Белорусский национальный технический университет
Факультет Международный институт дистанционного образования
Кафедра «Информационные системы и технологии»

ЭЛЕКТРОННЫЙ УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ПО УЧЕБНОЙ ДИСЦИПЛИНЕ

«ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ»

для специальности:
6-05-0612-01 «Программная инженерия»

Составители:
Радкевич Андрей Сергеевич, старший преподаватель каф. ИСиТ
Макареня Сергей Николаевич, к.т.н., доцент каф. ИСиТ

Минск БНТУ 2023

Перечень материалов

Электронный учебно-методический комплекс включает:

- теоретический раздел (конспект лекций),
- практический раздел (лабораторные занятия),
- контроль знаний (задания контрольной работе),
- вспомогательный раздел (программа дисциплины, литература, список вопросов)

Пояснительная записка

Электронный учебно-методический комплекс разработан для студентов специальности 6-05-0612-01 «Программная инженерия». Информационное наполнение ЭУМК соответствует программе дисциплины «Основы алгоритмизации и программирования».

ЭУМК может использоваться как при проведении занятий по дисциплине «Основы алгоритмизации и программирования», так и для организации самостоятельной работы студентов. Внедрение ЭУМК будет способствовать эффективной подготовке специалиста, уверенно владеющего возможностями, предоставляемыми современными компьютерными технологиями в среде программирования на алгоритмическом языке высокого уровня, а также программирования вычислительных алгоритмов.

Информация в ЭУМК хорошо структурирована. Теоретический раздел включает основные темы курса. Лабораторный практикум содержит необходимый базовый методический материал (цель лабораторной работы, краткие теоретические сведения, задания на лабораторную работу, контрольные вопросы). В ЭУМК приводится также список литературы и актуальная программа дисциплины. Модуль контроля знаний состоит из экзаменационных вопросов.

ЭУМК разработан в виде pdf-файла и не требует установки специального программного обеспечения

СОДЕРЖАНИЕ

РАЗДЕЛ 1. ТЕОРЕТИЧЕСКИЙ

1. Введение в язык программирования Си

Как выглядит программа, написанная на языке Си? Возможно, вы уже обратили внимание на пример, приведенный раньше, и нашли, что эта программа выглядит довольно специфически из-за наличия в ней символов типа { и \n". Начало темы будет посвящено обсуждению довольно простого примера программы и объяснению того, что она делает. При этом мы рассмотрим некоторые из основных средств языка Си. Если какие-то детали останутся для вас неясными и вы захотите получить более подробные ответы на возникшие вопросы, не огорчайтесь. Мы займемся этим в дальнейшем.

1.1. Пример простой программы на языке Си

Давайте рассмотрим простую программу на языке Си. Следует сразу сказать, что такой пример нужен нам лишь для выявления некоторых основных черт любой программы, написанной на языке Си. Далее мы дадим пояснения к каждой строке, но, перед тем как вы с ними познакомитесь, просто взгляните на программу и попробуйте понять, если сможете, что она будет делать.

```
/* X01.C */
#include<stdio.h>
void main(void) /*Первая программа*/
{
    int n;
    n = 1;
    printf("Я простая");
    printf("вычислительная машина\n");
    printf("Мое любимое число = %d, потому что оно самое первое\n",n);
    /* ################################################ */
        printf("Для продолжения нажмите любую клавишу >>");
        scanf("%d",&n);
    /* ################################################ */
}
```

Если вы считаете, что программа должна вывести нечто на экран дисплея, то вы совершенно правы! Несколько труднее понять, что же появится на экране на самом деле, поэтому давайте выполним программу на ЭВМ и посмотрим к чему это приведет. Первый шаг заключается в использовании имеющегося у вас текстового редактора для создания файла, содержащего текст программы. Этому файлу необходимо будет присвоить какое-то имя; если вам не приходит в голову ничего оригинального, то назовите его main.c. Выполните компиляцию вашей программы. (Для этого вы должны терпеливо ознакомиться с руководством по компилятору, имеющемуся в составе вашей вычислительной системы.) Теперь запустим программу. Если все пойдет хорошо, то результат должен выглядеть следующим образом:

```
Я простая
```

вычислительная машина
Мое любимое число = 1, потому что оно самое первое

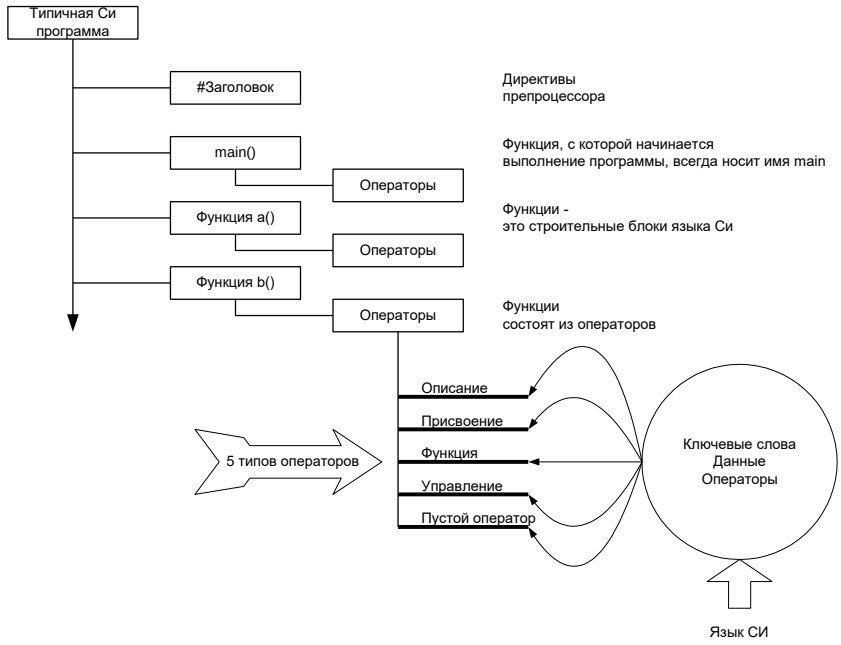
В общем этот результат не кажется особенно неожиданным. Но какую роль в программе выполняют символы \n и %d? И вообще некоторые строки выглядят немного странно. Здесь необходимы дополнительные пояснения.

ПОЯСНЕНИЯ. Мы выполним два просмотра текста программы: во время первого объясним смысл каждой строки, а во время второго — рассмотрим дополнительные вопросы и детали.

Первый просмотр: краткий обзор.

#include <stdio.h> —включение другого файла.

Эта строка указывает компилятору, что нужно включить информацию, содержащуюся в файле stdio.h. main() — имя функции



Структура программы, написанной на языке Си.

Любая программа, написанная на языке Си, состоит из одной или более «функций», являющихся основными модулями, из которых она собирается. Наша программа состоит из одной функции mainn, и круглые скобки указывают именно на то, что main() — имя функции.

/*простая программа*/ — комментарий

Вы можете использовать пары символов /* и */ в качестве открывающей и закрывающей скобок для комментария. Комментарии это примечания, помогающие понять смысл программы. Они предназначены для читателя и игнорируются компилятором.

{ — начало тела функции

Открывающая фигурная скобка отмечает начало последовательности операторов, образующих тело (или определение) функции. Конец определения отмечается

закрывающей фигурной скобкой).

int num; — оператор описания

С помощью такого оператора мы объявляем, что будем использовать в программе переменную num, которая принимает целые (int) значения.

num = 1; — оператор присваивания f

Этот оператор служит для присваивания переменной num значения 1.

printf (" Я простая"); — оператор вывода на печать

С его помощью выводится на печать фраза, заключенная в кавычки:

Я простая

printf ("вычислительная машина\n"); — еще один оператор вывода на печать Этот оператор добавляет слова

вычислительная машина.

в конец последней печатаемой фразы. Комбинация символов \n указывает компилятору на начало новой строки.

printf ("Мое любимое число %d, потому что оно самое первое. \n", num);

Этот оператор выводит на печать значение переменной num (равное 1), содержащееся во фразе в кавычках. Символы %d указывают компилятору, где и в какой форме печатать значение этой переменной num.

} — конец

Как уже упоминалось, программа завершается закрывающей фигурной скобкой.

Теперь рассмотрим нашу программу более внимательно.

Второй просмотр: детали.

#include < stdio.h>;

Файл с именем stdio.h является частью пакета, имеющегося в любом компиляторе языка Си и содержащего информацию о вводе-выводе (например, средства взаимодействия программы с вашим терминалом). В качестве имени файла используется аббревиатура английских слов (программисты называют набор данных, содержащийся в начале файла, заголовком):

Standard input/output header — стандартный заголовок ввода-вывода.

В некоторых случаях включение этой строки в начало программы обязательно, а в некоторых — нет. Мы не можем дать однозначную рекомендацию,. поскольку ответ зависит как от программы, так и от используемой вами вычислительной системы. При работе на нашей системе вводить указанную строку в эту программу совсем не обязательно, но на системе, имеющейся у вас, она может быть необходимой. В любом случае ее использование не принесет никакого вреда. В дальнейшем мы будем указывать эту строку только тогда, когда действительно необходимо.

Возможно, вас удивляет, почему одно из основных средств языка — процедуры ввода-вывода — не включается компилятором в программу автоматически. Дело в том, что этот пакет используется далеко не всегда, а ведь одна из целей создания языка Си — получение компактного объектного кода. Между прочим, упомянутая строка не является даже оператором языка Си. Символ # указывает, что она должна быть

обработана «препроцессором» языка Си. Как вы уже могли предположить из названия, препроцессор осуществляет некоторую предварительную обработку текста программы перед началом компиляции. В дальнейшем мы рассмотрим несколько примеров использования команд препроцессора.

main()

Выбор имени main в качестве названия нашей программы довольно очевиден; более того, назвать ее как-то по-другому и нельзя. Дело в том, что программа, написанная на языке Си, всегда начинает выполняться с функции, называемой main(), поэтому мы имеем возможность выбирать имена для всех используемых нами Функций кроме той, с которой начинается выполнение программы. Зачем здесь скобки? Как уже упоминалось, они указывают на то, что main() — имя функции. Дополнительные вопросы, относящиеся к функциям, будут обсуждаться ниже. Здесь мы только повторим, что функции — это основные модули программы, написанной на языке Си.

В круглых скобках в общем случае содержится информация, передаваемая этой функции. В нашем простом примере передача информации отсутствует и, следовательно, в скобках ничего не содержится. Заканчивая обсуждение данного вопроса, дадим вам один совет: при написании программы старайтесь не пропускать скобок. Файл, содержащий программу, может иметь любое имя, правда, с тем ограничением, что оно должно удовлетворять системным соглашениям и оканчиваться символом .c. Например, вместо main.c мы могли бы выбрать имена mighty.c или silly.c.

/*простая программа*/

Использование комментариев облегчает процесс понимания вашей программы любым программистом (включая вас самих). Большим удобством при написании комментариев является возможность располагать их на той же строке, что и операции, которые они объясняют. Длинный комментарий может помещаться на отдельной строке или даже занимать несколько строк. Все, что находится между символом, указывающим на начало комментария /*, и символом, указывающим на его конец */, игнорируется компилятором, поскольку он не в состоянии интерпретировать язык, отличающийся от Си.

{ и };

Фигурные скобки { } (и только они) отмечают начало и конец тела функции. Для этой цели не используются ни круглые (), ни квадратные [] скобки. Фигурные скобки применяются также для того, чтобы объединить несколько операторов программы в сегмент или «блок». Если вы знакомы с такими языками, как Паскаль или Алгол, вы легко сообразите, что такие скобки аналогичны операторам begin и end в этих языках.

int num;

«Оператор описания переменной» — одно из важнейших средств языка Си. Как уже упоминалось выше, в нашем простом примере вводятся два понятия. Первое — использование в теле функции «переменной», имеющей имя num; второе — с помощью слова int объявляется, что переменная num принимает целые значения. Точка с запятой в конце строки указывает на то, что в ней содержится оператор языка Си, причем этот

символ является здесь частью оператора, а не разделителем операторов, как в Паскале.

Слово int служит «ключевым словом», определяющим один из основных типов данных языка Си. Ключевыми словами называются специальные зарезервированные слова, используемые для построения фраз языка; список ключевых слов вы можете найти в приложении в конце книги.

В языке Си все переменные должны быть объявлены. Это означает, что, во-первых, в начале программы вы должны привести список всех используемых переменных, а во-вторых, необходимо указать «тип» каждой из них. Вообще объявление переменных считается «хорошим стилем» программирования.

Здесь вы можете задать три вопроса. Первый: каким образом надо выбирать имена? Второй: что такое типы данных? Третий: зачем вообще требуется объявлять переменные? Ответы на первый и третий вопросы приведены ниже и отмечены вертикальной линейкой голубого цвета.

Второй вопрос мы обсудим в позднее, а здесь сделаем краткое замечание. Язык Си имеет дело с некоторыми классами (или «типами») данных: целыми числами, символами и числами с плавающей точкой. Объявление переменной, имеющей целый или символьный тип, позволяет компилятору размещать данные в памяти, осуществлять их выборку и интерпретировать нужным образом.

Выбор имен. Мы предполагаем, что вы используете осмысленные обозначения переменных. Имя переменной может содержать от одного до восьми символов. (Фактически вы можете использовать и большее их число, но компилятор пропустит все символы, начиная с девятого. Поэтому имена Shakespeare и shakespencil считались бы одинаковыми, поскольку первые восемь букв у них совпадают.) Для образования имени переменной разрешается использовать строчные и прописные буквы, цифры и символ подчеркивания, считающийся буквой. Первым символом должна быть обязательно буква.

Правильные имена	Неправильные имена
Wiggly	$Z^**
Cat1	1cat
Hot__Tub	Hot—Tub
_kcaB	don't

В библиотечных процедурах часто используются имена, начинающиеся с символа подчеркивания. Это делается в предположении, что пользователи вряд ли выберут имена, начинающиеся с этого символа, поэтому маловероятно, что одно из них будет случайно выбрано для обозначения другого понятия. Старайтесь не использовать имен, начинающихся с символа подчеркивания, и вам удастся избежать взаимопересечений с множеством библиотечных имен.

Четыре довода в пользу объявления переменных.

1. Сведение всех операторов объявления переменных в начало программы облегчает понимание ее смысла. Это особенно справедливо, если вы даете переменным осмысленные имена (например, taxrate [налоговый тариф] вместо r) и, кроме того, включаете в программу комментарии для объяснения того, что обозначают переменные. Документирование программы подобным образом является одним из основных признаков хорошего стиля программирования.
2. Размышление о том, что поместить в секцию объявления переменных, побуждает спланировать программу перед тем, как погрузиться в ее написание. Это эквивалентно получению ответов на вопросы: какая информация необходима

программе при запуске? Какую выходную информацию хотелось бы получить?

3. Объявление переменных позволяет избежать одной из наиболее коварных и труднообнаруживаемых ошибок — неправильно написанных имен. Например, предположим, что программируя на некотором языке, вы использовали оператор

   B0ZO = 32.4;

   а дальше в программе вы ошибочно написали ANS = 19.7*BOZO - 2.0

   случайно заменив цифру 0 буквой О. Вследствие этого в программе появится новая переменная с именем BOZO, и будет использовано какое-то ее значение (возможно нуль или какой-то «мусор»). В результате переменная ANS получит неправильное значение, и вы, возможно, потратите много времени, пытаясь найти причину. Это не может произойти при программировании на языке Си (если только вы не объявили две переменные со столь похожими именами), поскольку компилятор сразу выдаст сообщение об ошибке, как только встретит в программе необъявленную переменную с именем BOZO.
4. Любая программа, написанная на языке Си, не будет выполняться, если не описать все используемые переменные. Мы полагаем, что последний довод окажется решающим в том случае, если первые три вас не убедили.

   num = 1;

«Оператор присваивания» является одним из основных средств языка. Приведенную выше строку программы можно интерпретировать так: «присвоить переменной num значение 1». Дело в том, что, согласно оператору в четвертой строке программы, переменной num была выделена ячейка памяти, и только теперь в результате выполнения оператора присваивания переменная получает свое значение. При желании мы могли бы присвоить ей другое значение — вот почему имя num обозначает переменную. Отметим, что этот оператор тоже заканчивается точкой с запятой.

```
printf (" Я простая" );
printf (" вычислительная машина\n");
printf (" Мое любимое число %d, потому что оно самое первое. \n" , num);
```

Во всех этих строках используется стандартная функция языка Си, называемая printf(); скобки указывают на то, что мы, конечно же, имеем дело с функцией. Строка символов, заключенная в скобки, является информацией, передаваемой функции printf() из нашей главной функции [main()].

Такая информация называется «аргументом»; в первом случае аргументом является строка " Я простая". Возникает вопрос: что функция printf() делает с этим аргументом? Ответ довольно очевиден: она просматривает все символы, содержащиеся между кавычками, и выводит их на экран терминала.

Данная строка дает нам пример того, как мы «вызываем» функцию или «обращаемся» к ней, программируя на языке Си. Для этого требуется только указать имя функции и заключить требуемый аргумент (или аргументы) в скобки. Когда при выполнении ваша программа «достигнет» этой строки, управление будет передано указанной функции [в данном случае printf()]. Когда выполнение функции будет завершено, управление вернется обратно в исходную («вызывающую») программу.

Что можно сказать по поводу следующей строки программы? В ней имеются

символы \n, которые не появились на экране. В чем дело? Эти символы служат директивой начать новую строку на устройстве вывода. Комбинация \n на самом деле представляет собой один символ, называемый «новая строка». Его смысл кратко формулируется так: начать вывод новой строки с самой левой колонки. Другими словами, с помощью этого символа осуществляются те же функции, что и с помощью клавиши [ввод], имеющейся на обычном терминале. Но вы можете сказать, что комбинация \n выглядит, как два символа, а не как один. Вы, конечно же, правы, но просто по смыслу они представляют собой один символ, для которого не существует соответствующей клавиши на клавиатуре. Возникает вопрос: почему для этой цели нельзя использовать клавишу [ввод]? В ответ скажем, что это может быть интерпретировано как некоторая директива вашему текстовому редактору, а не как команда, которая должна быть помещена в память ЭВМ. Другими словами, когда вы нажимаете клавишу [ввод], редактор прекращает заполнение текущей строки, с которой вы в данный момент работаете, и начинает новую строку, оставляя старую неоконченной.

