, при ее запуске нужно перенаправить поток ввода на файл исходных данных (*source.dat*), используя оператор переназначения стандартного устройства ввода *DOS* `<` (пример запуска: *D:\>progr.exe* < *source.dat*). При этом файл исходных данных должен состоять из разделенных пробельными символами целых чисел, первое из которых - количество строк исходной матрицы, второе - количество столбцов, и

далее - построчно элементы матрицы, начиная с нулевого элемента нулевой строки. Например, для (2×3) матрицы

$$\begin{pmatrix} 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

файл исходных данных должен состоять из последовательности чисел:

2 3
4 5 6 7 8 9 .

Вопросы для собеседования

1. Что такое указатель и какие операции над указателями определены? Каким образом производится объявление, инициализация указателя? Правила адресной арифметики.
2. Каким образом определяется тип переменной - указателя? По каким правилам выполняются арифметические операции с переменными-указателями?
3. Что такое индексное выражение, приведенный индекс, для чего они нужны и как используются?
4. Какие операции с распределением памяти ЭВМ и как можно выполнить в процессе исполнения программы (динамически)?
5. Что такое функция в Си-программе? Что такое прототип и определение функции?
6. Что такое формальные и фактические аргументы функции?
7. Какие фактические параметры при вызове функции могут соответствовать формальному аргументу, являющемуся: а) идентификатором статического массива; б) указателем на функцию; в) идентификатором переменной одного из базовых типов?
8. Можно ли разные формальные аргументы функции обозначать одинаковыми идентификаторами? А фактические?
9. Перечислите все возможные способы передачи информации из программы в вызываемую функцию и обратно.

2.5. Лабораторная работа N 5. Строки и операции над ними. Использование функций обработки текстовых данных

Вначале было слово, и было оно длиной семь бит… - ASCII7
(из истории развития вычислительной техники)

Представление строк в *C*

Определение. Строка представляет собой массив элементов типа чар, содержащий нуль байт, - символ конца строки.

В языке *C*, в отличие от большинства высокоуровневых языков программирования, нет специального типа для представления строк. Вместо этого было предложено использовать символьный тип, так как отдельные символы являются отдельными элементами строк.

Рассмотрим следующие объявления символьных массивов в языке *C*.

```
char S1[20];
char S2[20] = {‘A’, ‘B’, ‘C’, ‘D’};
char S3[20] = {‘A’, 0};
char S4[20] = “Hello, World”;
char *S5= “Hello, World”;
```

*S*1 – просто объявление символьного массива; является ли он строкой, или нет, будет зависеть от того, где и как он объявлен. Если массив объявляется как глобальный, то по умолчанию он заполняется нулями, то есть ему будет соответствовать строка нулевой длины, в случае, если он будет объявлен локально, то его содержимому будет соответствовать «информационный мусор» и со строкой на момент создания он никак не связан.

*S*2 – это тоже не строка, а символьный массив.

*S*3 – массив, содержащий один ненулевой элемент.

*S*4 – массив, инициализация которого выполняется с помощью объекта типа const char*.

*S*5 – указатель на символьный тип, не строка, хотя и связан с «константной строкой».

Дли строки – это длина блока символов в символьном массиве, не содержащего нуль-байт, причем данный блок располагается непосредственно в начале массива. Массив нулевой длины содержит только нулевой элемент, «токен» или «терминатор», так ещё называют символ конца строки.

`char Str[100] = {0};` - строка нулевой длины.

Обратите внимание, что длина строки может быть меньше размера символьного массива, в котором она хранится. Этот факт и то, что в языке *C* нет механизма контроля корректности значения индекса массива, часто приводят к ошибкам сегментации. Именно поэтому работа со строками, равно как и с массивами, требует от программиста особой аккуратности.

Рассмотрим следующие этапы «эволюции» символьного массива.

```
char str[20] = “_Hello, World!”;
```

str – это строка, длина которой равна значению 13.

```
str[0]='\0'; /* заменим содержимое первого символа строки */
```

Теперь *str* связан с «пустой» строкой, её длина – ноль.

```
str[0]='_';
str[14] = 32; /* 32 – ASCII-код пробельного символа */
```

Теперь мы уже не можем гарантировать, что *str* будет являться *C*-строкой, так как нам не известно содержимое ячейки *str*[15].

Функции для работы со строками

В состав языка *C* входит библиотека <*string.h*>, в которой определены функции для работы со строками, дополнительные методы, отвечающие за обработку отдельных символов, определены в библиотечном файле <*ctype.h*>.

```
#include <string.h>

size_t strlen(const char *s);

char *strcat(char *restrict s1, const char *restrict s2);
char *strncat(char *restrict s1, const char *restrict s2,
                  size_t n);

char *strchr(const char *s, int c);
char *strrchr(const char *s, int c);

int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t n);


char *strcpy(char *restrict s1, const char *restrict s2);
char *strncpy(char *restrict s1, const char *restrict s2,
             size_t n);

char *strstr(const char *s1, const char *s2);
size_t strspn(const char *s1, const char *s2);
size_t strcspn(const char *s1, const char *s2);

char *strerror(int errnum);

char *strpbrk(const char *s1, const char *s2);

char *strsep(char **stringp, const char *delim);

char *strtok(char *restrict s1, const char *restrict s2);
```

В составе платформы *BSD* существует дополнительная библиотека для работы с *C*-строками <*strings.h*>.

```
#include <strings.h>

char *index(const char *s, int c);

char *rindex(const char *s, int c);

int strcasecmp(const char *s1, const char *s2);

int strncasecmp(const char *s1, const char *s2, size_t n);
```

Детально разберем функции для работы со строками.

Базовая модель работы с С-строками реализована в функции *strlen*() (“*string length*” – длина строки). Она возвращает длину строки в байтах, т есть количество символов в строке без нуль-байта.

```
size_t strlen(const char *s) {
    size_t i=0U;
    while(s[i] != ‘\0’) i++;
    return i;
}
```

Существенным недостатком данной функции является то, что она не может проверить, а действительно ли переданный ей символьный массив является строкой, то есть содержит нуль-байт.