Символ «новая строка» служит одним из примеров того, что называется «управляющей последовательностью». Эта последовательность используется для представления символов, которые трудно или вообще невозможно вводить с обычной клавиатуры. Другими примерами служат \t для табуляции и \b для возврата на одну позицию. В любом случае управляющая последовательность начинается со знака \.

Теперь, мы думаем, стало понятно, почему три оператора печати вывели на экран только две строки: аргумент первого оператора не содержал символа «новая строка».

Вид второй строки, появившейся на экране, может вызвать недоуменный вопрос: почему отсутствуют символы %d, имеющиеся в операторе вывода? Напомним, что напечатанная строка имела следующий вид:

Мое любимое число 1. потому что оно самое первое.

Вы, наверное, уже догадались — при печати вместо символов %d было подставлено число 1, являющееся значением переменной num. По-видимому, комбинация символов %d служит своего рода указателем места в строке, куда необходимо вставить значение переменной num при печати. На языке Бейсик аналогичный оператор печати выглядел бы следующим образом:

PRINT "Мое любимое число"; num; "потому что оно самое -первое".

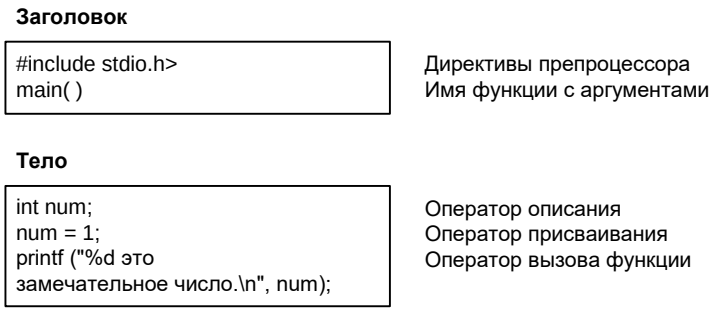
На самом деле в языке Си обсуждаемый оператор позволяет сделать несколько больше. Символ % сигнализирует программе, что, начиная с этой позиции, необходимо напечатать число, а d указывает, что переменную необходимо печатать в десятичном формате. Функция printf() предоставляет возможность выбора соответствующего формата для печати переменных. Буква f в имени функции printf() фактически служит напоминанием, что это оператор печати с заданным форматом.

1.2. Структура простой программы

Теперь, после того как мы привели конкретный пример, вы готовы к тому, чтобы познакомиться с несколькими общими правилами, касающимися программ, написанных на языке Си. Программа состоит из одной или более функций, причем

какая-то из них обязательно должна называться main(). Описание функции состоит из заголовка и тела. Заголовок в свою очередь состоит из директив препроцессора типа #include и т. д. и имени функции. Отличительным признаком имени функции служат круглые скобки, причем аргумент, вообще говоря, может отсутствовать. Тело функции заключено в фигурные скобки и представляет собой набор операторов, каждый из которых оканчивается символом «точка с запятой». В нашем примере тело состояло из оператора описания, в котором объявлялись имя и тип используемой переменной, оператора присваивания, с помощью которого переменная получила некоторое значение, и, наконец, трех операторов печати, каждый из которых представляет собой вызов функции

printf().



1.3. Как сделать программу читаемой

Создание читаемой программы служит признаком хорошего стиля программирования. Это приводит к облегчению понимания смысла программы, поиска ошибок и в случае необходимости ее модификации. Действия, связанные с улучшением читаемости программы, кроме того, помогут более четко понять, что программа делает. На протяжении всего изложения мы будем пытаться указывать полезные приемы, способствующие достижению этой цели.

Мы уже упоминали о двух таких способах: выбор осмысленных обозначений для переменных и использование комментариев. Заметим, что эти два метода дополняют друг друга. Если вы дали переменной имя width (ширина), то необходимость в комментарии, сообщающем о том, что данная переменная определяет ширину, отпадает.

Еще один прием состоит в использовании пустых строк для того, чтобы отделить одну часть функции, соответствующую некоторому семантическому понятию, от другой. Например, в нашей простой программе одна пустая строка отделяет описательную часть от выполняемой (присваивание значения и вывод на печать). Синтаксические правила языка Си не требуют наличия пустой строки в данном месте, но поскольку это стало уже традицией, то и мы делаем также.

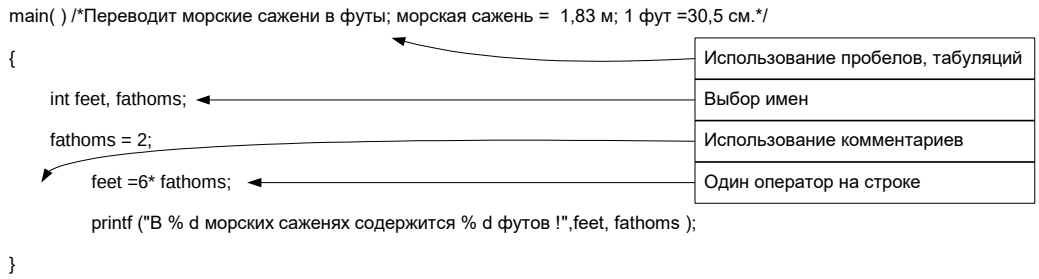
Четвертый принцип, которому мы следуем, заключается в том, чтобы помещать каждый оператор на отдельной строке. Опять же это только соглашение, которое никак не регламентируется правилами языка, так как Си имеет «свободный формат». Вы можете

поместить несколько операторов на одной строке или распространить один

оператор на несколько строк. Нижеследующий пример является абсолютно правильной программой:

```
main() {int four; four
=
4
;
printf(
“%d\n”,
four);
```

Совершенно очевидно, что символ «точка с запятой» указывает компилятору, где кончается один оператор и начинается следующий, но логика программы окажется проще, если вы последуете соглашениям, приведенным выше. Поскольку в нашем примере запутанной логики нет, вид программы в данном случае не влияет на понимание ее смысла, но, по нашему мнению, лучше прививать хорошие привычки с самого начала.



Следующий шаг. Наша первая программа была довольно простой, и следующий пример будет ненамного сложнее. Он выглядит так:

```
main()/* Переводит 2 морские сажени в футы*/
{
    int feet; fathoms;

    fathoms=2;
    feet= 6*fathoms;
    printf ("В %d морских саженях содержится %d футов!", feet, fathoms);
}
```

Что здесь нового? Во-первых, мы описали две переменные вместо одной. Для этого потребовалось только разделить в операторе описания имена двух переменных запятой.

Во-вторых, мы выполнили вычисления — использовали громадную вычислительную мощность нашего компьютера для умножения 2 на 6. В Си, так же как и во многих других языках, символ * обозначает умножение. Поэтому смысл оператора

```
feet = 6*fathoms;
```

заключается в следующем: взять величину переменной fathoms, умножить ее на 6 и присвоить результат переменной feet. (Судя по этой парафразе, обычный английский язык менее лаконичен, чем простой язык Си; это одна из причин, лежащих в основе разработки языков программирования.)

И наконец, мы использовали функцию printf() более сложным образом. Если вы выполните эту программу на компьютере, то результат должен выглядеть так:

В 2 морских саженях содержится 12 футов!

Можно заметить, что было произведено две подстановки: первое вхождение символов %d в строку, заключенную в кавычки, было заменено значением первой переменной (fathoms) из списка, следующего за указанной строкой, а второе — значением второй переменной (feet) из этого же списка. Обратите внимание, что список печатаемых переменных расположен в конце оператора.

Область применения данной программы несколько ограниченна, но она может послужить прообразом программы перевода морских саженей в футы. Все, что нам потребуется — специальный способ присваивания произвольных значений переменной feet; о том, как это делается, вы узнаете несколько позже.

Дополнительный пример. Здесь мы приведем еще один пример. До сих пор в наших программах использовалась только стандартная функция printf(). В данном разделе мы хотим продемонстрировать, как включить в программу и использовать функцию, которую вы сами написали.

```
main()/* butler*/
{
printf("Я вызываю функцию butler.\n");
butler();
printf ("Да. Принесите мне чашку чая и гибкие диски.\n");
}

butler()
{
printf("Вы вызывали, сэр?\n");
}
```

Результаты работы программы выглядят следующим образом:

```
Я вызываю функцию butler.
Вы вызывали, сэр?
Да. Принесите мне чашку чая и гибкие диски.
```

Функция butler() определяется точно так же, как и функция main(); ее тело заключено в фигурные скобки. Вызов функции осуществляется путем простого указания ее имени, включая круглые скобки. Мы вернемся к этому важному вопросу позднее, а здесь хотели продемонстрировать ту легкость, с которой вы можете включать в программу свои собственные функции.

Что вы должны были узнать. Ниже приведена краткая сводка строгих правил (но не чрезмерно жестких), которые, мы надеемся, вы усвоили. Мы включили сюда же краткие примеры.

Как назвать файл, содержащий вашу программу: eye.c, или black.c, или infan.c и т. п.

Какое имя можно использовать в качестве названия программы, состоящей из одной функции: main()

Структура простой программы: заголовок, тело, фигурные скобки, операторы

Как описать целую переменную: int varname; Как присвоить значение переменной: varname = 1024; Как напечатать фразу printf ("Хотите купить утку?"); Как напечатать значение переменной: printf ("%d", varname); Символ новая строка: \n

Как включать комментарии в программу: /*анализ движения наличных денег*/

Вопросы и ответы. Ниже приведено несколько вопросов, которые помогут вам проверить и расширить свое понимание материала.

Вопросы

1. Икабод Боуди Марфут (ИБМ) подготовил программу, приведенную ниже, и принес ее вам для проверки. Пожалуйста, помогите ему найти в ней ошибки.

include studio.h

main() /*эта программа печатает число недель в году/*

(

int s

s: = 56;

print (В году s недель.);

2. Что будет напечатано в каждом из примеров, приведенных ниже, в предположении, что они являются частями некоторых полных программ?

а. printf(" Б-э-э Б-э-э, Черная Овца.");

а. printf("У тебя есть шерсть?\n");

б. printf("Убирайся!\n Мешок сала!");

в. printf("Что?\n Нет/n Кларнет?\n");

г. int num; num = 2;

printf("%d + %d = %d", num, num, num + num);

Ответы.

1. Строка 1: данная строка должна начинаться с символа #; правильное написание имени файла — stdio.h; имя файла должно быть заключено в угловые скобки. Строка 2: вместо фигурных скобок () необходимо использовать круглые (); комментарий должен оканчиваться символами */, а не /* Строка 3: вместо круглой скобки (должна стоять фигурная {. Строка 4: оператор должен оканчиваться символом «точка с запятой». Строка 5: эту строку (пустую) м-р ИБМ написал совершенно правильно! Строка 6: в операторе присваивания необходимо использовать символ =, а не : =. (К сожалению, м-р ИБМ имеет представление о языке Паскаль.) В году 52 недели, а не 56

Строка 7: оператор должен выглядеть так printf ("В году %d недель. \n*,.s); Строка 8: отсутствует, но она обязательно должна быть и содержать закрывающую фигурную скобку —).

2 а Б-э-э Б-э-э, Черная Овца.У тебя есть шерсть?

(Заметим, что пробел после точки отсутствует. Для того чтобы поместить в это место пробел, необходимо было вместо "У тебя писать" У тебя)

б. Убирайся!

Мешок сала!

(Отметим, что курсор теперь находится в конце второй строчки.)

в. Что?

Нет/n Кларнет?

Заметим, что символ (/) производит не тот же эффект, как символ (\)

г. 2+2= 4

(Отметим, что каждое вхождение комбинации символов %d в строку заменяется значением соответствующей переменной из списка. Заметим также, что символ + означает сложение и что таким образом вычисления могут быть проведены «внутри» оператора printf().)

Упражнения. Чтобы изучить язык Си, одного только чтения недостаточно. Вы должны попробовать сами написать одну или две простые программы и посмотреть, пройдет ли все так же гладко, как это может показаться в результате этого материала. Мы хотим предложить вам несколько идей, но, если желаете, вы можете воспользоваться своими собственными соображениями на этот счет.

1. Напишите программу, печатающую ваше имя.

2. Напишите программу, печатающую ваши имя и адрес, используя три или более строк.

3. Напишите программу, которая укажет ваш возраст, данный в годах, в днях. Не усложняйте ее, учитывая високосные и невисокосные годы.

2. Данные языка программирования C/C++

Программы имеют дело с данными. Мы вводим в компьютер числа, буквы и слова и ожидаем, что он будет проводить над ними какие-то операции. Здесь мы сосредоточим наше внимание на данных различных типов и их свойствах. В соответствии с этим мы будем последовательно останавливаться на каждом из типов и смотреть, как их можно использовать. Но, поскольку заниматься одним только обсуждением представляется нам не очень веселым делом, мы рассмотрим также небольшие программы обработки данных.

Этот материал в основном посвящена обсуждению двух важнейших классов типов данных: целым числам и числам с плавающей точкой. Язык Си предоставляет программисту возможность использовать несколько разновидностей этих типов. Мы займемся изучением следующих вопросов: что такое типы данных, как их описать, как и когда их использовать. Кроме того, мы обсудим различия между константами и переменными.

2.1. Данные: переменные и константы

Компьютер, выполняя программу, может заниматься разнообразной деятельностью. Он может складывать числа, сортировать имена, заниматься распознаванием речи и изображения на экране видеодисплея, вычислять орбиты комет, подготавливать список почтовых адресов абонентов, чертить фигуры, делать логические выводы или что-нибудь еще, что только вы можете себе представить. Чтобы заниматься всем этим, программам необходимо работать с «данными» — числами и символами, т. е. объектами, которые несут в себе информацию, предназначенную для использования. Некоторые данные устанавливаются равными определенным значениям еще до того, как программа начнет выполняться, а после ее запуска сохраняют такие значения неизменными на всем протяжении работы программы. Это «константы». Другие данные могут изменяться, или же им могут быть

присвоены значения во время выполнения программы; они называются «переменными». Мы уже использовали данные термины, но вы знакомитесь с ним только здесь. Вспомните формулу площади круга при рассмотрении различных алгоритмов.

Различие между переменной и константой довольно очевидно: выполнения программы значение переменной может быть (например, с помощью присваивания), а значение константы — нет. Указанное различие приводит к тому, что обработка переменных компьютером оказывается немного сложнее и требует больше времени, чем обработка констант, но, несмотря на это, он вполне справляется с такой деятельностью.

2.2. Данные: типы данных

Помимо различия между переменными и константами существует еще различие между типами данных. Некоторые данные в программе являются числами, некоторые — буквами, или, более обобщенно, символами. Компьютер должен иметь возможность идентифицировать и обрабатывать требуемым образом данные любого из этих типов. В языке Си предусмотрено использование нескольких основных типов данных. Если величина есть константа, то компилятор обычно может распознать ее тип только по тому виду, в каком она присутствует в программе. Однако в случае переменной необходимо, чтобы ее тип был объявлен в операторе описания. Дополнительные детали, относящиеся к типам данных, мы будем сообщать вам по мере изложения. Рассмотрим основные типы данных, имеющиеся в языке Си. В стандарте языка Си используется семь ключевых слов, указывающих на различные типы данных. Приведем список этих ключевых слов:

Int
Long
Short
unsigned

char

float
double

Первые четыре ключевых слова используются для представления целых, т. е. целых чисел без десятичной дробной части. Они могут появляться в программе по отдельности или в некоторых сочетаниях, как, например, unsigned short. Следующее слово char предназначено для указания на буквы и некоторые другие символы, такие, как #, $, % и &. Последние два ключевых слова используются для представления чисел с десятичной точкой. Типы, обозначаемые этими ключевыми словами, можно разделить на два класса по принципу размещения в памяти машины. Первые пять ключевых слов определяют «целые» типы данных, в то время как последние два — типы данных с «плавающей точкой».

В этом месте у некоторых из вас могут появиться недоуменные вопросы: «Целые типы данных? Типы данных с плавающей точкой?» Не пугайтесь. Если эти термины

кажутся вас непривычными или непонятными, мы дадим краткое объяснение их смысла.

Биты, байты и слова. Термины «бит», «байт» и «слово» обычно используются для описания как элементов данных, которые обрабатывает компьютер, так и элементов памяти. Здесь мы займемся рассмотрением второго смысла этих терминов.

Наименьшая единица памяти называется бит. Она может принимать одно из двух значений: 0 или 1. (Иначе говоря, бит может находится в состояниях «включен» или «выключен»; эта фраза совершенно аналогична первому высказыванию.) В один бит нельзя поместить достаточное количество информации, но в машине содержится большое число битов; дело в том, что бит — основной «строительный блок», из которых создается память компьютера.

Байт — более удобный элемент памяти. В большинстве машин байт состоит из 8 бит. Поскольку каждый бит можно установить либо в состояние 0, либо в состояние 1, всего в байтовом формате можно представить 256 (два в восьмой степени) различных комбинаций из нулей и единиц. Такие комбинации можно использовать, например, для представления целых чисел в диапазоне от 0 до 255 или для кодирования набора символов. Это можно получить при помощи «двоичного кода», в котором для представления чисел используются только нули и единицы.

При современном подходе к проектированию компьютеров слово является самым естественным элементом памяти. В 8-разрядных микрокомпьютерах, таких, как ЭВМ фирмы Sinklair или первые модели машин фирмы Apple, слово занимает как раз 1 байт. Многие более новые персональные вычислительные системы, такие, как IBM PC и Lisa фирмы Apple, являются 16-разрядными. Это означает, что размер слова у них 16 бит, т. е. 2 байта. Большие компьютеры могут иметь 32-, 64-разрядные слова или даже более длинные. Совершенно очевидно, что чем длиннее слово, тем больше информации можно туда поместить. Обычно в компьютерах предусмотрена возможность объединять вместе два или более слов для того, чтобы помещать в память элементы данных большей длины, но этот процесс сильно замедляет работу компьютера.

В наших примерах мы предполагаем, что длина слова равна 16 бит, если мы не оговорили противного.

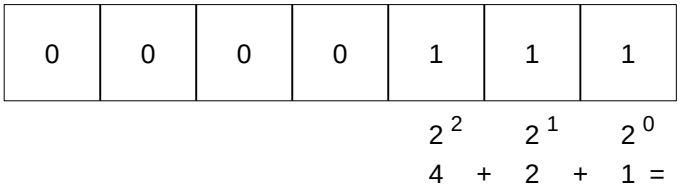
Для человека различие между целым числом и числом с плавающей точкой выражается в способе записи. Для компьютера различие выражается в способе занесения этих чисел в память. Давайте рассмотрим по очереди каждый из двух классов чисел.

Целые числа

У целого числа никогда не бывает дробной части и, согласно правилам языка Си, десятичная точка в его записи всегда отсутствует. В качестве примера можно привести числа 2, —23 и 2456. Числа вида 3.14 и 2/3 не являются целыми. Представив целое число в двоичном виде, его нетрудно разместить в памяти машины.

Например, число 7 в двоичном виде выглядит как 111. Поэтому, чтобы поместить это число в 1-байт слово, необходимо первые 5 бит установить в 0, а последние 3 бит — в 1.

Один байт

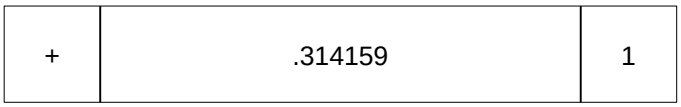


Числа с плавающей точкой

Числа с плавающей точкой более или менее соответствуют тому, что математики называют «вещественными числами». Они включают в себя числа, расположенные между целыми. Вот некоторые из них: 2.75, 3.16E7, 7.00 и 2e-8. Очевидно, что любое число с плавающей точкой можно записать несколькими способами. Более полное обсуждение «E-нотации» будет проведено дальше, а мы только кратко поясним, что запись вида «3.16E7» означает число, полученное в результате умножения 3.16 на 10 в седьмой степени, т. е. на 1 с семью нулями. Число 7 называется «порядком» (показателем степени при основании 10).

Наиболее существенным моментом здесь является то, что способ кодирования, используемый для помещения в память числа с плавающей точкой, полностью отличается от аналогичной схемы для размещения целого числа. Формирование представления числа с плавающей точкой состоит в его разбиении на дробную часть и порядок; затем обе части раздельно помещаются в память. Поэтому число 7.00 из вышеприведенного списка нельзя поместить в память тем же способом, что и целое число 7, хотя оба имеют одно и то же значение. В десятичной записи (точно так же как и в двоичной) число «7.0» можно было бы записать в виде «0.7E1»; тогда «0.7» будет дробной частью, а «1» — порядком. Для размещения чисел в памяти машины будут, конечно, использоваться двоичные числа и степени двойки вместо степеней десяти. Здесь мы остановимся лишь на различиях, связанных с практическим использованием чисел этих двух типов.

5. Целые числа не имеют дробной части, в то время как числа с плавающей точкой могут представлять как целые, так и дробные числа.
6. Числа с плавающей точкой дают возможность представлять величины из более широкого диапазона, чем целые.
7. При некоторых арифметических операциях, например при вычитании одного большого числа из другого, использование чисел с плавающей точкой приводит к большей потее точности.
8. Операции над числами с плавающей точкой выполняются, как правило, медленнее, чем операции над целыми числами. Однако сейчас уже появились микропроцессоры, специально ориентированные на обработку чисел с плавающей точкой, и в них эти операции выполняются довольно быстро.



Знак | Дробная часть | Показатель степени

$+ \quad .314159 \quad * \quad 10^{1} \quad = \quad 3.14159$

Ошибки округления чисел с плавающей точкой. Возьмите некоторое число. Добавьте к нему 1, а затем вычтите из полученной суммы исходное число. Что у вас получится? У нас получилась 1. Но вычисления, производимые над числами с плавающей точкой, могут дать и совершенно неожиданный результат:

```
/*ошибка вычислений*/
main()
{
    float a, b;
    b = 2.0e20 +1.0;
    a = b - 2.0e20;
    printf(" %f \n", a);
}
```

Результат равен
0.000000

Причина появления такого странного результата состоит в отсутствии достаточного числа разрядов для выполнения операций с требуемой точностью. Число 2.0e20 записывается как двойка с последующими двадцатью нулями, и, добавляя к нему 1, мы пытаемся изменить 21-ю цифру. Чтобы выполнить эту операцию корректно, программа должна иметь возможность поместить в память число, состоящее из 21 цифры. Но число типа float (т. е. с плавающей точкой) путем изменения порядка можно увеличить или уменьшить лишь на 6 или 7 цифр. Попытка вычисления оказалась неудачной. С другой стороны, если бы мы использовали, скажем, число 2.0e4 вместо 2.0e20, мы смогли бы получить правильный ответ, поскольку в этом случае мы пытались бы изменить 5-ю цифру, и точность представления чисел типа float оказалась бы вполне достаточной для этого.

2.3. Типы данных в языке Си

Давайте теперь рассмотрим некоторые специфические особенности основных типов данных, используемых в языке Си. Для каждого типа мы покажем, как описать переменную, как представить константу и как лучше всего использовать данные этого типа. В некоторых компиляторах с языка Си не реализована обработка всех типов данных; поэтому вам необходимо свериться с руководством по языку Си, имеющимся в комплекте вашей машины, чтобы посмотреть, какие из типов доступны для использования.

Типы int, short и long

В языке Си имеется несколько целых типов, поэтому у вас есть возможность вносить изменения в свою программу, чтобы она удовлетворяла требованиям конкретной машины или определенного задания. Если вы не хотите заботиться о таких деталях, то, вообще говоря, вы можете просто остановиться на типе int и не думать больше о других возможностях.

Все данные типов int, short и long являются «числами со знаком», т. е. допустимыми значениями переменных этих типов могут быть только целые числа — положительные, отрицательные и нуль. Один бит используется для указания знака числа, поэтому максимальное число со знаком, которое можно представить в слове, меньше, чем максимальное число без знака. Например, в формате 16-битного слова можно представить любые целые числа без знака из диапазона от 0 до 65535. Точно так же 16-битное слово можно использовать для представления целых чисел со знаком из диапазона от — 32768 до + 32767. Заметим, что длины диапазонов в обоих случаях одинаковые.

Язык Си предоставляет пользователям возможность выбора размера элемента памяти (одного из трех) для представления целых чисел. Типу int обычно соответствует стандартная длина слова, принятая на используемой машине. При этом гарантируется, что размер элементов памяти, отводимых под данные типа short и long, будет соответственно не больше и не меньше длины элемента памяти, выделяемого типу int. В некоторых вычислительных системах один или оба этих типа реализованы точно так же, как int. Все зависит от того, какое представление лучше соответствует архитектуре конкретной ЭВМ. В табл. 3.1 для каждого компьютера из некоторого множества приведено число битов, используемое для представления данных различных типов, а также диапазоны отображаемых чисел.

Описание данных целого типа

При описании данных необходимо ввести только тип, за которым должен следовать список имен переменных. Ниже приведены некоторые возможные примеры описаний:

```
int erns;
short stops;
long Johns;
int hogs, cows, goats;
```

В качестве разделителя между именами переменных необходимо использовать запятую; весь список должен оканчиваться символом «точка с запятой». Вы можете собрать в один оператор описания переменных с одним и тем же типом или, наоборот, разбить одно описание на несколько операторов. Например, описание

```
int erns, hogs, cows, goats;
```

будет давать тот же самый эффект, что и два отдельных описания типа int в предшествующем примере. При желании вы даже могли бы использовать четыре различных описания данных типа int — по одному для каждой переменной. Иногда вам

могут встретиться сочетания ключевых слов, как, например, long int или short int. Эти комбинации являются просто более длинной записью ключевых слов long и short.

Целые константы

Согласно правилам языка Си, число без десятичной точки и без показателя степени рассматривается как целое. Поэтому 22 и -273 — целые константы. Но число 22.0 нецелое, потому что в его записи имеется десятичная точка, и число 22E3 тоже нецелое, поскольку в записи использован порядок. Кроме того, указывая целое число, нельзя использовать запятые. Нужно записать 23456 вместо 23,456.

Если вы хотите ввести некоторую константу типа long, то можете это сделать, указав признак L или 1 в конце числа. Использование прописной буквы L более предпочтительно, поскольку ее труднее спутать с цифрой 1. Примером такой константы служит число 212L. Очевидно, что само по себе число 212 не очень большое, но добавление признака L гарантирует, что в памяти для него будет отведено необходимое число байтов. Это может оказаться важным для достижения совместимости, если данное число должно использоваться вместе с другими переменными и константами типа long.

Вполне возможно, что вам уже ничего больше не нужно знать про то, как записывают константы, но в языке Си имеются еще и два других способа.

Первый: если целое начинается с цифры 0, оно интерпретируется как «восьмеричное» число. Восьмеричные числа — это числа, представляемые «по основанию восемь» (т. е. их запись состоит из комбинаций степеней числа восемь). Например, 020 — это удвоенная первая степень основания восемь, т. е. восьмеричный эквивалент числа 16. При отсутствии в первой позиции нуля это просто обыкновенное (десятичное) число 20.

Второй: целое, начинающееся с символом 0x или 0X интерпретируется как шестнадцатеричное число, т. е. число, записываемое по основанию 16. Поэтому запись 0x20 представляет собой удвоенную первую степень числа 16, или 32.

Восьмеричные и шестнадцатеричные числа чрезвычайно популярны среди программистов. Поскольку 8 и 16 являются степенями числа 2, а 10 — нет, использование этих систем счисления при работе на машине является вполне естественным. Например, число 65536, которое часто возникает при программировании на 16-Разрядных компьютерах, в шестнадцатеричной записи имеет вид 10000.

Инициализация переменных целого типа

Константы часто применяются при «инициализации» переменных. Это означает присваивание переменной некоторого значения перед началом обработки. Ниже приводятся примеры использования инициализации:

```
erns = 1024;
stops = -3;
Johns = 12345678;
```

Если захотите, вы можете инициализировать переменную в операторе описания. Например:

```
int hogs = 23;
int cows = 32, goats =14;
short dogs, cats = 92;
```

Заметим, что в последней строке была инициализирована только переменная cats. При невнимательном чтении может создаться впечатление, что переменная dogs тоже инициализирована значением 92, поэтому лучше избегать смешивания инициализируемых и неинициализируемых переменных в одном операторе описания.

Рекомендации. Какие переменные целого типа со знаком лучше всего использовать? Одной из целей введения в язык трех классов целых чисел, имеющих различные размеры, было предоставить возможность согласования типа переменной с требованиями задачи. Например, если переменная типа int занимает одно слово, а переменная типа long — два, то тип long позволяет обрабатывать большие числа. Если в вашей задаче такие большие числа не используются, то незачем и вводить в программу переменную типа long, так как, если вместо числа, занимающего одно слово памяти, используется число, занимающее два слова, работа машины замедляется. Вообще говоря, необходимость введения данных типа long целиком зависит от вашей вычислительной системы, поскольку под данные типа int на одной машине может отводиться больше памяти, чем под данные типа long на другой. В конце мы еще раз хотим напомнить вам, что обычно вполне достаточно использовать переменную типа int.

Переполнение, возникающее при обработке целых чисел. Что происходит, когда в процессе обработки данных появляется значение, лежащее вне того диапазона чисел, которому соответствует данный целый тип? Давайте присвоим некоторой переменной целого типа наибольшее возможное значение, выполним операцию сложения и посмотрим, что произойдет.

```
/* переполнение */
main()
{
int i = 32767;
printf(" %d %d %d\n", i, i+1, i+2);
}
```

Ниже приведен результат работы этой программы, выполненной на нашей вычислительной системе:

```
32767 32768 32767
```

Целая переменная i ведет себя здесь как одометр в машине (прибор для определения пройденного расстояния). Когда его показания достигают максимума, данная величина «сбрасывается», и все начинается сначала. Основное отличие состоит в том, что показания одометра растут, начиная с нуля, а значения нашей переменной типа int — с величины — 32768.

Заметим, что при этом вам не сообщают, что переменная i превысила максимальное значение. Для регистрации подобных событий вы должны использовать свои программные средства.

Описанный подход не вытекает непосредственно из правил языка Си, а является

довольно распространенным способом реализации.

Тип данных unsigned

Обычно данный тип служит модификатором одного из трех ранее описанных типов. Поэтому мы можем использовать комбинации ключевых слов unsigned int или unsigned long как обозначения типов. Для указания типа unsigned int достаточно привести только ключевое слово unsigned. Некоторые вычислительные системы никак не обеспечивают аппаратную реализацию типа unsigned long; кроме того, существуют модели микропроцессоров, в которых unsigned - специальный тип фиксированного размера.

Целые беззнаковые константы записываются точно так же, как и обычные целые константы, с тем лишь исключением, что использование знака — запрещено.

Целые переменные без знака описываются и инициализируются совершенно аналогично тому, как это делается в случае обычных целых переменных. Ниже приведено несколько примеров:

```
unsigned int students;
unsigned players;
unsigned short ribs = 6;
```

Применение данного типа при введении в программу некоторой переменной гарантирует, что она никогда не станет отрицательной. Кроме того, если вы имеете дело только с положительными числами, вы сможете воспользоваться тем, что данные указанного типа могут принимать большие значения, чем данные эквивалентного типа со знаком. Обычно это применяется при адресации памяти и организации счетчиков.

Тип данных char

Этот тип определяет целые числа без знака в диапазоне от 0 до 255. Обычно такое целое размещается в одном байте памяти. В машине используется некоторый код для перевода чисел в символы и обратно. В большинстве компьютеров это код ASCII, описанный в приложении в конце книги. Во многих компьютерах фирмы IBM (но не IBM PC) применяется другой код, называемый EBCDIC. На протяжении всей книги мы будем использовать код ASCII и, проведя различные примеры, будем ссылаться на него.

Описание символьных переменных. Для описания символьной переменной применяется ключевое слово char. Правила, касающиеся описания более чем одной переменной и инициализации переменных, остаются теми же, что и для других основных типов. Поэтому строки, приведенные ниже, являются допустимыми операторами.

```
char response;
char intable, latan;
char isma = 'S';
```

Символьные константы. В языке Си символы заключаются в апострофы. Поэтому, когда мы присваиваем какое-то значение переменной broiled типа char, мы

должны писать

```
broiled = 'T'; /* ПРАВИЛЬНО */
```

а не

```
broiled = T; /* НЕПРАВИЛЬНО */
```

Если апострофы опущены, компилятор «считает», что мы используем переменную с именем T, которую забыли описать.

В стандарте языка Си принято правило, согласно которому значениями переменной или константы типа char могут быть только одиночные символы. В соответствии с этим последовательность операторов, указанная ниже, является недопустимой, поскольку там делается попытка присвоить переменной bovine значение, состоящее из двух символов:

```
char bovine; bovine = 'ox' /* НЕПРАВИЛЬНО */
```

Если вы посмотрите на таблицу кода ASCII, то увидите, что некоторые из «символов» в ней не выводятся на печать. Например, при использовании в программе символа номер 7 терминал компьютера издает звуковой сигнал. Но как использовать символ, который невозможно набрать на клавиатуре? В языке Си для этого имеются два способа.

В первом способе используется сам код ASCII. Вы должны только указать номер символа вместе с предшествующим знаком «обратная косая черта». Мы уже делали это в нашей программе «золотой эквивалент». Вот эта строка

```
beep = ' \007';
```

Здесь имеются два важных момента, которые вы должны отчетливо представлять себе. Первый — это то, что последовательность знаков заключается в апострофы точно так же, как это делается с обычным символом. Второе — то, что номер символа должен быть записан в восьмеричном виде. При записи последовательности знаков мы можем случайно пропустить нули в первых позициях; в этом случае для представления кода «сигнал» мы могли бы использовать '\07' или даже '\7'. Но ни в коем случае не опускайте в записи последние нули! Последовательность символов '\020' можно записать в виде '\20', но не '\02'.

При использовании кода ASCII необходимо отметить различие между числами и символами, обозначающими числа. Например, символу "4" соответствует код ASCII, равный 52. Это символ " 4" , а не число 4.

Во втором способе представления «неудобных» знаков используются специальные последовательности символов. Они называются управляющими последовательностями и выглядят следующим образом:

\n	новая строка
\t	табуляция
\b	шаг назад
\r	возврат каретки
\f	подача бланка
\\	обратная косая черта (\)
\’	апостроф (')

\” кавычки (")

При присваивании символьной переменной эти последовательности тоже должны быть заключены в апострофы. Например, мы могли бы написать оператор

nerf = ' \n';

а затем вывести на печать переменную nerf; это приведет к продвижению на одну строку вперед на печатающем устройстве или на экране дисплея.

Первые пять управляющих последовательностей являются общепринятыми символами, предназначенными для управления работой печатающего устройства: символ «новая строка» вызывает переход к новой строке; символ «табуляция» сдвигает курсор или печатаюшую головку на некоторое фиксированное число позиций 5 или 8; символ «шаг назад» производит сдвиг назад на одну позицию; символ «возврат каретки» осуществляет возврат к началу строки; символ «подача бланка» вызывает протяжку бумаги на одну страницу. В последних трех управляющих последовательностях символы \, ', " можно считать символьными константами [поскольку они служат для определения символьных констант и непосредственно используются в операторе printf(), применение их самих в качестве символов могло бы привести к ошибке]. Если вы хотите вывести на печать строку:

Запомните, “ символ \ называется обратная косая черта”

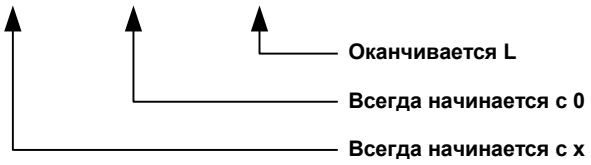
оператор будет выглядеть так:

printf(" Запомните, \" символ \\ называется обратная косая черта. \" \n");

Здесь у вас могут возникнуть два вопроса. Во-первых, почему мы не заключили управляющие последовательности в апострофы? Во-вторых, в каких случаях необходимо использовать код ASCII и когда управляющие последовательности, которые мы только что обсуждали? (Мы надеемся, что у вас возникли как раз эти вопросы, потому что мы собираемся отвечать именно на них.)

1. Когда символ является частью строки символов, заключенной в кавычки, он входит туда без апострофов независимо от того, является ли он управляющим или нет. Заметим, что все остальные символы в нашем примере (3,а, п, о, м, н и т. д.) тоже присутствуют в этой строке без кавычек. Строка символов, заключенная в кавычки, называется символьной строкой или цепочкой. Мы обсудим этот вопрос в следующей главе.
2. Если у вас есть возможность выбора одной из двух форм записи некоторой специальной управляющей последовательности, скажем ' \f', или эквивалентного кода из таблицы кодов ASCII — ' \016', то рекомендуем использовать ' \f'. Во-первых, это более наглядно. Во-вторых, лучше согласуется с требованием переносимости программ, поскольку даже в том случае, когда в системе не используется код ASCII, обозначение ' \f будет продолжать «работать».