Функция *stpcpy*() (“string copy” – копирование строки) копирует содержимое строки, определенную указателем *src*, в строку, адрес которой определен указателем *dst*. Данная функция, как и большинство библиотечных функций, не выполняет проверку структуры обрабатываемой строки, то есть, потенциально не безопасна.

Функция *strcpy*() реализует приблизительно следующий код:

```
char * strcpy(char *dest, const char *src) {
     size_t i=0;
     while(src[i] != 0) {
         dest[i] = src[i]; i++;
     }
     return dest;
}
```

Замечание. Практически все функции семейства *str* предполагают, что для обработки им передаются корректные данные, размер приемника данных не менее размера источника данных. Все это связано с тем, что в языке *C* нет механизмов передачи информации о размере массива, такие средства появились уже в *C*++ вместе с классами и контейнерами.

Именно поэтому в библиотеке *<string.h>* имеются функции, для которых задается ограничение на количество копируемых данных. В

частности, функция *strncpy*() выполняет копирование содержимого строки *S*2 в строку *S*1, но копирует не более *n* байт.

```
char *strncpy(char *restrict s1, const char *restrict s2, size_t n)
{
     size_t i=0;
     while(s2[i] != '\0') {
         if(i == n) /* если достигнут лимит копирования */
             break; /* прервать копирование            */
         s1[i] = s2[i];
         i++;
     }
     s1[i]='\0';
     return s1;
}
```

Функции *strcat*() и *strncat*() ("*string catenation*" – сцепление строк) «объединяют» строки, добавляют содержимое строки, адресуемой аргументом *S*2 в конец строки, адресуемой аргументом *S*2. Запись строки содержимого *S*2 в *S*1 начинается с позиции символа конца строки в *S*1. Символьный массив, связанный со строкой S1, должен быть достаточного размера, чтобы вмещать результирующую строку. Функция *strncat*() копирует в строку *S*1 содержимое строки *S*2, при этом количество копируемых символов не превышает значения параметра *n*, третьего аргумента функции.

```
char *strcat(char *restrict s1, const char *restrict s2) {
    size_t i=0U, j=0U;
    while(s1[i++] != '\0'); /* смещаемся в конец строки s1 */
    while(s2[j] != '\0') {
        s1[i]=s2[j]; i++; j++;
    }
    s1[i]='\0';
    return s1;
}
```

Функции *srtcmp*() и *strncmp*() ("*string comparison*" – сравнить строки) выполняют лексикографическое сравнение содержимого двух строк, адресуемого значениями указателей *S*1 и *S*2. Функция *strncmp*() в качестве третьего параметра принимают максимальную величину области, подлежащей процедуре сравнения.

```
int strcmp(const char *s1, const char *s2) {
     int res=0;
     size_t len1=strlen(s1), len2=strlen(s2);
     size_t len = (len1 > len2 ? len1 : len2);
     size_t i;
```

```
     for(i = 0U; i < len; i++) {
         res = s1[i] – s2[i];
         if(res != 0)
             break;
     }
     return res;
}
int strncmp(const char *s1, const char *s2, size_t n) {
     int res=0;
     size_t len1=strlen(s1), len2=strlen(s2);
     size_t len = (len1 > len2 ? len1 : len2);
     size_t i;
     for(i = 0U; i < len && i < n; i++) {
         res = s1[i] – s2[i];
         if(res != 0)
             break;
     }
     return res;
}
```

Функции *strchr*() и *strnchr*() выполняю поиск символа, заданного параметром *c* в строке, заданной указателем *S*. Функция *strchr*() осуществляет поиск с начала, а *strrchr*() с конца символьного массива. Они возвращают адрес найденного символа, либо значение *NULL*, если такого символа в строке нет.

```
char *strchr(const char *s, int c) {
    char *res = NULL;
    size_t i = 0U;
    while(s[i] != '\0') {
        if(s[i] == c) { res = (s + i); break; }
    }
    return res;
}
char *strrchr(const char *s, int c) {
    char *res = NULL;
    size_t i, len = strlen(s);
    if(i != 0) {
        for(i= len -1; i >= 0; i--) {
            if(s[i] == c) { res = (s + i); break;}
        }
    }
    return res;
}
```

Функция *strtok*() в строке, заданной параметром S1 выполняет поиск символов, заданных набором символов разделителей в строке S2. В случае, как только будет найдено первое совпадение символа строки и символа из набора токенов, в строку вставляется символ конца строки и возвращается адрес следующего за ним символа. Если в обрабатываемой строке нет ни одного символа, заданного массивом S2, то возвращается значение *NULL*.

```
char *strtok(char *restrict s1, const char *restrict s2) {
    char *res = NULL;
    size_t i, j, len1 = strlen(s1), len2 = strlen(s2);
    for(i=0U; i< len1; i++) {
        for(j=0U; j<len2; j++) {
            if(s1[i] == s2[j]) {
                s1[i] = '\0'; res = (s1 + i+ 1); break;
            }
        }
    }
    return res;
}
```

Задания. Источник данных и результат работы программы – текстовый файл. Для кодирования текса используется кодовая таблица *DOS*. Составить программу, обрабатывающую файл согласно заданному алгоритму.

1. Для двух заданных различных символов, значения кодов которых хранятся в переменных типа char *X* и *Y* ,задаваемых пользователем, определить, сколько раз в тексте встречается буква, определенная значением кода, хранящимся в *X*. Везде, в обрабатываемом тексте, заменить эту букву буквой, определенной значением кода, хранящимся в переменной *Y*.

2. В заданном тексте везде заменить слово *A*1 на слово *A*2 (длины слов в общем случае не совпадают).

3. Составить триады из символов текста, расположенных в нечетных позициях. Обрабатываются только отображаемые символы.

4. По заданному тексту сформировать список слов, состоящих только из букв русского либо латинского алфавита. Для русских букв используется кодировка Кириллица-*DOS*. Упорядочить полученный список слов по алфавиту. Количество имен в списке заранее не известно.

5. Удалить текст, содержащийся в обрабатываемом файле, начиная с первой встретившейся буквы '*M*' до третьей по счету буквы '*M*'. Вывести текст после преобразования и номера позиций начала и конца удаленной

части в исходной строке. В обрабатываемом файле содержится текст на русском языке в кодировке Кириллица-*DOS*.