Класс целых констант			
Тип	Шест.	Восм.	Десят.
Char	Отсут.	'\034'	Отсут.
Short	± 0x23	± 078	± 092
Unsigned short	0x23	078	92
Long	0x23L	078L	92L



Программа. Ниже приводится короткая программа, позволяющая узнавать номер кода символа даже в том случае, если на вашей машине не используется код ASCII.

```
main() /* определяет номер кода символа */
{
char ch;
printf("Введите, пожалуйста, символ \n");
scanf(" %c",&ch); /* ввод пользователем символа */
printf("Код символа %c равен %d.\n", ch, ch);
}
```

При работе с этой программой не забывайте нажимать клавишу [ввод] или [возврат] после ввода символа. Затем функция scanf() прочтет введенный символ; знак амперсанд (&) указывает, что символ должен быть присвоен переменной ch. Функция printf() выводит на печать величину ch дважды: первый раз как символ (в соответствии со спецификацией %c), а затем как десятичное целое число (в соответствии со спецификацией %d).

Типы данных float и double

В большинстве проектов разработки программного обеспечения оказывается вполне достаточным использовать данные целых типов. Однако в программах вычислительного характера часто применяются числа с плавающей точкой. В языке Си такие данные описываются типом float; они соответствуют типу real в Фортране и Паскале. Указанный подход, как вы могли заметить при внимательном чтении, позволяет представлять числа из гораздо более широкого диапазона, включая и десятичные дроби. Числа с плавающей точкой совершенно аналогичны числам в обычной алгебраической записи, используемой при работе с очень большими или малыми числами. Давайте рассмотрим ее подробнее.

Алгебраическая запись числа представляет собой произведение некоторого десятичного числа на степень, основание которой равно десяти. Ниже приведено несколько примеров.

Число	Алгебраическая	Запись для ввода в

	запись	машину
1 000 000 000	$1.0*10^9$	1,0e9
123 000	$1,23*10^5$	1,23e5
322.56	$3,2256*10^2$	3,2256e2
0.000056	$5,6*10^{-5}$	5,6e-5

В первом столбце числа изображены в обычной записи, во втором приведена соответствующая алгебраическая запись, а в третьем столбце числа показаны в том виде, в котором они обычно представляются при вводе в машину и при выводе из нее — с символом e, за которым следует показатель степени по основанию десять (порядок).

Обычно для размещения в памяти числа с плавающей точкой отводится 32 бита — 8 бит для представления порядка и знака и 24 бита — для мантиссы (т. е. коэффициента при степени десяти). Важным фактом, который вам необходимо знать, является то, что такой способ дает возможность представлять числа с точностью до 6—7 десятичных цифр в диапазоне $\pm(10^{37} — 10^{38})$. Это может оказаться удобным, если вам понадобится обрабатывать числа того же порядка, что масса Солнца (2.0e30 кг) или заряд протона (1.6e—19 Кл). (Многим нравится использовать подобные числа.)

Во многих ЭВМ предусматривается обработка данных типа double (вычислений с двойной точностью), когда для представления чисел используется удвоенное число битов, чаще всего 64. В некоторых машинах все 32 добавочных бита используются для хранения мантиссы. Это увеличивает число значащих цифр и уменьшает ошибку округления. В других машинах некоторое число битов из дополнительного набора используется для хранения большего порядка: это расширяет диапазон представления чисел.

Другой способ определения данных типа double заключается в использовании ключевых слов long float.

Описание переменных с плавающей точкой

Переменные с плавающей точкой описываются и инициализируются точно таким же образом, что и переменные целого типа. Ниже приведено несколько примеров:

```
float noah, jonah;
double trouble;
float planck = 6.63e-34;
```

Константы с плавающей точкой

Правила языка Си допускают несколько способов записи констант с плавающей точкой. Наиболее общая форма записи константы — это последовательность десятичных цифр со знаком, включающая в себя десятичную точку, затем символ e или E и показатель степени по основанию 10 со знаком. Вот два примера:

```
-1.56E+12       2.87e-3
```

Знак + можно не писать. Разрешается также опускать либо десятичную точку, либо экспоненциальную часть, но не одновременно. Кроме того, можно не писать

дробную или целую часть, но не обе сразу. Ниже приведено еще несколько правильно записанных констант с плавающей точкой:

3.14159 .2 4e16 .8E-5 100

Использовать пробелы при записи констант запрещается

1.56E+ 12 — НЕПРАВИЛЬНО

В процессе обработки константы с плавающей точкой рассматриваются в формате с удвоенной точностью. Предположим, например, что переменная some типа float получает свое значение в результате выполнения оператора

some = 4.0*2.0;

В этом случае константы 4.0 и 2.0 размещаются в памяти как данные типа double, т. е. для каждой из них (обычно) отводится 64 бит. Их произведение (равное 8) вычисляется с помощью операции умножения, выполняемой с двойной точностью, и только после этого производится усечение результата до нормального размера, соответствующего типу float. Все это обеспечивает максимальную точность ваших вычислений.

Переполнение и потеря значимости при обработке чисел с плавающей точкой. Что произойдет, если значение переменной типа float выйдет за установленные границы? Например, предположим, что вы умножаете 10e38 на 100 (переполнение) или делите 10e — 37 на 1000 (потеря значимости). Результат целиком зависит от реакции вашей вычислительной системы. В нашей системе при возникновении состояния «переполнение» результат операции заменяется максимально допустимым числом, а при потере значимости — нулем. В других системах в подобной ситуации могут выдаваться предупреждающие сообщения, выполнение задачи можно приостановить, или вам будет предоставлена возможность предпринять что-нибудь самому. Если этот вопрос окажется для вас существенным, вам необходимо будет свериться с правилами, действующими для вашей ЭВМ. В случае если вы не сможете найти никакой информации, не бойтесь пробовать другие возможности.

Резюме: основные типы данных.

Ключевые слова. Данные основных типов вводятся в программу при помощи следующих семи ключевых слов: int, long, short, unsigned, char, float, double.

Целые числа со знаком. Данные этих типов могут принимать положительные и отрицательные значения.

int: основной целый тип, используемый в вычислительной системе;

long или long int: может содержать целое значение, не меньшее максимальной величины, допускаемой типом int, или даже большее;

short или short int: максимальное целое число типа short не больше, чем максимальное целое число типа int, а может быть, и меньше.

Обычно числа типа long бывают больше чисел типа short, а тип int реализуется как один из двух указанных типов. Например, компилятор Lattice C на IBM PC под данные типов short и int отводит 16 бит, а под данные типа long — 32 бита. Все зависит от конкретной системы.

Целые числа без знака. Данные этих типов принимают только положительные значения или нуль. Это расширяет диапазон возможных положительных значений. При указании типа используйте ключевое слово unsigned: unsigned int, unsigned long, unsigned short. Просто unsigned соответствует написанию unsigned int.

Символы. Эти знаки соответствуют типографским символам, таким, как A, &, + и т.д. Обычно под каждый символ отводится 1 байт памяти.

char: ключевое слово, используемое для указания данных этого типа.

Числа с плавающий точкой. Данные этих типов могут принимать положительные и отрицательные значения. float: основной тип данных с плавающей точкой в системе; double или long float при размещении в памяти чисел с плавающей точкой такого типа отводится (воз. можно) элемент памяти большего размера; при этом может допускаться либо боль. шее число значащих цифр, либо большее значение порядка.

Резюме: как описывать простые переменные

1. Выбрать требуемый тип данных.
2. Выбрать имя для переменной.
3. Для оператора описания использовать нижеследующий формат: спецификация-типа имя-переменной; Спецификация-типа формируется из одного или более ключевых слов. Вот несколько примеров:
   int erest;
   unsigned short cash;

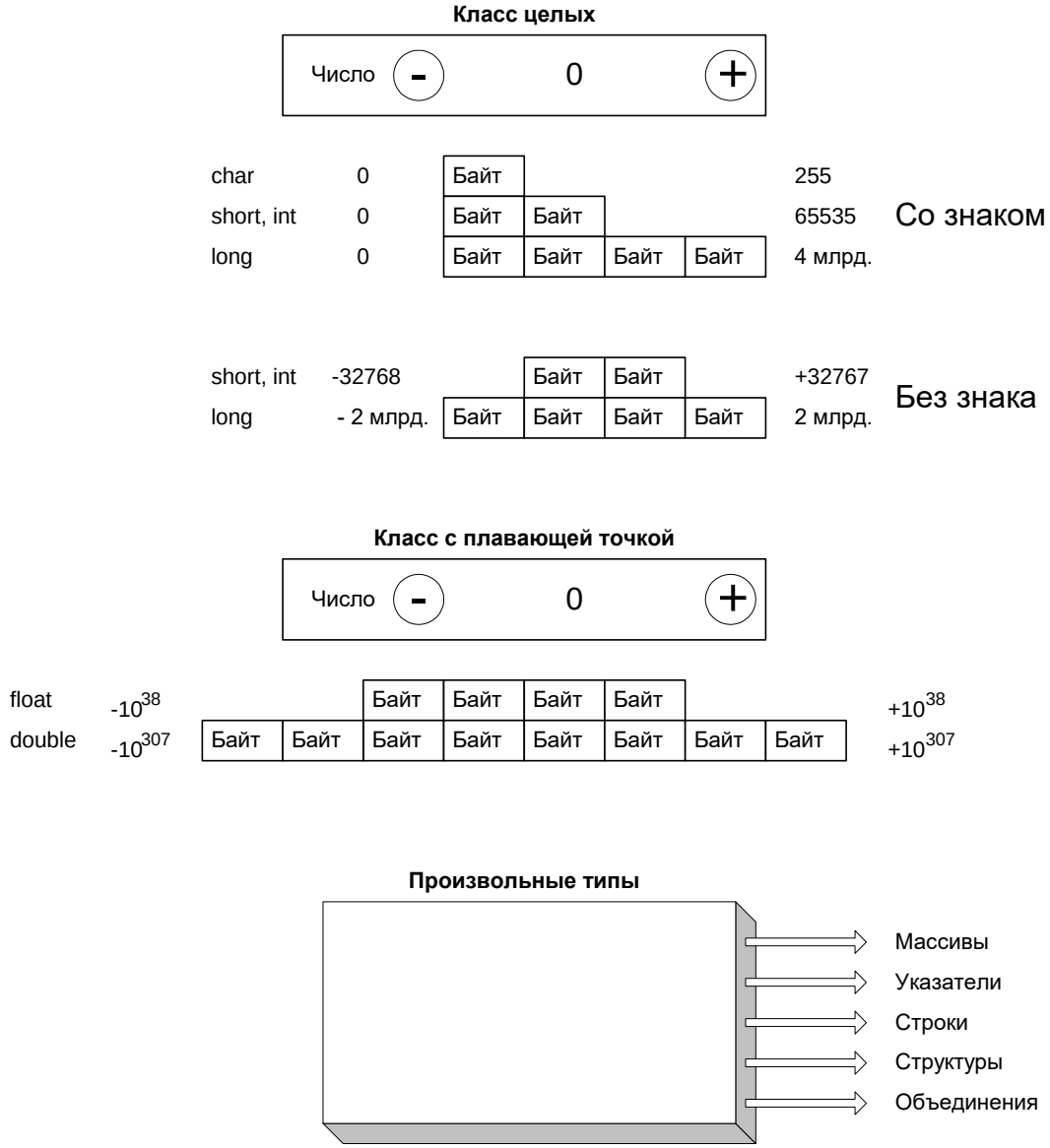
4. Вы можете описать в одном операторе несколько переменных одного типа, разделяя их имена запятыми:
   char ch, unit, ans;

5. В операторе описания вы имеете возможность инициализировать переменную: float mass = 6.0E24;

Другие типы и размеры данных

Этот раздел завершает рассмотрение основных типов данных. Некоторым читателям их число может показаться слишком большим. Остальные могут полагать, что описанных типов недостаточно; например, им захочется иметь булев тип или строковый,тип данных. В языке Си они отсутствуют, но, несмотря на это, он вполне подходит для написания программ, связанных с обработкой логических данных или строк. Самые простые возможности работы со строками мы рассмотрим в следующей главе.

В языке Си имеются и другие типы данных, построенные с использованием основных типов. Они включают в себя массивы, указатели, структуры и объединения. Хотя эти типы являются предметом рассмотрения последующих глав, мы, не подозревая об этом, уже применили указатели в примерах, приведенных в данной главе. [Указатели используются функцией scanf(); признаком этого в данном случае служит префикс &.]



Размеры данных. Приведем таблицу размеров данных для некоторых распространенных вычислительных систем.

Размер слова	**DEC PDP-11**	**DEC VAX**	**Interdata**	**IBM PC**
	16 бит	32 бита	32 бита	16 бит
char	8	8	8	8
int	16	32	32	16
short	16	16	16	16
long	32	32	32	32
float	32	32	32	32
double	64	64	64	64
Диапазон порядка(double)	±38	±38	±76	-307 - 308

Как обстоит дело на вашей машине? Попробуйте выполнить нижеследующую

программу:

```
/* X05.C */
#include <stdio.h>
void main(void) /* определение размера для данных */
{
    int count;
    printf("Данные типа int занимают %d байта.\n",sizeof(int));
    printf("Данные типа char занимают %d байта.\n",sizeof(char));
    printf("Данные типа long занимают %d байта.\n",sizeof(long));
    printf("Данные типа short занимают %d байта.\n",sizeof(short));
    printf("Данные типа unsigned занимают %d байта.\n",sizeof(unsigned));
    printf("Данные типа double занимают %d байта.\n",sizeof(double));
    printf("Данные типа float занимают %d байта.\n",sizeof(float));
}
```

В языке Си имеется встроенная операция sizeof, которая позволяет определить размер объектов в байтах. Результат работы этой программы на нашей системе выглядит так:

```
Данные типа int занимают 2 байта.
Данные типа char занимают 1 байт.
Данные типа long занимают 4 байта.
Данные типа double занимают 8 байт.
```

Мы определили размеры данных только четырех типов, но вы легко можете модифицировать эту программу и найти размер объекта любого другого интересующего вас типа.

Использование типов данных. Во время разработки программы вам необходимо составить список требуемых переменных и указать при этом, какого они должны быть типа. Скорее всего вы будете использовать тип int или, возможно, float для определения чисел и тип char для символов. Описывайте эти данные в самом начале тела функции, в которой они используются. Имена переменных выбирайте таким образом, чтобы они указывали на их смысл. При инициализации переменной следите за тем, чтобы тип константы соответствовал типу переменной.

```
int apples = 3; /* ПРАВИЛЬНО V
int oranges = 3.0; /* НЕПРАВИЛЬНО */
```

Язык Си «рассматривает» такие несоответствия менее жестко, чем, скажем, Паскаль, но в любом случае лучше учиться избегать дурных привычек.

Что вы должны были узнать. Здесь мы рассмотрели довольно большой материал. Суммируя его, мы обратим основное внимание на практическую сторону тех вопросов, которые здесь обсудили. Так же как и раньше, мы дадим краткие примеры. Ниже приводится сводка тех фактов, которые вы должны были узнать.

Что такое основные типы данных языка Си: int, short, long, unsigned, char, float, double.

Как описать переменную любого типа: int beancount, float root-beer; и т. д.

Как записать константу типа int: 256, 023, OXF5 и т. д.
Как записать константу типа char: 'r', 'U, '\007', '?' и т. д.
Как записать константу типа float: 14,92, 1,67e —27 и т. д.
Что такое слова байты и биты.
В каких случаях используются различные типы данных.

Вопросы и ответы.

Рассмотрение приводимых ниже вопросов должно помочь вам глубже усвоить материал.

Вопросы

1. Какими типами вы будете пользоваться при обработке данных -следующего вида:
   а. Население Рио-Фрито.
   б. Средний вес картины Рембрандта.
   в. Наиболее часто встречающаяся буква в тексте.
   г. Сколько раз указанная буква встречается в тексте.
2. Определите тип и смысл (если он есть) каждой из следующих констант:
   а. '\b'
   б. 1066
   в. 99.44
   г. OXAA
   д. 2.0e30
3. Вирджила Анн Ксенопод (ВАКС) написала программу с множеством ошибок. Помогите ей обнаружить их.

```
#include <stdio.h>
main
    (
        float g; h;
        float tax, rate;

            g = e21;
            tax = rate*g;
    )
```

Ответы

1. а. int, возможно short; население выражается целым числом
   б. float; маловероятно, что среднее окажется целым числом
   в. char
   г. int, возможно unsigned
2. а. char, символ «шаг назад»
   б. int, историческая дата2'
   в. float, степень чистоты после мытья
   г. Шестнадцатеричное число типа int; десятичное значение 170
   д. float; масса Солнца в кг
3. Строка 1: правильная
   Строка 2 должна содержать пару круглых скобок вслед за именем main, т. е. main()
   Строка 3: нужно использовать { , а не (
   Строка 4: g и h должны разделяться запятой, а не точкой с запятой

Строка 5: правильная
Строка 6: (пустая) правильная
Строка 7: перед e должна стоять по крайней мере одна цифра
Например: 1e21 или 1.0e21
Строка 8: правильная
Строка 9: нужно использовать }, а не)

Недостающие строки: первая — переменной rate нигде не присваивается значение. Вторая — переменная h нигде не используется. Кроме того, программа никак не информирует нас о результатах вычислений. Ни одна из этих ошибок не будет служить препятствием для выполнения программы (хотя при компиляции может быть выдано предупреждение о неиспользуемой переменной), но все ошибки существенно снижают эффективность программы и без того уже ограниченную.

3. Символьные строки, директива #define, функции printf() и scanf()

Здесь мы продолжим работу с данными: покопаемся в вопросах, выходящих за пределы тех, которые были связаны с типами данных, и рассмотрим символьную строку. Сначала опишем важное средство языка — препроцессор Си — и узнаем, как задавать и использовать символические константы. Затем вновь обсудим способы ввода и вывода данных, при этом более полно исследуем возможности функций printf() и scanf(). Ну, а теперь вы, вероятно, ожидаете примера программы, который должен быть помещен в начале; мы не будем вас разочаровывать и приведем его.

```
/* непринужденный разговор */
#define DENSITY 62.4 /* плотность тела человека в фунтах на кубический фут */
main() /* любопытствующая программа*/
{
    float weight, volume;
    int size, letters;
    char name[40]; /* или попробуйте "static char name [40];" */
    printf(" Привет! Как вас зовут?\n" );
    scanf("%s" , name);
    printf("%s, Каков ваш вес в фунтах?\n", name);
    scanf("%f", &weight);
    size = sizeof name;
    letters = strlen (name);
    volume = weight/DENSITY;
    printf(" Прекрасно, %s, ваш объем %2.2f кубических фута.\n", name, volume);
    printf("Кроме того, ваше имя состоит из %d букв,\n", letters);
    printf("и для его размещения в памяти у нас есть %d байт,\n", size);
}
```

Результат работы программы «непринужденный разговор» может, например, выглядеть следующим образом:

Привет! Как вас зовут? Анжелика
Анжелика, каков ваш вес в фунтах?
102,5
Прекрасно, Анжелика, ваш объем 1,64 кубических фута
Кроме того, ваше имя состоит из 8 букв, и для его размещения в памяти у нас есть 40 байт.

Перечислим основные новые черты этой программы.

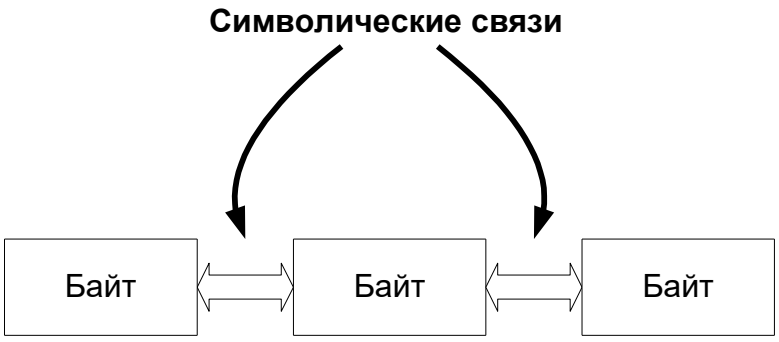
9. Мы использовали «массив» для хранения «символьной строки» — в данном случае для некоторого имени.
10. При вводе и выводе строки была использована «спецификация преобразования» %s.
11. Для определения символической константы DENSITY был использован препроцессор языка Си.
12. Для нахождения длины строки была использована функция strlen().

Способ ввода-вывода, реализованный в языке Си, может показаться вначале несколько более сложным по сравнению с вводом-выводом, предусмотренным, например, в Бейсике. Однако эта сложность окупается улучшенными возможностями управления вводом-выводом и большей эффективностью получаемых программ. Указанная трудность — не единственная, с которой вы столкнетесь в дальнейшем.
Давайте исследуем все новые моменты более детально.

3.1. Символьные строки

«Символьная строка» — это последовательность, состоящая из одного или более символов. В качестве примера рассмотрим следующую строку:
"Строки изливались прямо из сердца!"

Кавычки не являются частью строки. Они вводятся только для того, чтобы отметить ее начало и конец, т. е. играют ту же роль, что и апострофы в случае одиночного символа. В языке Си нет специального типа, который можно было бы использовать для описания строк. Вместо этого строки представляются в виде «массива» элементов типа char. Это означает что символы в строке можно представить себе расположенными в соседних ячейках памяти - по одному символу в ячейке (Смотри рисунок).



Описание char name[3] объединяет вместе три объекта типа char

Необходимо отметить, что на рисунке последним элементом массива является символ \0. Это «нуль-символ», и в языке Си он используется для того, чтобы отмечать конец строки. Нуль-символ — не цифра 0; он не выводится на печать и в таблице кода ASCII (American Standard Code for Information Interchange — Американский стандартный код для обмена информацией) имеет номер 0. Наличие нуль-символа означает, что количество ячеек массива должно быть по крайней мере на одну больше, чем число символов, которые необходимо размещать в памяти.
Ну, а теперь спросим, что такое массив? Массив можно представить себе как совокупность нескольких ячеек памяти, объединенных в одну строку. Если вы предпочитаете более формальные и строгие определения, то массив — это упорядоченная последовательность элементов данных одного типа. В нашем примере мы создали массив из 40 ячеек памяти, в каждую из которых можно поместить один элемент типа char. Мы осуществили это с помощью оператора описания:

```
char name [40];
```

Квадратные скобки указывают, что переменная name — массив, 40 — число его элементов, а char задает тип каждого элемента.
(В комментариях к программе было отмечено, что при желании вы можете воспользоваться более сложным оператором описания):

```
static char name [40];
```

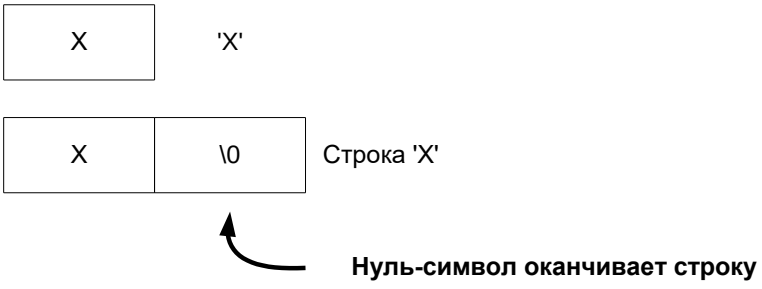
Ввиду некоторой специфики, связанной с реализацией функции scanf() в нашей системе, мы вынуждены использовать эту вторую форму, но весьма вероятно, что вы сможете выбрать любую из них. Если обнаружится, что при работе с первой формой оператора описания у вас возникнут трудности при решении наших примеров, попробуйте воспользоваться второй. В действительности вторая форма должна работать в любой системе, но мы не хотим применять тип Static до тех пор, пока не рассмотрим понятие классов памяти.)
На первый взгляд все это выглядит довольно сложным: вы должны создать массив, расположить символы в виде строки и не забыть добавить в конце \0. К счастью, о большинстве деталей компилятор может «позаботиться» сам.
Попробуйте выполнить приведенную ниже программу, чтобы посмотреть, как просто все происходит на практике:

```
/* похвала 1*/
#define PRAISE " Вот эта да, какое великолепное имя!"
main()
{
    char name [50];
    printf(" Как вас зовут?\n" );
    scanf(" %s", name);
    printf(" Привет, %s. %s\n" , name, PRAISE);
}
```

Символ %s служит указанием функции printf() напечатать строку. Результат выполнения программы похвала 1 может выглядеть, например, так:

```
Как вас зовут?
Элмо Бланк.
Привет, Элмо. Вот это да, какое великолепное имя!
```

Как видите, нам не пришлось самим помещать нуль-символ в конец массива. Эта задача была выполнена за нас функцией scanf() при чтении вводимой строки. PRAISE — «символическая строковая константа». Ниже мы рассмотрим директиву #define более подробно, а пока вы должны знать, что кавычки, в которые заключена фраза, следующая за строковой константой PRAISE, идентифицируют эту фразу как строку, и поэтому в ее конец будет помещен нуль-символ.
Заметим (и это очень важно), что функция scanf() при вводе строки «Элмо Бланк» читает только имя Элмо. Дело в том, что, встретив какой-нибудь разделитель (пробел, символ табуляции или перевода строки), функция scanf() прекращает ввод символов, т. е. в данном случае она прекращает опрос переменной name в тот момент, когда доходит до пробела между «Элмо» и «Бланк». Вообще говоря, функция scanf() вводит только одиночные слова, а не целую фразу в качестве строки. Для чтения входной информации в языке Си имеются другие функции, например функция gets(), предназначенная для обработки строк общего вида. Более полно работу со строками мы рассмотрим позднее.
Необходимо заметить также, что строка "x" не то же самое, что символ 'x'. Первое различие: 'x' — объект одного из основных типов (char), в то время как "x" — объект производного типа (массива элементов типа char). Второе различие: "x" на самом деле состоит из двух символов — символа 'x' и нуль-символа.



3.2. Длина строки — функция strlen()

Раньше мы практически без объяснений использовали операцию sizeof, которая дает нам размер объектов в байтах Функция strten() позволяет определять длину строки числом символов. Поскольку для размещения одного символа в памяти отводится 1 байт, можно было бы предположить, что в результате применения любой из этих двух операций к одной строке будет получен одинаковый результат. Оказывается, это не так. Давайте не-много изменим нашу предыдущую программу (добавим к ней несколько строк), и тогда мы поймем, в чем дело.

```
/* похвала 2*/
#define PRAISE " Вот это да, какое великолепное имя!"
main()
{
    char name [50];
    printf(" Как вас зовут?\n" );
    scanf(" %s" , name);
    printf(" Привет, %s. %s\n" , name, PRAISE);
    printf(" Ваше имя состоит из %d букв и занимает %d ячеек памяти. \n",
    strlen (name), sizeof name);
    printf(" Хвалебная фраза состоит из %d букв" , strlen (PRAISE));
    printf(" и занимает %d ячеек памяти. \n" , sizeof PRAISE);
}
```

Заметим, что случайно мы воспользовались двумя методами для обработки длинных операторов printf(). В первом случае мы записали один оператор печати в двух строках программы (строкой программы считается строка до запятой).Мы сделали это, поскольку разрешается разбивать строку между аргументами, но не посередине строки. В другом случае использовались два оператора printf() для печати одной строки; мы указали символ «новая строка» (\ n) только во втором из них. Представленный ниже результат работы данной программы поможет понять подобную ситуацию:

```
Как вас зовут?
Перки.
Привет, Перки. Вот это да, какое великолепное имя!
Ваше имя состоит 5 букв и занимает 50 ячеек памяти.
Хвалебная фраза состоит из 35 букв и занимает 36 ячеек памяти.
```

Давайте посмотрим, в чем дело. Массив name занимает 50 ячеек памяти, и именно об этом сообщает операция sizeof. Но для хранения имени Перки требуются только первые пять ячеек, и как раз об этом нас информирует функция strlen(). В шестой ячейке массива name содержится нуль-символ, и его появление служит сигналом для функции strlen() прекратить подсчет символов.

При переходе к обработке константы PRAISE обнаруживается, что функция strlen() опять дает нам точное число символов (включая пробелы и знаки пунктуации) в строке. Результат операции sizeof оказывается на единицу большим, поскольку при этом учитывается и «невидимый» нуль-символ, помещенный в конец строки. Мы не указываем компилятору, какой объем памяти он должен отвести для размещения всей фразы; он сам подсчитывает число символов между кавычками.
Еще одно замечание: мы использовали операцию sizeof со скобками, а здесь — без них. Решение, использовать ли скобки или нет, зависит от того, что вы хотите знать: объем памяти, отводимый под элементы конкретного типа, или объем памяти, занимаемый определенным объектом. В первом случае вы писали бы sizeof(char) или sizeof(float), а во втором — sizeof name или sizeof 6.28.
В данном разделе операции strlen() и sizeof использовались только для удовлетворения нашего любопытства; в действительности они представляют собой важные программные средства. Функция strlen(), например, полезна в любого сорта программах обработки строк или символов, в чем вы сможете убедиться позднее.
Приступим теперь к рассмотрению директивы #define.

3.3. Константы и препроцессор языка Си

Иногда возникает необходимость использовать в программах константы. Например, оператор, позволяющий определять длину окружности, можно было бы записать в следующем виде:

```
circ = 3.14 * diameter;
```

Приведенная здесь константа 3.14 — известное число пи. Чтобы ввести ту или иную константу в программу, нужно указать ее фактическое значение, как было сделано выше. Однако существуют веские причины использовать вместо этого «символические константы»; например, мы могли бы применять оператор:

```
circ = pi * diameter;
```

а позже компилятор подставил бы в него фактическое значение константы.
В чем достоинства такого метода? Во-первых, имя говорит нам больше, чем число. Сравним два оператора

```
owed = 0.015 * housevl;
owed = taxrate * housevl;
```

Если мы изучаем большую программу, то второй вариант будет нам более понятен.
Во-вторых, предположим, что некоторая константа использовалась в нескольких местах программы и впоследствии возникла необходимость изменить ее значение — ведь в конце концов и налоговые тарифы (taxrate) меняются, и, к примеру, некое законодательное собрание приняло однажды закон впредь считать число пи равным 3 1/7. (Весьма вероятно, что окружности пришлось при этом скрываться от правосудия!) В таком случае требуется только изменить определение символической константы, а не отыскивать каждый случай ее появления в программе.
Теперь осталось выяснить, как можно создать такую символическую константу? Первый способ заключается в том, чтобы описать некоторую переменную и положить ее равной требуемой константе. Мы могли бы сделать это следующим образом:

```
float taxrate;
taxrate = 0.015;
```

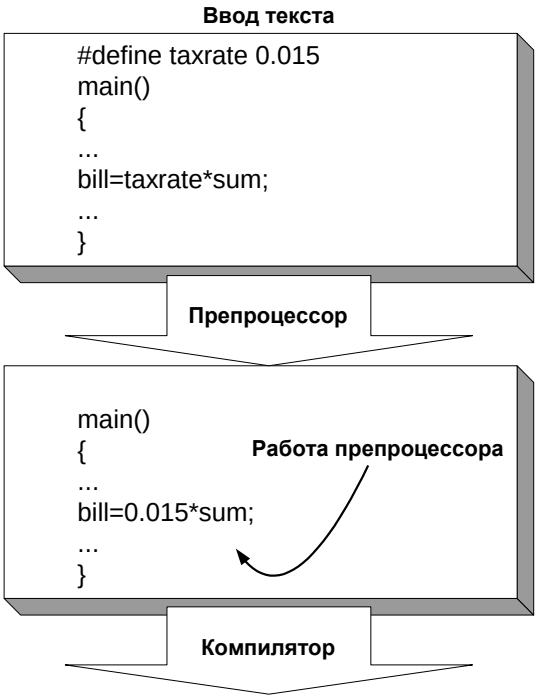
Такой способ подходит для небольшой программы, в других же случаях он несколько неэкономичен, поскольку каждый раз при использовании переменной taxrate компьютер должен будет обращаться к той ячейке памяти, которая отведена данной переменной. Это служит примером подстановки «во время выполнения», так как она производится именно при выполнении программы. К счастью, в языке Си имеется и другой, лучший способ.
Этот способ реализуется с помощью препроцессора языка Си. Мы уже видели, как препроцессор использует директиву #include для включения информации из другого файла в программу. Кроме того, препроцессор дает нам возможность задавать константы. Для этого в начало файла, содержащего вашу программу, необходимо добавить только одну строку, аналогичную следующей:

```
#defne TAXRATE 0.015.
```

При компиляции программы каждый раз, когда появится переменная TAXRATE, она будет заменяться величиной 0.015. Это называется подстановкой «во время компиляции». К тому моменту, когда вы начнете выполнение своей программы, все подстановки будут уже сделаны.
Несколько замечаний по поводу формата. Сначала идет ключевое слово #define. Оно должно начинаться с самой левой позиции. Потом следует символическое имя константы, а затем ее величина. Символ «точка с запятой» не используется, поскольку это не оператор языка Си. Почему имя TAXRATE пишется прописными буквами? В процессе использования языка Си выработалась традиция писать константы прописными буквами. Если при просмотре программы вам встретится имя, написанное прописными буквами, вы сразу поймете, что имеете дело с константой, а не с переменной. Это еще один способ улучшить читаемость программы. Ваша программа будет работать даже и тогда, когда вы будете писать константы строчными буквами, но при этом вы должны чувствовать свою вину, поскольку нарушаете традицию. Приведем простой пример:

```
/* пицца */
#define PI 3.14159
main()/* изучение вашей пиццы */
```

```
{
float area, circum, radius;
printf("Чему равен радиус вашей пиццы? \n");
scanf("%f", &radius);
area = PI * radius * radius;
printf(" Основные параметры вашей пиццы следующие: \n");
printf(" длина окружности = %1.2f, площадь =%1.2f\n", circum, area);
}
```



Использование спецификации %1.2f в операторе printf() приведет к тому, что при печати результаты будут округлены до двух десятичных цифр. Мы понимаем, конечно, что написанная выше программа может и не отражать ваши собственные вкусы, касающиеся пиццы, но во множестве программ, посвященных этому вопросу, она займет свое скромное место. Вот один из примеров ее выполнения.

```
Чему равен радикс вашей пицы?
6.0
Основные параметры вашей пиццы следующие:
длина окружности - 37.70, площадь = 113.10
```

Директиву #define можно также использовать для определения символьных и строковых констант. Необходимо использовать знак «апостроф» в первом случае и кавычки — во втором. Примеры, приведенные ниже, вполне правомерны:

```
#define BEEP '\007'
#define ESS 'S'

/* X091.C */
#include "E:\LECTURE\AlgorithmProgramming\Tests\X09.h"
```

```
#include <stdio.h>
void main(void) /* Определение константы во внешнем файле x09.h */
{
    float r;
    double cir;
    printf(" Чему равен радиус r ? ");
    scanf("%f",&r);
    cir = 2*PI*r;
    printf(" Длина окружности с r= %1.2f равна- %1.3f\n",r,cir);
}

/* E:\LECTURE\AlgorithmProgramming\Tests\X09.h */
#define PI 3.14159

#define NULL '\0'
#define OOPS " Ну вот, вы и сделали это!"
```

А теперь мы хотим обрадовать лентяев. Предположим, вы разрабатываете целый пакет программ, использующих один и тот же набор констант. Вы можете произвести следующие действия:

1. Соберите все ваши директивы #define в один файл и назовите его, например, const.h.
2. В начало каждого файла, содержащего программу, включите директиву #include "const.h."

Тогда, если вы будете выполнять программу, препроцессор прочтет файл с именем const.h и использует все директивы #define вашей программы. Получилось так, что символ .h в конце имени файла напомнит вам, что этот файл является «заголовком», т. е. в нем содержится вся информация, которая должна попасть в начало вашей программы. Самому препроцессору безразлично, используете ли вы символ .h в имени файла или нет.