6. Удалить из текста заданное слово. Вывести обработанный текст и номера позиций начала удаленного слова в исходном тексте.

7. Вставить слово в текст между двух заданных. Вывести обработанный текст и номера позиций начала и конца вставленного слова в новом тексте.

8. Для текста на русском языке определить количество гласных и согласных букв.

9. Поменять местами фрагменты текста ограниченные позициями *M*1, *M*2 и *N*1, *N*2. На этапе проектирования программы следует учесть допустимость ситуации, когда разности *M*2-*M*1 и *N*2-*N*1 могут не совпадать.

10. Вывести из текста на консоль слова, начинающиеся и заканчивающиеся на одну и ту же букву.

11. Вывести из текста на консоль слова, у которых в *i*-й позиции располагается одна и та же буква. В исходном тексте поменять местами первую пару таких слов и вывести результат на консоль.

12. Упорядочить слова текста по алфавиту относительно символов, заключенных между 3-й и 6-й позициями каждого слова. Вывести на консоль полученный текст и колонку из соответствующих словам пар символов – для облегчения проверки правильности сортировки.

13. Переформатировать заданный текст в новые строки, считая признаком конца строки в исходном тексте символ "\%".

14. В заданном тексте определить частоту, с которой в нем встречаются различные буквы и построить гистограмму распределения частот. В начале каждой строки гистограммы вывести символ и соответствующую ему частоту в процентах.

15. Разбить исходный текст на строки длиной не более 50 символов. Перенос осуществлять на местах расположения пробельных символов (не разделяя слова на части).

16. В тексте, содержащем менее 50 непробельных символов, равномерно расставить пробелы между словами так, чтобы его длина равнялась 50 символам.

17. Проверить общую сбалансированность разнотипных скобок в тексте. Скобки считаются сбалансированными, если каждой закрывающей скобке данного типа предшествует открывающая скобка этого же типа и для каждого типа скобок количество открывающих и закрывающих скобок равны.

18. В заданном тексте удалить заключенную в круглые скобки часть (вместе со скобками) и вывести результат на консоль.

19. Определить количество слов в тексте и поменять местами первое и пятое слова.

20. Указать минимальное количество первых букв, по которым можно различить слова из данного набора.

21. Текст задан следующим образом: первые символы – десятичное целое, задающее длину первого слова. Без пробелов, после первого слова - цифры, задающие длину второго слова и т.д. Все слова начинаются не с цифр. По запросу вывести на консоль заданный текст и *N*-е слово.

22. Напечатать самое длинное слово из заданного текста. Разделители слов - пробельные символы.

23. Исходные данные – файл, содержащий текст на русском языке. Определить, какой процент слов в тексте содержит удвоенную согласную. Разделители слов - пробельные символы.

24. Определить, сколько раз в тексте встречается заданное слово. Разделители слов - пробельные символы.

25. Программа, обратная фильтрации лишних пробелов. В заданном тексте каждый пробельный символ заменить двумя.

26. Исходные данные – файл, содержащий текст на русском языке. Определить, сколько слов в тексте содержат один слог, два слога, 3 слога и т.д.

27. Исходные данные – текстовый файл. Удалить из текста строку, номер которой вводится пользователем по запросу. Строки разделены пробельными символами и заканчиваются разделительными символами. Вывести результат работы программы на консоль.

28. В тексте, состоящем из нескольких строк, обеспечить возможность вставки между заданной своим номером строкой и следующей за ней, пустой строкой и ввода в нее символов.

29. Исходные данные – файл, содержащий текст на английском языке. Определить, какие символы и сколько раз встречаются в тексте.

30. В тексте убрать лишние пробелы и разделительные символы, оставив между словами только по одному пробелу

Методические указания

1. Для всех задач объем исходного текста заранее не известен. При организации ввода текста признак окончания ввода – символ конца файла *EOF* (комбинация клавиш "*Ctrl+Z*", *UNIX/POSIX* стандарт работы с файлами).
2. Действия с фрагментами текста рекомендуется производить как с элементами строковых массивов.
3. Для преобразования текста в максимальной степени использовать библиотечные функции преобразования символьных данных.

Вопросы для собеседования

1. В каком виде символьная информация хранится в памяти ЭВМ.
2. Чем отличаются символьные массивы и строки. Как производится создание и инициализация строки.
3. По каким правилам выполняется лексикографическое сравнение символьных массивов?
4. Характеристика операций над символьными данными, обеспечиваемых библиотечными функциями С и алгоритмы их работы:

   strlen( ) - определение длины строки;
   strset( ) - заполнение строки заданным символом;
   strnset( ) – заполнение части строки заданным символом;
   strcpy( ) – копирование строки в строку;
   strcat( ) – соединение (конкатенация) строк;
   strcmp( ) – сравнение двух строк;
   strtok( ) – поиск и выделение лексических единиц в строке;
   strchr( ) – поиск заданного символа в строке;
   strpbrk( ) – поиск первого вхождения символа из шаблона в строке
   strspn( ) – определение длины начальной части строки-шаблона, которой нет в исследуемой строке;
   strstr( ) – поиск подстроки в строке (по образцу).

   Для указанных функций по алгоритмам составить программную реализацию.