Язык Си — искусный фокусник: создание псевдоимен. Возможности директивы #define не исчерпываются только символическим представлением констант. Рассмотрим, например, следующую программу:

```
#include "alias.h"
program begin
whole yours, mine then
spitout(" Введите, пожалуйста, целое число.\n") then
takein(" %d" , &yours) then
mine=yours times TWO then
spitout(" %d в два раза больше вашего числа!\n" , mine) then
end
```

Странно, текст что-то смутно напоминает, язык немного похож на Паскаль, но программа не похожа на Си-программу. Секрет лежит, конечно, в файле с именем alias.h. Давайте посмотрим, что в нем содержится?

alias.h.

```
#define program main()
#define begin {
#define end }
#define then ;
#define takein scanf
#define spitout printf
#define TWO 2
#define times *
#define whole int
```

Этот пример иллюстрирует, как работает препроцессор. Он просматривает вашу программу и проводит поиск элементов, определяемых директивами #define. Обнаружив такие элементы, он полностью заменяет их. В нашем примере во время компиляции все слова then заменяются символами «точка с запятой», end — } и т. д. Результирующая программа будет полностью идентична той, которую мы могли бы получить, если бы с самого начала писали ее в обычных терминах языка Си.

Эту мощную возможность языка можно использовать для задания макрокоманд, являющихся одним из вспомогательных средств программирования. Мы вернемся к обсуждению этой темы позднее.

Теперь необходимо упомянуть о некоторых ограничениях. Например, части программы, заключенные в кавычки, закрыты для подстановок. Операторы, приводимые ниже, служат иллюстрацией такого положения:

```
#define MN "минимифидианизм"
printf("ОН глубоко верил в MN.\n");
```

Распечатка будет выглядеть так:
Он глубоко верил в MN.

Однако после выполнения оператора
```
printf(" Он глубоко верил в %s.\n" , MN);
```

мы получим следующий результат:
Он глубоко верил в минимифидианизм

В последнем случае константа с именем MN находилась вне кавычек и поэтому была заменена соответствующим значением.

Препроцессор языка Си является полезным вспомогательным средством, поэтому при написании программ старайтесь пользоваться преимуществами, которые он предоставляет. По мере изложения мы покажем вам дополнительные возможности применения препроцессора.

3.4. Функции printf() и scanf()

Функции printf() и scanf() дают нам возможность взаимодействовать с программой. Мы называем их функциями ввода-вывода. Это не единственные функции, которыми мы можем воспользоваться для ввода и вывода данных с помощью программ на языке Си, но они наиболее универсальны. Указанные функции не входят в описание языка Си. И

действительно, при работе с языком Си реализация функций ввода-вывода возлагается на создателей компилятора; это дает возможность более эффективно организовать ввод-вывод на конкретных машинах. Однако в интересах обеспечения совместимости различные системы имеют дело с некоторыми вариантами функций scanf() и printf(). Все, о чем мы здесь говорим, должно быть в основном справедливо для большинства систем, но не удивляйтесь, если обнаружите некоторые отличия в имеющейся у вас версии.

Обычно функции printf() и scanf() «работают» во многом одинаково — каждая использует «управляющую строку» и список «аргументов». Сначала мы рассмотрим работу функции printf(), затем scanf().

Инструкции, передаваемые функции printf(), когда мы «просим» ее напечатать некоторую переменную, зависят от того, какого типа эта переменная. Например, при выводе на печать целого числа применяется формат %d, а при выводе символа — %c. Ниже перечислены все форматы, указываемые при обращениях к функции printf(), а затем показано, как они используются. Каждому формату соответствует тип выводимой (с их помощью) информации, причем первые пять покрывают большинство возникающих потребностей, а остальные четыре применяются достаточно редко.

Формат	Тип выводимой информации
%d	Десятичное целое число
%c	Один символ
%s	Строка символов
%e	Число с плавающей точкой, экспоненциальная запись
%f	Число с плавающей точкой, десятичная запись
%g	Используется вместо записей %f или %e, если он короче
%u	Десятичное целое число без знака
%o	Восьмеричное целое число без знака
%x	Шестнадцатеричное целое число без знака

Посмотрим теперь, как эти форматы применяются.

Использование функции printf()

Приведем программу, иллюстрирующую обсуждаемые вопросы.

```
/* печать чепухи*/
#define PI 3.14159
main()
{
int number = 5;
float ouzo = 13.5;
int cost = 31000;
printf("%d женщин выпили %f стаканов ликера.\n", number, ouzo);
printf("Значение числа pi равно %f.\n", PI);
printf("Прощай! Твое искусство слишком дорого для меня. \n");
printf("%c%d\n", '$', cost);
}
```

Результат выглядит так:

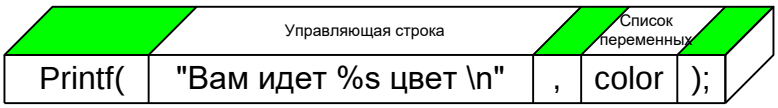
5 женщин выпили 13.500000 стаканов ликера

Значение числа pi равно 3.14159
Прощай! Твое искусство слишком дорого для меня.
$31000

Формат, указываемый при обращении к функции printf(), выглядит следующим образом:
printf(Управляющая строка, аргумент1, аргумент2, ...,);

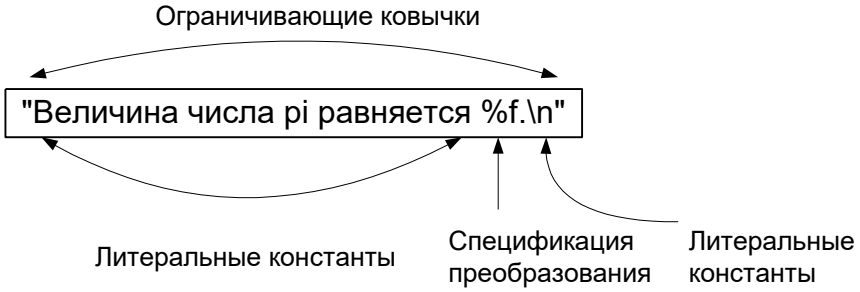
Аргумент1, Аргумент2 и т. д. — это печатаемые параметры, которые могут быть переменными, константами или даже выражениями, вычисляемыми вначале, перед выводом на печать.
Управляющая строка — строка символов, показывающая, как должны быть напечатаны параметры. Например, в операторе
printf("%d женщин выпили %f стаканов ликера\n", number, ouzo);

управляющей строкой служит фраза в кавычках (учитывая предыдущие замечания, это — строка символов), а number и ouzo — аргументы или в данном случае значения двух переменных.



Приведем еще пример.
printf("Значение числа pi равно %f.\n", PI);

На этот раз список аргументов содержит только один элемент — символическую константу PI.
Мы видим, что в управляющей строке содержится информация двух различных видов:
1. Символы, печатаемые текстуально.
2. Идентификаторы данных, называемые также «спецификациями преобразования».



Каждому аргументу из списка, следующего за управляющей строкой, должна соответствовать одна спецификация преобразования. Горе вам, если вы забудете это основное требование! Никогда не пишите, например, так:

printf(" Количество слизняков %d, червяков %d.\n" , score1);

Здесь отсутствует аргумент для второй спецификации преобразования %d. Способ

проявления этой ошибки целиком зависит от вашей вычислительной системы, но в лучшем случае вы получите бессмыслицу.
Если вам нужно напечатать какую-нибудь фразу, то нет необходимости использовать спецификацию преобразования; если же требуется только вывести данные на печать, то можно обойтись и без использования комментария. Поэтому каждый из операторов, приведенных ниже, вполне приемлем.

```
printf("Прощай! Твое искусство слишком дорого для меня\n");
printf(" %c%d\n", '$', cost);
```

Заметим, что во втором примере первый аргумент из печатаемого списка является символьной константой, а не переменной.
Поскольку символ % используется в функции printf() для идентифицирования спецификаций преобразования, возникает небольшая проблема в том случае, если вам нужно напечатать сам символ %. Если просто написать один знак %, то компилятор примет его за ошибочную спецификацию преобразования. Выходом из создавшейся ситуации служит довольно простое решение — писать два символа % подряд:

```
pc = 2*6;
printf("Только %d%% стряпни Салли было съедобно.\n", pc);
```

Результат работы программы будет выглядеть следующим образом:
Точько 12% стряпни Салли было съедобно.

Модификаторы спецификации преобразования, используемые в функции printf()
Мы можем несколько расширить основное определение спецификации преобразования, поместив модификаторы между знаком % и символом, определяющим тип преобразования. В приводимой ниже таблице дан список тех символов, которые вы имеете право туда поместить. При использовании одновременно нескольких модификаторов они должны быть указаны в том порядке, в котором перечислены в таблице. Заметим, что при этом допускаются не все комбинации.

Модификатор	**Значение**
—	Аргумент будет печататься с левой позиции поля заданной ширины (как объяснено ниже). Обычно печать аргумента оканчивается в самой правой позиции поля Пример: %—10d
строка цифр	Задает минимальную ширину поля. Большее поле будет использоваться, если печатаемое число или строка не помещаются в исходном поле Пример: %4d
строка цифр	Определяет точность: для типов данных с плавающей точкой — число печатаемых цифр справа от десятичной точки; для символьных строк — максимальное число печатаемых символов Пример: %4.2f (две десятичные цифры для поля шириной в четыре символа)
l	Соответствующий элемент данных имеет тип long, а не int. Пример: %ld

Примеры
Посмотрим, как эти модификаторы работают. Начнем с того, что продемонстрируем влияние модификатора ширины поля на печать целого числа. Рассмотрим следующую программу:

```
main()
{
pnntf("/%d/\n", 336);
pnntf("/%2d/\n", 336);
prmtf("/%10d/\n", 336);
pnntf("/%-10d/\n", 336);
}
```

Эта программа печатает одно и то же значение четыре раза, но использует при этом четыре различные спецификации преобразованиая. Мы вводим также символы /, чтобы вы могли видеть, где на-чинается и кончается каждое поле. Результат выполнения програм-мы выглядит следующим образом:

```
/336/
/336/
/ 336/
/336 /
```

Первая спецификация преобразования %d не содержит модификаторов. Мы видим, что поле печати здесь имеет ширину, равную количеству цифр данного целого числа. Это так называемый выбор «по умолчанию», т. е. результат действия компилятора в случае, если вы не дали ему никаких дополнительных инструкций. Вторая спецификация преобразования — %2d. Она указывает, что ширина поля должна равняться 2, но, поскольку число состоит из трех цифр, поле автоматически расширяется до необходимого размера. Следующая спецификация %10d показывает, что ширина поля равна 10. И действительно, между символами / имеется семь пробелов и три цифры, причем число сдвинуто к правому краю поля. Последняя спецификация %— 10d также указывает ширину поля, равную 10, а знак — приводит к сдвигу всего числа к левому краю, как показано в приведенном выше примере. Когда вы привыкнете к этой системе обозначений, она покажется вам простой и вы сумеете по вашему усмотрению менять вид выходной информации.
Рассмотрим теперь некоторые форматы, соответствующие данным с плавающей точкой. Допустим, у нас имеется следующая программа:

```
main()
{
printf(" /%f/\n", 1234.56);
printf(" /%e/\n", 1234.56);
prmtf(" /%4.2f/\n", 1234.56);
printf(" /%3.1f/\n", 1234.56);
printf(" /%10.3f/\n" , 1234.56);
printf(" /%10.3e/\n" , 1234.56);
```

```
}
```

На этот раз результат работы программы будет выглядеть так:

```
/1234.560059/
/1.234560E+03/
/1234.56/
/1234.6/
/ 1234.560/
/ 1.234E+03/
```

Мы снова начинаем с варианта, выбранного по умолчанию, т. е. со спецификации %f. В этом случае имеется две величины, значения которых используются по умолчанию: ширина поля и число цифр справа от десятичной точки. Вторая величина задает шесть цифр, а ширина поля берется такой, чтобы в нем могло поместиться число. Заметим, что печатаемое число несколько отличается от исходного. Это происходит потому, что на печать выводится 10 цифр, в то время как числа с плавающей точкой в нашей системе изображаются приблизительно с точностью до 6 или 7 цифр.
Рассмотрим вариант по умолчанию для спецификации %e. Как мы видим, при ее использовании печатается одна цифра слева от десятичной точки и шесть справа. В результате получается слишком много цифр! Чтобы избежать этого, необходимо задать число цифр справа от десятичной точки, и последние четыре оператора программы реализуют как раз указанную возможность. Обратите внимание на то, как четвертый и шестой операторы производят округление выводимых данных.
Теперь исследуем некоторые варианты строк. Рассмотрим при|мер:

```
#define BLURB "Выдающееся исполнение!"
main()
{
printf(" /%2s/\n" , BLURB);
printf(" /%25.s/\n" , BLURB);
printf(" /%25.5s/\n" , BLURB);
printff /%-25.5s/\n" , BLURB);
}
```

Вот результат работы программы:

```
/Выдающееся испопнение!/
/ Выдающееся исполнение!/
/ Выдаю/
/Выдаю /
```

Обратите внимание на то, как поле расширяется для того, чтобы поместились все указанные символы. Заметим также, как спецификация точности ограничивает число символов, выводимых на печать. Символы .5 в спецификации формата указывают функции printf() на необходимость напечатать только пять символов.
Теперь вы ознакомились с некоторым количеством примеров. А знаете ли вы, как подготовить оператор печати, чтобы напечатать нечто вроде следующей фразы:
Семья NAME, возможно, лишь на XXX.XX долларов богаче!

Здесь NAME и XXX.XX представляют значения соответствующих переменных в программе, скажем name [40] и cash. Вот одно из решений:
printf(" Семья %s, возможно, лишь на %.2f долларов богаче!\n", name, cash);

До сих пор мы без тени сомнения применяли спецификации преобразования для переменных разных типов: например, %1 для типа float и т. д. Но, как мы уже видели в нашей программе поиска кода ASCII, для некоторого символа функцию printf() можно использовать также для преобразования данных из одного типа в другой! Мы не намерены, однако, терять чувство реальности и по-прежнему будем работать с целыми типами.

Использование функции printf() для преобразования данных
Здесь мы снова займемся выводом на печать целых чисел. Поскольку мы уже осведомлены о полях, то не будем заботиться об использовании символа /, чтобы отмечать их начало и конец:

```
main()
{
printf(" %d\n", 336);
printf(" %o\n", 336);
printf(" %x\n", 336);
printf(" %d\n", -336);
printff %u\n" , -336);
}
```

В нашей системе результат будет выглядеть следующим образом:

```
136
520
150
-336
65200
```

Как вы, по-видимому, и ожидали, при использовании спецификации %d будет получено число 336 точно так же, как в примере, обсуждавшемся чуть выше. Но давайте посмотрим, что произойдет, когда вы «попросите» программу напечатать это десятичное целое число в восьмеричном коде. Она напечатает число 520, являющееся восьмеричным эквивалентом 336 (5*64+2*8 + 0*1 = 336). Аналогично при печати этого числа в шестнадцатеричном коде мы получим 150.
Таким образом, мы можем использовать спецификации, применяемые для функции printf() с целью преобразования десятичных чисел в восьмеричные или шестнадцатеричные и наоборот. Или же если вы захотите напечатать данные в желаемом для вас виде, то необходимо указать спецификацию %d для получения десятичных чисел, %o — для восьмеричных, а %x— для шестнадцатеричных. При этом не имеет ни малейшего значения, в какой форме число первоначально появилось в программе.
Сделаем еще несколько замечаний относительно вывода на печать. Печать числа -336 при использовании спецификации %d не вызывает никакого затруднения. При

применении же спецификации %u (unsigned — беззнаковая) получаем число 65200, а не 336, как можно было бы ожидать. Причина получения такого результата лежит в способе представления отрицательных чисел в нашей системе. Здесь используется так называемый «дополнительный код». Числа от 0 до 32767 отображаются обычным образом, а от 32768 до 65535 представляют отрицательные числа, причем 65535 кодирует число — 1, 65534 — число -2 и т. д. Поэтому числу -336 соответствует 65536 — 336 = 65200. Этот метод применяется не во всех системах. Тем не менее отсюда следует вывод: не ожидайте, что спецификация преобразования %u приводит просто к отбрасыванию знака числа.
Сейчас мы переходим к обсуждению интересного примера, которого мы уже касались ранее, а именно к использованию функции printf() для нахождения кода ASCII некоторого символа. Например, оператор

printf(" %c%d\n" , ' A', ' A'); выдаст следующий результат:
A 65

A — это буква, а 65 — десятичный код ASCII символа A. Мы могли бы использовать спецификацию %o, если бы хотели получить восьмеричный код ASCII символа A.
Все вышесказанное дает хороший способ нахождения кодов ASCII для различных символов и наоборот. Вполне возможно, конечно, что вы предпочтете ему поиск кодов в приложении Ж.
Что произойдет, если вы попробуете преобразовать число, большее 255, в символ? Следующая строка и результат ее выполнения дадут ответ на этот вопрос:

printf(" %d %c\n" , 336, 336);
336 P

Десятичный код ASCII символа P равен 80, а 336 — это 256 + 80. Данное число, очевидно, интерпретируется по модулю 256. (Это математический термин, обозначающий остаток от деления числа на 256.) Другими словами, всякий раз при получении числа, кратного 256, отсчет начинается сначала, и 256 рассматривается как 0, 257 — как 1, 511 — как 255, 512 — как 0, 513 — как 1 и т. д. И наконец, попытаемся напечатать число (65616), превышающее максимальное значение, которое могут принимать данные типа int в нашей системе (32767):
printf("%d %d\n", 65616, 65616);

Результат будет выглядеть так:
65616 80

Мы снова видим, что действия выполняются по «модулю». На этот раз счет ведется группами по 65536. Числа между 32767 и 65536 будут выводиться на печать как отрицательные из-за способа их представления в памяти машины. Системы с разными размерами ячеек памяти, отводимых под данные целого типа, ведут себя в общем одинаково, но при этом дают разные числовые значения.
Мы не исчерпали всех возможных комбинаций данных и спецификаций преобразования, поэтому вы можете пытаться экспериментировать сами. Но будет лучше, конечно, если вы сможете заранее предсказать результат, который будет

получен при печати данных, когда используется какая-нибудь спецификация преобразования, выбранная вами.

Применение функции scanf()

Поскольку в дальнейшем мы будем пользоваться функцией scanf() лишь эпизодически, мы рассмотрим здесь только основные особенности ее применения.

Так же как для функции printf(), для функции scanf() указываются управляющая строка и следующий за ней список аргументов. Основное различие двух этих функций заключается в особенностях данного списка. Функция printf() использует имена переменных, константы и выражения, в то время как функция scanf() — только указатели на переменные. К счастью, при применении этой функции мы ничего не должны знать о таких указателях. Необходимо помнить только два правила:

1. Если вам нужно ввести некоторое значение и присвоить его переменной одного из основных типов, то перед именем переменной требуется писать символ &.
2. Если вы хотите ввести значение строковой переменной, использовать символ & не нужно.