ЛИТЕРАТУРА

1. Гуденко Д.А. Сборник задач по программированию / Д.А. Гуденко, Д.В. Петроченко. – СПб.: Питер, 2003. – 475 с.: ил.
2. Златопольский Д.М. Программирование: типовые задачи, алгоритмы, методы / Д.М.Златопольский. – М.: БИНОМ. Лаборатория знаний, 2007. – 223 с.:ил.
3. Златопольский Д.М. Сборник задач по программированию /Д.М.Златопольский. – СПб.: БХВ-Перетбург, 2011. – 304 с.:ил. – (ИиИКТ)
4. Иванова Г.С. Основы программирования: Учебник для вузов / Г.С.Иванова. – М.: Изд-во МГТУ им.Н.Э. Баумана, 2007. – 416 с.:ил.
5. Лафоре Р. Объектно-ориентированное программирование в С++. Классика Computer Science. 4-е изд. / Р. Лафоре. – СПб.: Питер, 2004. – 924 с.: ил.
6. Климова Л.М. СИ++. Практическое программирование. Решение типовых задач / Л.М. Климова. – М.: КУДИЦ-ОБРАЗ, 2001. – 592с.
7. Роббинс А. Linux: Программирование в примерах. Пер с англ. / А.Роббинс. – М.: КУДИЦ-ОБРАЗ, 2005. – 656 с.
8. Спинелис Диомидис. Анализ программного кода на примере проектов Open Source.: Пер. с англ. / Д.Спинелис. – М.: Издательский дом «Вильямс», 2004. – 528 с.: ил. – Парал. тит. англ.
9. Фуско Дж. Linux. Руководство программиста / Дж.Фуско. – СПб.: Питер, 2011. – 448 с.:ил.
10. Шилдт Г. Полный справочник по С / Г. Шилдт. – М.:Издательский дом «Вильямс», 2002.

ПРИЛОЖЕНИЕ 1.

Справочная информация о использовании функций *printf* и *scanf*.

Форматированный вывод данных

Функция *printf*( ) (прототип содержится в файле *stdio.h*) обеспечивает форматированный вывод. Ее можно записать в следующем формальном виде:

int printf("управляющая строка", аргумент _1, аргумент _2,...);

Управляющая строка содержит компоненты трех типов: обычные символы, которые просто копируются в стандартный выходной поток (выводятся на экран дисплея); спецификации преобразования, каждая из которых вызывает вывод на экран очередного аргумента из последующего списка; управляющие символьные константы.

Каждая спецификация преобразования начинается со знака % и заканчивается некоторым символом, задающим преобразование. Между знаком % и символом преобразования могут встречаться другие знаки в соответствии со следующим форматом:

% [признаки] [ширина_поля] [точность] [*F*|*N*|*h*|*l*|*L*] *c_n*

Все параметры в квадратных скобках не являются обязательными. На месте параметра c_n (символ преобразования) могут быть записаны:

c □ значением аргумента является символ;

d или *i* □ значением аргумента является десятичное целое число;

e □ значением аргумента является вещественное десятичное число в экспоненциальной форме вида 1.23*e*+2;

E □ значением аргумента является вещественное десятичное число в экспоненциальной форме вида 1.23*E*+2;

f □ значением аргумента является вещественное десятичное число с плавающей точкой;

g (или *G*) □ используется, как e или *f*, и исключает вывод незначащих нулей;

o □ значением аргумента является восьмеричное целое число;

s □ значением аргумента является строка символов (символы строки выводятся до тех пор, пока не встретится символ конца строки или же не будет, выведено число символов, заданное точностью);

u □ значением аргумента является беззнаковое целое число;

x □ значением аргумента является шестнадцатеричное целое число с цифрами 0,..., 9, *a*, *b*, *c*, *d*, *e*,*f*;

X значением аргумента является шестнадцатеричное целое число с цифрами 0,..., 9, *A*, *B*, *C*, *O*, *E*, *F*;

p значением аргумента является указатель;

n применяется в операциях форматирования. Аргумент, соответствующий этому символу спецификации, должен быть указателем на целое. В него возвращается номер позиции строки (отображаемой на экране), в которой записана спецификация %n.

Необязательные параметры в спецификации преобразования:

- признак минус (□) указывает, что преобразованный параметр должен быть выравнен влево в своем поле;
- признак плюс (+) требует вывода результата со знаком;
- строка цифр, задающая минимальный размер поля (ширина поля). Здесь может также использоваться символ *, который тоже позволяет задать минимальную ширину поля и точность представления выводимого числа;
- точка (.), отделяющая размер поля от последующей строки цифр;
- строка цифр, задающая максимальное число выводимых символов или же количество цифр, выводимых справа от десятичной точки в значениях типов *float* или *double* (точность);
- символ *F*, определяющий указатель типа *far*;
- символ *N*, определяющий указатель типа *near*;
- символ *h*, определяющий аргумент типа *short int* (используется вместе с символами преобразования *d*, *i*, *o*, *u*, *x*, *X*);
- символ *l*, указывающий, что соответствующий аргумент имеет тип *long* (в случае символов преобразования *d*, *i*, *o*, *u*, *x*, *X*) или *double* (в случае символов преобразования *e*, *E*, *f*, *g*, *G*);
- символ *L*, указывающий, что соответствующий аргумент имеет тип *long doubl*e (используется вместе с символами преобразований *e*, *E*, *f*, *g*, *G*);
- символ #, который может встречаться перед символами преобразования *g*, *f*, *e* и перед символом *x*. В первом случае всегда будет выводиться десятичная точка, а во втором префикс 0*x* перед соответствующим шестнадцатеричным числом.

Если после знака % записан не символ преобразования, то он выводится на экран. Таким образом, строка %% приводит к выводу на экран знака %.

Функция *printf( )* использует управляющую строку, чтобы определить, сколько всего аргументов и каковы их типы. Аргументами могут быть переменные, константы, выражения, вызовы функций; главное, чтобы их значения соответствовали заданной спецификации.

При наличии ошибок, например, в числе аргументов или типе преобразования результаты будут неверными.

Среди управляющих символьных констант наиболее часто используются следующие:

a □ для кратковременной подачи звукового сигнала;
b □ для перевода курсора влево на одну позицию;
f □ для подачи формата;
n □ для перехода на новую строку;
r □ для возврата каретки;
t □ горизонтальная табуляция;
v □ вертикальная табуляция;
\\ □ вывод символа \;
\' □ вывод символа ' ;
\" □ вывод символа ";
\? □ вывод символа ?.

Например, в результате вызова функции:

```
printf("\tЭВМ\n%d\n", i);
```

сначала выполняется горизонтальная табуляция (*t*), т.е. курсор сместится от края экрана, затем на экран будет выведено слово ЭВМ, после этого курсор переместится в начало следующей строки (*n*), затем будет выведено целое число i по формату %d (десятичное целое), и окончательно курсор перейдет в начало новой строки (\n).

Напечатать строку символов можно и так:

```
printf("Это строка символов");
```

Форматированный ввод данных

Функция *scanf*( ) (прототип содержится в файле *stdio.h*) обеспечивает форматированный ввод. Ее можно записать в следующем формальном виде:

int scanf("управляющая строка", аргумент_1, аргумент_2,...);

Аргументы *scanf*( ) должны быть *указателями* на соответствующие значения. Для этого перед именем переменной записывается символ &. Назначение указателей будет подробно рассмотрено далее.

Управляющая строка содержит спецификации преобразования и используется для установления количества и типов аргументов. В нее могут включаться:

- пробелы, символы табуляции и перехода на новую строку (все они игнорируются);
- спецификации преобразования, состоящие из знака %, возможно, символа * (запрещение присваивания), возможно, числа, задающего максимальный размер поля, и самого символа преобразования;
- обычные символы, кроме % (считается, что они должны совпадать с очередными неизвестными символами во входном потоке).

Рассмотрим символы преобразования функции *scanf*( ) (указываются после символа %):

c м на входе ожидается появление одиночного символа;

d или *i* на входе ожидается десятичное целое число и аргумент является указателем на переменную типа *int*;

D или *l* м на входе ожидается десятичное целое число и аргумент является указателем на переменную типа *long*;

e или *E* на входе ожидается вещественное число с плавающей точкой;

f на входе ожидается вещественное число с плавающей точкой;

g или *G* на входе ожидается вещественное число с плавающей точкой;

o на входе ожидается восьмеричное целое число и аргумент является указателем на переменную типа *int*;

O на входе ожидается восьмеричное целое число и аргумент является указателем на переменную типа *long*;

s на входе ожидается появление строки символов;

x на входе ожидается шестнадцатеричное целое число и аргумент является указателем на переменную типа *int*;

X на входе ожидается шестнадцатеричное целое число и аргумент является указателем на переменную типа long;

p на входе ожидается появление указателя в виде шестнадцатеричного числа;

n применяется в операциях форматирования. Аргумент, соответствующий этому символу спецификации, должен быть указателем на целое. В него возвращается номер позиции (после ввода), в которой записана спецификация % *n*;

u на входе ожидается беззнаковое целое число и аргумент является указателем на переменную типа *unsigned int*;

U на входе ожидается беззнаковое целое число и аргумент является указателем на переменную типа *unsigned long*;

[ ] сканирует входную строку для получения символов.

Перед некоторыми символами преобразования могут записываться следующие модификаторы:

F □ изменяет указатель, заданный по умолчанию, на указатель типа *far*;

N □ изменяет указатель, заданный по умолчанию, на указатель типа *near*;

h □ преобразует аргумент к типу *short int* (может записываться перед символами *d*, *i*, *o*, *u*, *x*);

l □ преобразует аргумент к типу *long int* (может записываться перед символами *d*, *i*, *o*, *u*, *x*);

L □ преобразует аргумент к типу *long double* (может записываться перед символами *e*, *f*, *g*).

Ввести целое число (*int a*;), символ (*char b*;) и вещественное число (*float t*;) можно так:

```
scanf("%d", &a);
scanf("%c", &b);
scanf("%d%c%f", &a, &b, &t);
```

Замечание. Функция *scanf*( ) осуществляет "чтение" данных из стандартного потока ввода *stdin* в соответствии со значениями спецификаторов, указанных в управляющей строке. Процесс "чтения" данных из потока синхронизирован с их извлечением. По организации механизма размещения данных *stdin* представляет собой дек, то есть очередь, реализованную по правилу "*FIFO*" (*Fist Input Fist Output* – первым пришел, первым вышел). Однако, если в потоке *stdin* нет данных нужного типа, функция *scanf*( ) не может их самостоятельно извлечь. Для очистки потока *stdin* используется функция *fflush*( ), объявленная в файле *stdio.h*. Поэтому организация ввода данных в формате должна быть реализована по следующей схеме:

```
int Flag;      // переменная, для чтения результата вызова
               // функции scanf()
int X1;        // инициализируемая переменная, значение которой,
               // например, должно быть в диапазоне от 0 до 10
 do
   {
     printf("\nВведите значение целочисленной переменной X1:");
     Flag=scanf("%d",&X1);
     if(Flag==0||Flag==EOF)
       {
         printf("\a Ошибка ввода данных."
```

```
            "\n-----------------------------"
             "\n Повторная инициализации");
        fflush(stdin);
      }
    if(X1<0||X1>40)
      {
        printf("\nВводимое значение должно быть в диапазоне"
             " от 0 до 40. Повторите ввод значения"
             " переменной.");
        Flag=0;
      }
}while(Flag==0||Flag==EOF);
```

ПРИЛОЖЕНИЕ 2 Образец оформления титульного листа отчета

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИМ. Р.Е. АЛЕКСЕЕВА

Кафедра "Вычислительные системы и технологии"

ПРОГРАММИРОВАНИЕ

Отчёт

по лабораторной работе № __

_____
(название работы)
Вариант № ___

Выполнил студент(ка) группы 1_-ИВТ-_
_____
«___» _____ 20___ г.

Провел ст.преподаватель кафедры ВСТ
Мартынов Д.С.
«___» _____ 20___ г.

Нижний Новгород 201_

ПРИЛОЖЕНИЕ 3 Образец оформления титульного листа альбома отчетов

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

ГОУ ВПО НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИМ. Р.Е. АЛЕКСЕЕВА

ИНСТИТУТ РАДИОЭЛЕКТРОНИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОЛГИЙ

Кафедра "Вычислительные системы и технологии"

ПРОГРАММИРОВАНИЕ

Отчёты
о выполнении лабораторного практикума

Студента(ки) группы 1_-ИВТ-_

_____

Провел ст.преподаватель кафедры ВСТ
Мартынов Д.С.