Приведем правильную программу:

```
main()
{
int age;
float assets;
char pet [30];
printf(" Укажите ваш возраст, состояние и любимое животное.\n");
scanf(" %d %f", &age, &assets);
scanf(" %s", pet); /* & отсутствует при указании массива символов */
printf(" %d $%.0f %s\n" , age, assets, pet);
}
```

Вот пример диалога:

```
Укажите ваш возраст, состояние и любимое животное.
82
8345245.19 носорог
82 $8345245 носорог
```

Функция scanf() использует некоторые специальные знаки (пробелы, символы табуляции и «новая строка») для разбиения входного потока символов на отдельные поля. Она согласует последовательность спецификаций преобразования с последовательностью полей, опуская упомянутые специальные знаки между ними. Обратите внимание, что наша входная информация располагается на двух строках. Точно так же мы могли бы использовать одну или пять строк при условии, что вводимые величины разделяются по крайней мере одним знаком типа «новой строки», пробела или символа табуляции. Единственным исключением из этого является спецификация %c, обеспечивающая чтение каждого следующего символа даже в том случае, если это «пустой символ».

Функция scanf() использует практически тот же набор символов спецификации преобразования, что и функция printf(). Основные отличия в случае функции scanf()

следующие:

1. Отсутствует спецификация %g.
2. Спецификации %f и %e эквивалентны. Обе спецификации допускают наличие (или отсутствие) знака, строки цифр с десятичной точкой или без нее и поля показателя степени.
3. Для чтения целых чисел типа short применяется спецификация %h.

Функция scanf() не является одной из наиболее часто используемых функций языка Си. Мы обсуждаем ее здесь главным образом из-за ее универсальности (она позволяет читать данные всех имеющихся типов); однако в Си имеется еще несколько других функций, осуществляющих ввод, например getchar() и gets(), которые более удобны для выполнения конкретных задач — чтения одиночных символов или строк, содержащих пробелы.

Советы по применению. Задание фиксированной ширины полей оказывается полезным при печати данных столбцами. Поскольку шириной поля по умолчанию является «ширина» числа, при повторном использовании оператора

```
printf("%d %d %d\n", vall, val2, val3);
```

будут получены неровные столбцы чисел, если эти числа состоят из разного количества цифр. Например, результат мог бы выглядеть следующим образом:

```
12 234 1222
4 5 23
22334 2322 10001
```

(Здесь предполагается, что между обращениями к оператору печати значения переменных изменялись.)

Эти же данные можно представить в улучшенном виде, если задать достаточно большую фиксированную ширину поля. При использовании оператора

```
printf("%9d %9d %9d\n", vall, val2, val3);
```

результат будет выглядеть так:

```
       12      234     1222
        4        5       23
    22344     2322    10001
```

Наличие пробелов между спецификациями преобразования гарантирует, что даже в том случае, если все поле будет заполнено, символы, соответствующие данному числу, не перейдут в следующее по-
ле. Это вызвано тем обстоятельством, что обычные символы, имеющиеся в управляющей строке, включая пробелы, всегда печатаются.

С другой стороны, если печатаемое число включено в некоторую фразу, то часто при его выводе оказывается удобным задать ширину поля равной или меньше требуемой Это дает возможность включить число в фразу без добавления лишних пробелов. Например, результатом работы оператора

```
printf("Скороход Беппо пробежал %.2f мили за 3 ч.\n", distance);
```

могла бы быть следующая фраза:
Скороход Беппо пробежал 10.22 мили за 3 ч.

Изменяя спецификацию преобразования на %10.2f, получим
Скороход Беппо пробежал 10.22 мили за 3 ч.

Что вы должны были узнать.

- Что такое строка символов: несколько символов, расположенных в ряд
- Как записывать строку символов: " несколько символов, расположенных в ряд"
- Как строка хранится в памяти: " несколько символов, расположенных в ряд\0"
- Где разместить строку: char phrase [25] или static char phrase [25]
- Как определить длину строки: использовать функцию strlen (строка)
- Как напечатать строку: printf(" %s", phrase) Как прочитать строку, состоящую из одного слова-scanf(" %s" ,&name)
- Как задать числовую константу: #define TWO 2 Как задать символьную константу: #define WOW !' Как задать строковую константу: #define WARN "Не делай этого!"
- Спецификации преобразования при вводе-выводе: %d %f %e %g %c %s %u %o %x
- Как улучшить вид входной информации: %-10d %3.2f Как выполнять преобразования: printf(" %d %o %c\h", WOW, WOW, WOW);

Вопросы и ответы.

Вопросы

1. Выполните снова программу, приведенную в начале, но на этот раз в ответ на вопрос о вашем имени введите имя и фамилию. Что произойдет? Почему?
2. Что выведет на печать каждый из нижеприведенных программных фрагментов в предположении, что они являются частью некоторой полной программы?
   a. printf("Он продал картину за $%2.2f.\n", 2.345e2);
   b. printf("%c%c%c\n", 'H', 105, '\41');
   c. #define Q "Его Гамлет был смешным, но не вульгарным.". printf("%s\n имеет %d символов.\n", Q, strlen(Q));
   d. printf("%2.2e то же самое, что и %2.2f?\n", 1201.0, 1201.0);
3. Какие изменения необходимо внести в программу п. 2в, чтобы строка Q была выведена на печать заключенной в апострофы?
4. Очередная задача по обнаружению ошибок в программе.

```
define B a-яй-яй
define X 10
main()
{
int age;
char name;
printf(" Укажите, пожалуйста, свое имя.");
scanf(" %s" , name);
printf(" Прекрасно, %c, сколько вам лет?\n", name);
scanf(" %f", age);
xp = age + X;
```

```
printf(" %s! Вам должно быть по крайней мере %d.\n", B, xp);
}
```

Ответы

1. «Взрывоопасная» программа. Первый оператор scanf() читает ваше имя, оставляя фамилию непрочитанной; при этом она все-таки попадает во входной «буфер». (Этот буфер выполняет функции области памяти, используемой для временного хранения поступающих данных.) Следующий оператор scanf() должен ввести в программу величину вашего веса; он начинает вводить символы как раз с того места, где завершился предыдущий ввод, и поэтому читает вашу фамилию, принимая ее за вес. В результате в программу попадает «мусор». С другой стороны, если вы в ответ на вопрос об имени введете строку типа «Саша 144», то величина 144 будет рассматриваться как ваш вес, несмотря на то что вы ввели ее до того, как программа запросила величину веса.
2.
    a. Он продал картину за 234.50 долл.
    b. Hi!

Примечание: первый символ — это символическая константа, второй — десятичное целое число, преобразованное в символ, а третий — представление символической константы в коде ASCII.

    c. Его Гамлет был смешным, но не вульгарным, имеет 41 символ.
    d. 1.20E+03 то же самое, что и 1201.00?

3. Вспомните, что раньше говорилось по поводу управляющих последовательностей, и попробуйте записать оператор в таком виде:

```
printf(" \" %s\" \n имеет %d символов.\n", Q, strlen(Q));
```

4.
    - Строка 1: символ # опущен; вместо a-яй-яй должно стоять " a-яй-яй".
    - Строка 2: символ # опущен.
    - Строка 6: переменная name должна быть массивом, например char name [25].
    - Строка 8: в управляющей строке должен стоять символ \n.
    - Строка 10: вместо %c должно быть %s.
    - Строка 11: поскольку переменная age целого типа, необходимо использовать %d, а не %f; кроме того, вместо age должно стоять &age.
    - Строка 12: имя xp нигде не было описано.
    - Строка 13: правильная, но при выводе на печать результат будет испорчен из-за ошибки, допущенной при определении B.

Кроме того, программа служит примером плохого стиля программирования.

РАЗДЕЛ 2. ПРАКТИЧЕСКИЙ

ЛАБОРАТОРНЫЕ РАБОТЫ

Лабораторная работа №1

ЛИНЕЙНЫЕ АЛГОРИТМЫ

Цель работы: Научиться программировать простейшие линейные алгоритмы, компилировать исходную программу, находить и исправлять ошибки компиляции. Освоить операции ввода/вывода в консольном приложении.

Задание к работе

Осуществить построение простой программы на языке C++ по варианту условия, определенному номером подгруппы. Предусмотреть объявление основных типов переменных и организацию ввода/вывода с помощью операторов cin и cout, а также реализовать необходимые операции, указанные в задании. Предусмотреть комментарии в написанной программе.

Индивидуальные задания

1. Написать программу вычисления площади прямоугольника. Ввод данных производится с клавиатуры. Предусмотреть приглашение к вводу данных и читабельность результатов выполнения программы.
2. Написать программу вычисления объема параллелепипеда. Ввод данных производится с клавиатуры. Предусмотреть приглашение к вводу данных и читабельность результатов выполнения программы.
3. Написать программу вычисления площади поверхности параллелепипеда. Ввод данных производится с клавиатуры. Предусмотреть приглашение к вводу данных и читабельность результатов выполнения программы.
4. Написать программу вычисления объема куба. Ввод данных производится с клавиатуры. Предусмотреть приглашение к вводу данных и читабельность результатов выполнения программы.

5. Написать программу вычисления объема цилиндра. Ввод данных производится с клавиатуры. Предусмотреть приглашение к вводу данных и читабельность результатов выполнения программы.
6. Написать программу вычисления площади треугольника, если известна длина основания и высота. Ввод данных производится с клавиатуры. Предусмотреть приглашение к вводу данных и читабельность результатов выполнения программы.
7. Написать программу вычисления площади треугольника, если известны длины двух его сторон и величина угла между этими сторонами. Ввод данных производится с клавиатуры. Предусмотреть приглашение к вводу данных и читабельность результатов выполнения программы.
8. Написать программу вычисления сопротивления электрической цепи, состоящей из двух параллельно соединенных сопротивлений, а затем вычислить сопротивление электрической цепи, состоящей из двух последовательно соединенных сопротивлений.
9. Написать программу, вычисляющую скорость, с которой спортсмен пробежал дистанцию. К полученному результату применить операцию унарного вычитания. Ввод данных производится с клавиатуры. Предусмотреть приглашение к вводу данных и читабельность результатов выполнения программы. Закомментировать программу.
10. Написать программу вычисления площади поверхности цилиндра. Реализовать составную операцию присваивания. Ввод данных производится с клавиатуры. Предусмотреть приглашение к вводу данных и читабельность результатов выполнения программы. Закомментировать программу.
11. Написать программу пересчета расстояния из верст в километры (1 верста – это 1066,8 м). Реализовать явное преобразования типов. Ввод данных производится с клавиатуры. Предусмотреть приглашение к вводу данных и читабельность результатов выполнения программы. Закомментировать программу.

12. Написать программу вычисления силы тока в электрической цепи. Ввод данных производится с клавиатуры. Реализовать составную операцию присваивания. Предусмотреть приглашение к вводу данных и читабельность результатов выполнения программы. Закомментировать программу.
13. Написать программу вычисления расстояния между населенными пунктами, изображенными на карте. Исходными данными будут являться: масштаб карты (количество километров в одном сантиметре) и расстояние между точками, изображающими населенные пункты. К полученному результату применить операцию унарного сложения. Предусмотреть приглашение к вводу данных и читабельность результатов выполнения программы. Закомментировать программу.
14. Вычислить значение выражения $y = \frac{2 \cdot a + 16 \cdot b \cdot c - 8 \cdot c}{2} + e^{2 \cdot a \cdot b \cdot c} \cdot 32 \cdot a \cdot b$.

    Операции умножения и деления выполнить с помощью операций сдвига. Закомментировать программу.
15. Написать программу пересчета веса из фунтов в килограммы (1 фунт – 405,9 грамма). Реализовать явное преобразования типов. Ввод данных производится с клавиатуры. Предусмотреть приглашение к вводу данных и читабельность результатов выполнения программы. Закомментировать программу.
16. Написать программу вычисления поездки на автомобиле на дачу. Исходными данными являются: расстояние до дачи (км), количество бензина, которое потребляет автомобиль на 100 км пробега, цена одного литра бензина. К полученному результату применить операцию унарного сложения. Ввод данных производится с клавиатуры. Предусмотреть приглашение к вводу данных и читабельность результатов выполнения программы. Закомментировать программу.

Лабораторная работа №2

РАЗВЕТВЛЯЮЩИЕ И ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ

Цель работы: Научиться программировать разветвляющие алгоритмы с использованием инструкций if - else и switch - case. Изучить логические операторы !, && и ||. Научиться программировать циклические алгоритмы. Изучить операторы цикла while, do while, for .

Задание к работе

Осуществить построение программы на языке C++ по варианту задания, определенному номером подгруппы. Организовать форматированный ввод-вывод данных с использованием функций форматированного ввода/вывода printf(), scanf(). Для реализации поставленной задачи использовать указанную в задании структуру выбора или повторения.

Индивидуальные задания

1. Написать программу простейшего калькулятора (умножение, деление, сложение и вычитание), используя структуру выбора if-else. Предусмотреть невозможность деления на 0. Использовать форматированный ввод/вывод данных.
2. Написать программу простейшего калькулятора, используя структуру выбора switch. Предусмотреть невозможность деления на 0. Использовать форматированный ввод/вывод данных.
3. Написать программу для вычисления значений корней квадратного уравнения $a \cdot x^2 + b \cdot x + c = 0$. Использовать структуру выбора if-else. Использовать форматированный ввод/вывод данных.
4. Написать программу, которая переводит время из минут и секунд в секунды. Программа должна проверять правильность введенных пользователем данных и в случае, если данные не верные, выводить соответствующее

сообщение. Использовать структуру выбора if-else. Использовать форматированный ввод/вывод данных.

5. Написать программу, которая вычисляет оптимальный вес для пользователя, сравнивает его с реальным весом и выводит рекомендацию о необходимости поправиться или похудеть. Оптимальный вес вычисляется по формуле: Рост (см)-100. Использовать структуру выбора if-else. Использовать форматированный ввод/вывод данных.
6. Написать программу, которая запрашивает у пользователя номер месяца, а затем выводит соответствующее название времени года. В случае, если пользователь введет недопустимое число, программа должна вывести сообщение “Ошибка ввода данных”. Использовать структуру выбора if-else. Использовать форматированный ввод/вывод данных.
7. Написать программу, которая вычисляет стоимость междугороднего телефонного разговора (цена одной минуты определяется расстоянием до города, в котором находится абонент). Исходными данными для программы являются код города и длительность разговора. Использовать структуру выбора switch. Использовать форматированный ввод/вывод данных.
8. Написать программу вычисления площади кольца. Программа должна проверять правильность исходных данных. Использовать структуру выбора if-else. Использовать форматированный ввод/вывод данных.
9. Написать программу, которая выводит на экран таблицу квадратов первых десяти целых положительных чисел с использованием структуры повторения for. Использовать форматированный ввод/вывод данных.
10. Написать программу, которая выводит на экран таблицу значений функции $y=2x^2-5x-8$ в диапазоне от -4 до 4. Шаг изменения аргумента 0,5. Использовать структуру повторения while. Использовать форматированный ввод/вывод данных.
11. Написать программу, которая вычисляет сумму первых членов ряда 1,3,5,7…. Количество суммируемых членов ряда задается во время работы

программы. Использовать структуру повторения for. Использовать форматированный ввод/вывод данных.

12. Написать программу вычисления суммы нечетных чисел и количества четных чисел. Завершить программу после ввода десяти чисел или, когда будет введено три четных числа. Использовать структуру повторения for, операторы continue или break. Использовать форматированный ввод/вывод данных.
13. Написать программу, которая определяет максимальное число из введенной с клавиатуры последовательности положительных чисел (длина последовательности не ограничена). Использовать структуру повторения do while. Использовать форматированный ввод/вывод данных.
14. Написать программу, вычисляющую сумму и среднее арифметическое последовательности положительных чисел, которые вводятся с клавиатуры. Использовать структуру повторения do while. Использовать форматированный ввод/вывод данных.
15. Написать программу, которая вычисляет наибольший общий делитель двух целых чисел (алгоритм Евклида). Использовать структуру повторения while. Использовать форматированный ввод/вывод данных.
16. Написать программу вычисления факториала числа (n!) с использованием структуры повторения for. Использовать форматированный ввод/вывод данных.
17. Написать программу, которая вычисляет стоимость междугороднего телефонного разговора (цена одной минуты определяется расстоянием до города, в котором находится абонент). Исходными данными для программы являются код города и длительность разговора. Использовать структуру выбора switch. Использовать форматированный ввод/вывод данных.

Лабораторная работа №3

УКАЗАТЕЛИ И МАССИВЫ

Цель работы: Изучить понятие указателей. Объявление указателей. Использование указателей. Инициализация указателей. Арифметические действия над указателями. Обработка одномерных и многомерных массивов с помощью указателей.

Задание к работе

Осуществить построение программы на языке C++ по варианту задания, определенному номером подгруппы. Поставленную задачу реализовать, используя массивы данных либо переменные типа указатель (согласно заданию).

Индивидуальные задания

1. Написать программу, которая вводит с клавиатуры одномерный массив из десяти целых чисел, после чего выводит количество ненулевых элементов. Перед вводом каждого элемента должен выводиться текст с номером элемента. Использовать форматированный ввод-вывод данных.
2. Написать программу, которая выводит минимальный элемент введенного с клавиатуры массива целых чисел. Использовать форматированный ввод-вывод данных.
3. Написать программу, которая складывает матрицы вещественных чисел. Использовать форматированный ввод-вывод данных.
4. Написать программу, которая вводит строку, определяет ее длину и количество пробелов в ней. Использовать форматированный ввод-вывод данных.
5. Написать программу, которая проверяет, сколько раз введенное с клавиатуры число встречается в массиве. Использовать форматированный ввод-вывод данных.