Нижний Новгород 201_

ПРИЛОЖЕНИЕ 4

Использование псевдокода для описания алгоритмов и программ

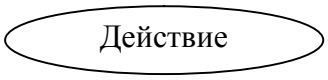
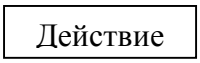
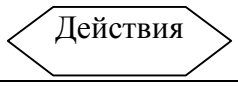
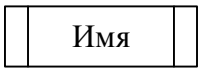
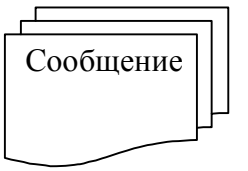
Таблица

Структура	Псевдокод
Следование	<Действие 1> <Действие 2>
Ветвление	**Если** <Условие> **то** <Действие 1> **иначе** <Действие 2> **Все-если**
Выбор	**Выбор** <код> <код 1>: <Действие 1> <код 2>: <Действие 2> . . . **Все-выбор**
Цикл-пока	**Цикл-пока** <Условие> <Действие> **Все-цикл**
Цикл с заданным количеством повторений	**Для** <индекс> = <*n*>,<*k*>,<*h*> <Действие> **Все-цикл**
Цикл-до	**Выполнять** <Действие> **До** <Условие>

ПРИЛОЖЕНИЕ 5

Использование блок-схем для описания алгоритмов и программ

Таблица

Название блока	Обозначение	Название блока
Терминатор		Начало, завершение программы или подпрограммы
Процесс		Обработка данных (вычисления, присваивание и т.д.)
Данные		Операции ввода-вывода
Решение		Ветвление, выбор, итерационные циклы
Подготовка		Счетные циклы
Граница цикла		Любые циклы
Предопределенный процесс		Вызов процедур, функций, передача управления другому процессу
Взаимодействие с пользователем	  	Операции вывода на экран: Вывод данных в стандартный поток. Вывод данных на дисплей Данные, вводимые пользователем с клавиатуры
Соединитель		Маркировка разрывных линий
Комментарий	- - - [Комментарий	Пояснения к программе

ПРИЛОЖЕНИЕ 6

Контрольные вопросы по курсу
«Программирование».
Основы алгоритмизации и программирования на языке Си (*ANSI C99*)
(1-й учебный семестр)

1. Обобщенная структура ЭВМ. Фон Неймановская архитектура ЭВМ.
2. Принцип программного управления.
3. Структура машинной команды.
4. Основные этапы решения задач на ЭВМ.
5. Понятие алгоритма. Виды алгоритмов.
6. Кодирование информации. Позиционные системы счисления.
7. Перевод чисел в позиционных системах счисления.
8. Арифметические операции в позиционных системах счисления.
9. Машинное представление данных. Прямой и обратный дополнительный код.
10. Представление чисел в формате с фиксированной запятой. Представление чисел с плавающей запятой.
11. Классификация языков высокого уровня по назначению.
12. Отладка программы.
13. Восходящее и нисходящее программирование.
14. Общая структура консольного приложения на языке Си (консольная программа). Логическая структура программного модуля. Достоинства и недостатки языка Си.
15. Базовые конструкции языка Си.
16. Базовые типы данных. Модификаторы.
17. Идентификаторы. Правила составления идентификаторов.
18. Ключевые слова.
19. Переменные, область видимости и время жизни. Объявление и инициализация переменных.
20. Классы памяти.
21. Пространство имен.
22. Объявления.
23. Константы. Константное выражение.
24. Выражения. Как определяется тип переменной и тип выражения.
25. Операции. Приоритет операций. Учет переполнения при операциях побитового сдвига.
    Задача: int x=32767; x*=2; x/=2; Каково значение переменной x в случае
    16-разрядной реализации Си.
26. Особенности использования операции *sizeof*().

27. Преобразование типов. Использование операции приведения типа.
28. Операторы языка Си.
29. Форматный ввод-вывод в языке Си, функции *printf*() и *scanf*(). Спецификации форматного ввода-вывода. Управляющие escape последовательности.
30. Массивы. Объявление, инициализация, доступ к элементам массива. Индексное выражение. Приведенный индекс массива.
31. Указатели. Инициализация указателя, операция обращения по адресу.
32. Каким образом определяется тип переменной-указателя? По каким правилам выполняются арифметические операции с переменными указателями.
33. Связь указателей и массивов. Массивы указателей и указатель на указатель.
34. Структура программы и модификаторы типа памяти для указателей в 16-разрядныз ОС. Модели памяти.
35. Указатели и динамические переменные. Резервирование памяти в куче.
36. Функции. Объявление, определение и вызов функции. Способы передачи параметров в функции.
37. Функции с переменным числом параметров.
38. Библиотечные функции.
39. Параметры и возвращаемое функцией *main*() значение.
40. Рекурсия. Рекурсивные функции. Применение рекурсий.
41. Указатель на функцию.
42. Препроцессор языка Си. Директивы препроцессора. Макроопределения (макросы). Условная компиляция.
43. Строки и операции над ними.
44. Сортировка данных. Внутренняя сортировка: сортировка методом прямого включения, сортировка методом прямого выбора, сортировка методом прямого обмена (сортировка методом пузырька).
45. Быстрые методы сортировки: метод Шелла, сортировка с помощью дерева (пирамиды), быстрая сортировка Хоара (сортировка разделением).
46. Процесс взаимодействия системы с клавиатурой. Функции языка Си для работы с клавиатурой.

ПРИЛОЖЕНИЕ 7

Примеры учебных программ

Листинг 1. Создание в памяти компьютера динамического массива для хранения двумерного целочисленного массива и функции для транспонирования матрицы, соответствующей исходному массиву.

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <process.h>

int *A, *B;              /* Указатели на динамические массивы */
int numbRows, numbCols;   /* Размерности массивов */
char *exitMsg="Для завершения работы программы нажмите”
          “ на любую клавишу...";

/* .....Блок объявления пользовательских функций..... */
void transp(int *arr1, int *arr2, int nRows, int nCols);
void initSizeTable(void);
void CreateTable(void);
void inputCountTable(int *arr, int nRows, int nCols);
void displTable(int *arr, int nRows, int nCols);

/* .............Точка входа в программу.............. */
int main(void)
  {
    clrscr();  /* Очистка экрана */
    initSizeTable();
    CreateTable();
    inputCountTable(A,numbRows,numbCols);
    clrscr();
    printf("\nИсходная матрица-массив %d*%d :",numbRows,numbCols);
    displTable(A,numbRows,numbCols);
    transp(A,B,numbRows,numbCols);
    printf("\nТранспонированная матрица-массив %d*%d :",
          numbCols,numbRows);
    displTable(B,numbCols,numbRows);
    printf(exitMsg);
    (void)getch();
    free(A);
    free(B);
```

```
    clrscr();
    printf("\nПравило хорошего тона: уходя уходи...");
    return 0;
  }
/* .......Блок определения пользовательских функций..... */

/*
    Функция транспонирования матрицы –
    целочисленного динамического массива:
    int *arr1 - указатель на исходную матрицу-массив,
    int *arr2 - указатель на транспонированную матрицу-массив.
*/
void transp(int *arr1, int *arr2, int nRows, int nCols)
  {
    int rows, cols;
    for(rows=0;rows<nRows;rows++)
        for(cols=0;cols<nCols;cols++)
          *(arr2+cols*nRows+rows)=*(arr1+rows*nCols+cols);
  }

/*
    Функция для отображения двумерного массива в текстовом режиме
    на экране компьютера с формированием фона за отображаемыми
    данными.
*/
void displTable(int *arr, int nRows, int nCols)
  {
    int rows, cols;
    int curX, curY;
    printf("\n");
    curX=wherex();
    curY=wherey();
    textbackground(LIGHTBLUE);
    for(int i=0; i<nRows; i++)
        {
          for(int j=0; j<nCols*8; j++)
            {
               printf(" ");
            }
          printf("\n");
        }
    gotoxy(curX,curY);
```

```
    for(rows=0; rows<nRows; rows++)
      {
         for(cols=0; cols<nCols; cols++)
           cprintf("%6d ",*(arr+rows*nCols+cols));
         printf("\n");
      }
    textbackground(BLACK);
  }

/* Инициализация глобальных переменных - размерностей массива */
void initSizeTable(void)
  {
    printf("Введите количество строк: ");
    scanf("%d",&numbRows);
    printf("Введите количество столбцов: ");
    scanf("%d",&numbCols);
  }

/* Выделение памяти для динамических массивов */
void CreateTable(void)
  {
    if((A=(int*)calloc(numbRows*numbCols, sizeof(int)))==NULL)
      {
         printf("\nОшибка при создании массива A.");
         printf("\nНет доступной оперативной памяти.");
         printf(exitMsg);
         (void)getch();
         exit(-1);
      }
    if((B=(int*)calloc(numbRows*numbCols, sizeof(int)))==NULL)
      {
         printf("\nОшибка при создании массива B.");
         printf("\nНет доступной оперативной памяти.");
         printf(exitMsg);
         (void)getch();
         exit(-1);
      }
  }

/* Инициализация значений двумерного массива, переданного по ссылке
Пользователь может прервать ввод значений, нажав комбинацию клавиш
Ctrl+Z */
```

```
void inputCountTable(int *arr, int nRows, int nCols)
  {
    printf("\nИнициализация значений массива:\n");
    printf("Замечание: чтобы прервать ввод данных”
          “ нажмите Ctrl+Z\n");
    printf("Неинициализированные значения будут заполнены”
         “ нулями.\n");
    for(int rows=0; rows<nRows; rows++)
       for(int cols=0; cols<nCols; cols++)
          {
            printf("Введие значение A[%d][%d]: ",rows,cols);
            if(scanf("%d",(arr+rows*nCols+cols))==EOF)
            {
               printf("\nДосрочный выход из блока ввода данных...");
               goto END;
            }
          }
END: printf("\nВвод данных окончен. Нажмите любую клавишу для”
          “ продолжения...");
    (void)getch();
  }
```

Листинг 2. Программная реализация генератора псевдо-случайных чисел (ГПСЧ), подчинённых нормальному закону распределения; табличное постраничное отображение смоделированных данных, их сортировка и отображение упорядоченной по возрастанию последовательности.

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <process.h>
#include <math.h>

long int X1, X2;
float Mo, SKO;
FILE *out, *out1;
double Ravn1(void);
double Ravn2(void);
double Norm(void);
void StrSel(double *A, int nn);
void DispArr(double *A, int nn);
void fileRes(FILE *a, double *A, int nn);
```

```
char *msg1="\nПолучение выборки ПСЧ, подчиненных нормальному “
        “закону распределения";
char *msg2="ОШИБКА: нет доступной свободной оперативной память!\n"
        "Перегрузите работающую DOS-сессию, из которой запускается”
     “ программа.";
char *msgExit="Для завершения работы программы нажмите любую”
          “ клавишу...";
char *msgRes="\n\tРабота программы завершена."
          "\n\tВыходные данные моделирования сохранены в текстовых”
       “ файлах: "\n\tB файле GAUS_RES.TXT - полученная “
       “последовательность ПСЧ, "\n\ta в файле GAUS_RES1.TXT”
       “ отсортированные в порядке возрастания \n\t значений “
       “для дальнейшей обработки и построения гистограммы.\n";
int main(void)
  {
    int n;        /* Количество генерируемых значений */
    double  *arr;  /* Указатель на динамический массив данных */
    int    i;

    clrscr();
    printf(msg1);
    printf("\n");
    for(i=0;i<80;i++) putch('*');
    printf("\n\Введите объем выборки (целое положительное число”
         “ не более 100): ");
    scanf("%d",&n);

    if((arr=(double*)calloc(n,sizeof(double)))==NULL)
      {
        printf("\n%s",msg2);
        printf("\n%s",msgExit);
        (void)getch();
        exit(-1);
      }
    //arr1=arr;

    if((out=fopen("gaus_res.txt","w+"))==NULL)
      {
        puts("\n\tОШИБКА: Не удалось создать файл для “
        “результатов...\n");
        printf("Вывод данных будет осуществляться только на “
```

```
            “экран!\n");
      }
    if((out1=fopen("gaus_res1.txt","w+"))==NULL)
      {
         puts("\n\tОШИБКА: Не удалось создать файл для “
         “результатов...\n");
         printf("Вывод данных будет осуществляться только на “
           “экран!\n");
      }
    printf("\n\tВведите значение для инициализации ГПСЧ, “
         “X1[0]:");
    scanf("%ld",&X1);
    printf("\n\tВведите значение для инициализации ГПСЧ, “
         “X2[0]:");
    scanf("%ld",&X2);

/* Блок формирования результатов  */

    printf("\nИнициализация параметров распределения\n");
    for(i=0;i<80;i++) putch('*');
    printf("Введите значения математического ожидания Mo: ");
    scanf("%f",&Mo);
    printf("Введите значение среднего квадратичного отклонения “
         “CKO: ");
    scanf("%f",&SKO);
/* Формирование "нормальной" последовательности */
    for(i=0;i<n;i++)
      {
          *(arr+i)=Norm();
      }
    getch();
    clrscr();
    printf("Полученная последовательность:\n");
    for(i=0;i<n;i++)
      {
         fprintf(out,"\n A[%d] : %7.2f",i,*(arr+i));
      }
    DispArr(arr,n);
    if((fclose(out))!=NULL)
      {
         puts("\n\tCannot close file results");
      }
```

```
    printf("\n\tДля продолжения нажмите любую клавишу.");
    getch();
    clrscr();
    printf("\n\tДанные, упорядоченные по возрастанию:");
    (void)getch();
    StrSel(arr,n);
    DispArr(arr,n);
    if((fclose(out1))!=NULL)
      {
         puts("\n\tCannot close file results");
      }
    free(arr);
    printf(msgRes);
    printf("\n\t%s",msgExit);
    (void)getch();
   // clrscr();
    return 0;
  }

/* ********************************************************** */
double Ravn1(void)
  {
    X1=(430*X1+2531)%11979;
    return X1/11979.;
  }
double Ravn2(void)
  {
    X2=(430*X2+2531)%11979;
    return X2/11979.;
  }
double Norm(void)
  {
    return (Mo+SKO*sqrt(-2*log(Ravn1()))*cos(2*M_PI*Ravn2()));
  }
/* Функция сортировки методом Прямого выбора */
void StrSel(double *A, int nn)
  {
    int i,j,k;
    double x;
    for(i=0;i<nn-1;i++)
      {
         x=*(A+i); k=i;
```

```
        for(j=i+1;j<nn;j++)
          if(*(A+j)<x)
            {
               k=j; x=*(A+k);
            }
          *(A+k)=*(A+i); *(A+i)=x;
       }
  }
void DispArr(double *A, int nn)
  {
/* Данные отображаются постранично в виде таблицы из 5 столбцов */
    int page,index=0,i,j;
    page=nn/100+1;
    while(page>0)
     {
        clrscr();
        for(i=0;i<5;i++)
          {
            for(j=0;j<20;j++)
              {
                if(index>=nn) break; /* Вывод окончен */
                gotoxy(i*16+1,j+2);
                printf("%3d %.2f",index,*(A+index));
                index++;
              }
            if(index>=nn) break;
          }
        gotoxy(2,23);
        printf("Для продолжения нажмите любую клавишу...");
        (void)getch();
        page--;
     }
    printf("\nДанные выведены полностью. Нажмите любую клавишу”
          “ для продолжения.");
    (void)getch();
    clrscr();
  }
void fileRes(FILE *a, double *A, int nn)
  {
/* Данные отображаются постранично в виде таблицы из 5 столбцов */
    int page,col,blok,numbPage=0,index=0,i,j;
    page=nn/100+1;
```

```
  while(page>1)
   {
        for(i=0;i<20;i++)
         {
            for(j=0;j<5;j++)
              {
                index=numbPage*100+j*20+i;
                fprintf(a,"%3d %.2f",index,*(A+index));
              }
            fprintf(a,"\n");
            //if(index>=nn) break;
         }
       numbPage++;
       page--;
   }
 /* Формирование неполной страницы... */
   blok = nn-index; /* Вычисляем, сколько элементов осталось
                       вывести...*/
   col=blok/20;
   if(blok%20>0) col++;
   for(i=0;i<20;i++)
         {
            for(j=0;j<5;j++)
              {
                index=numbPage*100+j*20+i;
                if(index>=nn)continue;
                fprintf(a,"%3d %.2f",index,*(A+index));
              }

            fprintf(a,"\n");
            //if(index>=nn) break;
         }

}
```

ПРИЛОЖЕНИЕ 8

Рекомендации по составлению встроенной документации

Большинство начинающих программистов недооценивает роль и место комментариев, встраиваемых в листинг программы. Это в связано, в первую очередь, с отсутствием опыта работы над большими программными проектами, в которых объем кода может достигать 100 тысяч строк на один программный модуль. Невозможно понять столь большой объём инструкций даже на высокоуровневом языке программирования, не имею дополнительной исчерпывающе информации о бизнес - логике работы приложения.

Кроме этого, студенты просто не знают, как нужно оформлять встроенную документацию, что именно нужно документировать и как можно использовать комментарии для улучшения читаемости исходного кода.

```
/*********************************************************************
*

*********************************************************************
 ********** WARNING: This is an example of a  **********
 **********    warning message that grabs the **********
 **********    attention of the programmer.    **********

*********************************************************************

*********************************************************************
/

/*-------------> Another, less important warning <------------*/

/* >>>>>>>>>>>>>>>>>>> Major section header <<<<<<<<<<<<<<<<<<<
*/

/*********************************************************************
*
 *       We use boxed comments in this book to denote the      *
 *       beginning of a section or program.                    *

*********************************************************************
/
```

```
/*-------------------------------------------------------------*\
 *            This is another way of drawing boxes.            *
\*-------------------------------------------------------------*/

/*
 * This is the beginning of a section.
 * ^^^^ ^^ ^^^ ^^^^^^^^^^ ^^ ^ ^^^^^^^
 *
 * In the paragraph that follow, we explain what
 * the section does and how it works.
 */

/*
 * A medium-level comment explaining the next dozen (or so) line
 * of code. Event though we don't have the bold typeface, we can
 * **emphasize** words.
 */

/*          A simple comment explaining the next line.          */
```

Вышеприведенные примеры заимствованы из книги «Practical C Programming, 3rd Edition By Steve Oualline.»