6. Написать программу, которая вводит по строкам с клавиатуры двумерный массив и вычисляет сумму его элементов по столбцам. Использовать форматированный ввод-вывод данных.
7. Написать программу, которая вводит по строкам с клавиатуры двумерный массив и вычисляет сумму его элементов по строкам. Использовать форматированный ввод-вывод данных.
8. Написать программу, которая проверяет, представляют ли элементы введенного с клавиатуры массива возрастающую последовательность. Использовать форматированный ввод-вывод данных.
9. Написать программу, которая выводит минимальный элемент введенного с клавиатуры массива целых чисел. Для доступа к элементам массива использовать указатель. Использовать форматированный ввод-вывод данных.
10. Написать программу, которая выводит максимальный элемент введенного с клавиатуры массива целых чисел. Для доступа к элементам массива использовать указатель. Использовать форматированный ввод-вывод данных.
11. Написать программу, которая вводит строку, выделяет из строки слова, записывает их в массив и выводит массив слов. Использовать указатели и форматированный ввод-вывод данных.
12. Написать программу, которая сортирует строки в алфавитном порядке, используя функцию strcmp(). Использовать указатели и форматированный ввод-вывод данных.
13. Написать программу, которая вводит строку, вычисляет длину строки и количество пробелов в ней. Для вычисления длины строки использовать функцию strlen(). Использовать указатели и форматированный ввод-вывод данных.
14. Написать программу, которая задает строку, вычисляет ее длину и выводит строку в прямом и обратном порядке. Для вычисления длины строки

использовать функцию strlen(). Использовать указатели и форматированный ввод-вывод данных.

15. Написать программу, которая вводит строку, заменяет символ x в строке на символ y и выводит модифицированную строку. Необходимо так же вывести статистику замены символов в строке. Использовать указатели и форматированный ввод-вывод данных.
16. Написать программу, которая вводит две строки, вычисляет длину первой и второй строки, затем производит их конкатенацию и выводит результирующую строку вместе с ее длиной. Для объединения строк использовать функцию strcat(). Использовать указатели и форматированный ввод-вывод данных.

Лабораторная работа №4

ПОСТРОЕНИЕ ПРОГРАММЫ С ПОМОЩЬЮ ФУНКЦИЙ

Цель работы: Изучить механизм организации и использования функций. Понятия объявление функции, определение функции, вызов функции.

Задание к работе

Осуществить построение программы на языке C++ по варианту задания, определенному номером подгруппы. Реализацию поставленной задачи осуществить в отдельной функции. При написании программы использовать прототип функции.

Индивидуальные задания

1. Написать программу, содержащую функцию сортировки массива чисел методом пузырька. Ввод и вывод данных осуществить в функции main(). В программе использовать прототип функции.
2. Написать программу, содержащую функцию сортировки массива чисел выбором наименьшего элемента. Ввод и вывод данных осуществить в функции main(). В программе использовать прототип функции.

3. Написать программу, содержащую две функции: для вычисления факториала и для вывода таблицы факториалов. Предусмотреть передачу параметра в первую функцию по значению и передачу параметра во вторую функцию по ссылке с использование указателей. В программе использовать прототипы функций.
4. Написать программу, содержащую функцию, обеспечивающую решение квадратного уравнения. Параметрами функции должны быть коэффициенты и корни уравнения. Значение, возвращаемое функцией, должно передавать в вызывающую программу информацию о наличии уравнения корней: 2 – два разных корня, 1 – корни одинаковые, 0 – уравнение не имеет решения. Если исходные данные неверные, то функция должна возвращать -1. Ввод, проверку корректности исходных данных с использованием структуры выбора switch и вывод результатов на экран осуществить в функции main(). В программе использовать прототип функции.
5. Написать программу, в которой необходимо найти корень уравнения, используя метод половинного деления. Предусмотреть использование указателя на функцию и прототипа функции. Вывести на экран корень уравнения и количество итераций.
6. Написать программу, в которой необходимо найти корень уравнения, используя метод хорд. Предусмотреть использование указателя на функцию и прототипа функции. Вывести на экран корень уравнения и количество итераций.
7. Написать программу, в которой необходимо найти корень уравнения, используя метод касательных. Предусмотреть использование указателя на функцию и прототипа функции. Вывести на экран корень уравнения и количество итераций.
8. Написать программу, в которой необходимо определить массив указателей на функции. Вводить цифру, определяющую какую функцию надо выполнить: 0-найти минимальное число, 1-найти максимальное число, 2-вычислить сумму чисел, 3-вычислить разность чисел, 4-найти произведение

чисел, 5-найти частное чисел, 6-завершить работу. Выполнить соответствующую функцию, используя указатель на нее, и вывести результаты на экран. В программе предусмотреть использование прототипа функции.

9. Написать программу, содержащую рекурсивную функцию вычисления чисел Фибоначчи. Предусмотреть ввод количества чисел в последовательности и вывести саму последовательность Фибоначчи на экран.
10. Написать программу, содержащую функцию со списком аргументов переменной длины. В функции необходимо подсчитать произведение чисел.
11. Написать программу, содержащую встроенную функцию, использующую аргументы по умолчанию. В функции необходимо рассчитать объем цилиндра.
12. Написать программу с использованием аргументов командной строки. В командной строке задается признак фигуры, площадь которой необходимо вычислить. Вычисление площади круга, квадрата и прямоугольника реализовать в отдельных функциях. Обращение к функциям реализовать через указатель на функцию.
13. Написать программу с использованием аргументов командной строки. В командной строке задается признак фигуры, площадь которой необходимо вычислить. Вычисление площади треугольника и трапеции реализовать в отдельных функциях. Обращение к функциям реализовать через указатель на функцию.
14. Написать программу для проверки вводимого с клавиатуры строкового пароля. Проверка его на корректность осуществляется в теле функции. Предусмотреть перегрузку функции.
15. Написать программу для проверки вводимого с клавиатуры числового пароля. Проверка его на корректность осуществляется в теле функции. Предусмотреть перегрузку функции.

16. Написать программу с использованием шаблонов функций. Необходимо ввести элементы массива целого и вещественного типов и вывести среднее арифметическое элементов каждого массива.

РАЗДЕЛ 3. КОНРОЛЬ ЗНАНИЙ

Общая формулировка заданий к контрольной работе

Номер варианта выбираем по номеру в журнале. При совпадении вариантов обе работы не принимаются.

Требования к отчету по контрольной работе (приложение 1). Оформляется один отчет на все задания.

Контрольная работа №1

Программирование линейных алгоритмов

Цель работы: выработать практические навыки работы с интегрированной средой разработки (IDE) на языке C/C++, научиться создавать, вводить в компьютер, выполнять и исправлять простейшие программы на языке C/C++.

Общие сведения:

Линейным называется алгоритм, в котором результат получается путем однократного выполнения заданной последовательности действий при любых значениях исходных данных. Операторы программы выполняются последовательно, один за другим, в соответствии с их расположением в программе.

Задачи:

1. Дана длина ребра куба. Найти объем куба и площадь его боковой поверхности.
2. Три сопротивления R1, R2, R3 соединены параллельно. Найти сопротивление соединения.
3. Вычислить значения функций

$$y = \left(e^{-x_1} + e^{-x_2}\right)/2 \text{ и } z = \left(a\sqrt{x_1} - b\sqrt{x_2}\right)/c,$$

где $x_1 = \left(b + \sqrt{|b^2 - 4ac|}\right)/2a$, $x_2 = \left(b - \sqrt{|b^2 - 4ac|}\right)/2a$.

4. Известна длина окружности. Найти площадь круга, ограниченного этой окружностью.
5. Треугольник задан координатами своих вершин. Найти:
   - периметр треугольника;
   - площадь треугольника.

6. Вычислить высоту треугольника, опущенную на сторону a, по известным значениям длин его сторон a, b, c.

7. Вычислить значение функции $y = ae^{-ax}\sin\omega x$ при $x = \left(\frac{\pi}{2} - \varphi\right)\Big/\omega$.

8. Вычислить объем цилиндра с радиусом основания r и высотой h.

9. Определить расстояние, пройденное физическим телом за время t, если тело движется с постоянным ускорением a и имеет в начальный момент времени скорость V0.

10. Вычислить координаты центра тяжести трех материальных точек с массами m_1, m_2, m_3 и координатами (x_1, y_1), (x_2, y_2), (x_3, y_3) по формулам:

$$x_0 = \frac{m_1x_1 + m_2x_2 + m_3x_3}{m_1 + m_2 + m_3};$$

$$y_0 = \frac{m_1y_1 + m_2y_2 + m_3y_3}{m_1 + m_2 + m_3}.$$

11. Определить координаты вершины параболы y=ax2+bx+c (a№0). Коэффициенты a,b,c заданы.

12. Вычислить полную поверхность цилиндра $S_{полн}=2\pi R(H+R)$.

13. Вычислить площадь поверхности $S = \pi(R+r)l + \pi R^2 + \pi r$ и объем $V = \frac{1}{3}\pi\left(R^2 + r^2 + Rr\right)h$ усеченного конуса.

14. По данным сторонам прямоугольника вычислить его периметр, площадь и длину диагонали.

15. Определить расстояние на плоскости между двумя точками с заданными координатами M1(x1,y1) и M2(x2,y2).

16. Вычислить координаты точки, делящей отрезок a_1a_2 в соотношении $n_1 : n_2$ по формулам

$$x = (x_1 + \gamma x_2)/(1+\gamma); \quad y = (y_1 + \gamma y_2)/(1+\gamma),$$

где $\gamma = n_1/n_2$.

Контрольная работа №2

Программирование разветвляющихся алгоритмов

Цель работы: научиться использовать операторы ветвления. Научиться составлять программы решения задач на разветвляющиеся алгоритмы.

Общие сведения

Алгоритм называется разветвляющимся, если он содержит несколько ветвей, отличающихся друг от друга содержанием вычислений. Выход

вычислительного процесса на ту или иную ветвь алгоритма определяется исходными данными задачи.

Перед выполнением работы необходимо ознакомится с правилами записи логических выражений, операций сравнения, операторов ветвления if, case.

Задачи

1. Даны действительные числа x, y. Если x, y отрицательны, то каждое значение заменить его модулем; если отрицательное только одно из них, то оба значения увеличить на 0.5; если оба значения не отрицательны и ни одно из них не принадлежит отрезку [0.5, 2.0], то оба значения уменьшить в 10 раз; в остальных случаях x, y оставить без изменения.

2. Меньшее из двух чисел заменить их полусуммой, большее - их удвоенным произведением.

3. По заданному номеру месяца вывести название следующего месяца. Использовать оператор case.

4. Вычислить значение $a=\begin{cases} e^{2t_1}+\cos(xt_2) \\ \ln(xt_1)\cdot\sqrt{\sin(t_2)} \\ xt_1 \end{cases}$ при $\begin{cases} t_1 < \sqrt{t_2+2} \\ t_1 = \sqrt{t_2+2} \\ t_1 > \sqrt{t_2+2} \end{cases}$

5. Даны действительные положительные числа x, y, z. Выяснить, существует ли треугольник с длинами сторон x, y, z (треугольник существует, если длина каждой стороны меньше суммы двух других сторон).

6. Написать программу простейшего калькулятора (сложение, вычитание, умножение, деление). Предусмотреть невозможность деления на 0. Использовать оператор case.

7. Определить и вывести на печать номер квадранта, в котором расположена точка M(x, y), x и y заданные вещественные числа.

8. Из величин, определяемых выражениями a=sinx, b=cosx, c=ln|x| при заданном x, определить и вывести на экран дисплея минимальное значение.

9. Определить знак заданного целого числа. Ответом должно быть «+», «-», «0». Использовать оператор case.

10. Вычислить значение функции Y для заданного пользователем значения X.

$y=\sqrt{a+b}$ $\quad a=\dfrac{1}{x+5}-\dfrac{1}{x-9}$ $\quad b=\begin{cases} \cos(x), при\, x \ge 0 \\ \sin(x), при\, x < 0 \end{cases}$

11. Определить, какая из двух точек - M1(x1,y1) или M2(x2,y2) - расположена ближе к началу координат. Вывести на экран дисплея координаты этой точки.

12. В японском календаре принят двенадцатилетний цикл. Годы внутри цикла носят названия животных: крысы, коровы, тигра, зайца, дракона, змеи, лошади, овцы, обезьяны, петуха, собаки, свиньи. Написать программу, которая позволяет ввести номер года нашей эры и выводит его название по японскому календарю. (1996 г. – начало очередного цикла). Использовать оператор case.

13. Определить, какая из двух фигур (круг или квадрат) имеет большую площадь. Известно, что сторона квадрата равна a, радиус круга r. Вывести на экран название и значение площади большей фигуры.

14. Определить, попадает ли точка M(x,y) в круг радиусом r с центром в точке (x0,y0).

15. Написать программу ввода буквы, цифры или спецзнака. Выводить сообщения «Это цифра …» или «Это буква …», «Это спецзнак …». К сообщению добавлять саму цифру, букву или спецзнак. Использовать оператор case.

16. Вычислить значение функции Y для заданного пользователем значения X.

1. $y=\begin{cases} \dfrac{1}{a+b}, при\ a \le b \\ a+\sqrt{b}, при\ a > b \end{cases}$ $a=\begin{cases} \cos(x), при\ x \ge 0 \\ \sin(x), при\ x < 0 \end{cases}$ $b=\sqrt{\dfrac{1}{x+1}}$

Контрольная работа №3

Программирование циклических алгоритмов

Цель работы: закрепить практические навыки работы с интегрированной средой разработки (IDE) на языке C/C++, научиться использовать различные операторы циклов; научиться составлять программы решения задач с использование циклических структур.

Общие сведения

Алгоритм называется циклическим, если он содержит многократное выполнение одних и тех же операторов при различных значениях промежуточных данных. Число повторений этих операторов может быть задано в явной (цикл с известным заранее числом повторений) или неявной (цикл с неизвестным заранее числом повторений) форме.

Перед выполнением работы необходимо изучить различные организации циклов и операторы for, while, foreach.

Задачи

1. Вычислить $z = \frac{a\sqrt[3]{x + \ln^2 y}}{|t^3|}$; Параметр **y** изменяется от y=$y_н$=0,3 до y=$y_к$=0,9 с шагом h_6=0,2. a, x, t – константы. Использовать цикл while или for.

2. Дано натуральное n. Вычислить: $(1+\frac{1}{1^2})+(1+\frac{1}{2^2})+(1+\frac{1}{3^2})+\ldots+(1+\frac{1}{n^2})$;

3. Дано натуральное n. Вычислить: $\frac{1}{\sin 1}+\frac{1}{\sin 1+\sin 2}+\ldots+\frac{1}{\sin 1+\ldots \sin n}$;

4. Для введенной последовательности целых чисел признаком конца которой является ноль определить максимальное число, сумму всех чисел и среднее арифметическое. Использовать цикл for.

5. Дано действительное число x, натуральное n. Вычислить: x (x - n)(x - 2 n)(x - 3 n)…(x - n2);

6. Дано действительное число x, натуральное n. Вычислить: $\frac{1}{x}+\frac{1}{x(x+1)}+\ldots+\frac{1}{x(x+1)\ldots(x+n)}$;

7. Вычислить $w = \frac{ax^2 + Sin^2 Z}{\sqrt{1+e^y}}$; Параметр x изменяется: от x=$x_н$=1 до x=$x_к$=4,5 с шагом h=0,5. a, z, y – константы. Использовать цикл while или for.

8. Дано действительное число x, натуральное n. Вычислить: $\frac{x^1}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots+\frac{x^n}{n!}$;

9. Дано натуральное n. Вычислить: $\prod_{i=1}^{n}(2 + 1/i!)$;

10. Вычислить $t = \frac{b^2 + \sqrt{|q|}}{\cos^2 x + b\ln y}$; Параметр x изменяется от x=$x_н$=1 до x=$x_к$=5 с шагом h=1. b, q, y – константы. Использовать цикл while или for.

11. Вычислить приближенно значение бесконечной суммы: $1+\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{4^2}+\ldots = \frac{\pi^2}{6}$;

12. На промежутке от 1 до M найти все числа Армстронга. Натуральное число из n цифр называется числом Армстронга, если сумма его цифр, возведенных в n-ю степень, равна самому числу.

13. Вычислить $q = \frac{Sin^2(z+a)^3}{t\sqrt{e^{a+2q}}}$. Параметр z изменяется от $z=z_н=0,5$ до $z=z_к=1$ с шагом h=0,1. a, q, t – константы. Использовать цикл while или for.

14. Вычислить приближенно значение бесконечной суммы: $\frac{1}{1\cdot 3}+\frac{1}{2\cdot 4}+\frac{1}{3\cdot 5}+\ldots=\frac{3}{4}$;

15. Вычислить приближенно значение бесконечной суммы: $1+\frac{x^1}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots=e^x$; Нужное приближение считается полученным, если вычислена сумма нескольких первых слагаемых, и очередное слагаемое оказалось по модулю меньше данного положительного числа e.

16. Вычислить $z = \frac{3x^2-\sqrt{Cos(q^3)}}{\ln\ (y+\alpha\ )t}$; Параметр **x** изменяется от $x=x_н=0,2$ до $x=x_к=0,6$ с шагом h=0,1. a, q, t – константы. Использовать цикл while или for.

Контрольная работа 4

Программирование с использованием массивов

Цель работы: научиться описывать различные массивы, уметь инициализировать массивы, распечатывать содержимое массива; научится решать задачи на использование массивов.

Общие сведения:

Массив - это структурированный тип данных, который используется для описания упорядоченной совокупности фиксированного числа элементов одного типа, имеющих общее имя. Для обозначения элементов массива используются имя переменной-массива и индекс.

Перед выполнением работы необходимо изучить правила описания и использования переменных типа массив, типизированных констант типа массив.

Задания:

1. Вычислить X=m/(n-m), где m - сумма квадратов отрицательных элементов первого вектора, n - сумма квадратов отрицательных элементов второго вектора.

2. В массиве из n элементов найти сумму элементов массива, среднее арифметическое чисел, входящих в массив, произведение положительных элементов.

3. Составить вектор, элементы которого равны среднему арифметическому каждой строки прямоугольной матрицы.

4. Найти наибольший элемент, расположенный в заштрихованной части матрицы размерности n*n.

а) б) в) г)

5. Дана квадратная вещественная матрица размерности n. Найти количество нулевых элементов, стоящих: выше главной диагонали; ниже главной диагонали.

6. В массиве из n элементов найти количество нулевых элементов массива, сумму отрицательных и произведение положительных элементов массива.

7. Дана последовательность 100 различных целых чисел. Найти сумму чисел этой последовательности, расположенных между максимальным и минимальным числами (в сумму можно включить и сами эти два числа).

8. Дана вещественная матрица размерности n * m. Сформировать вектор b, в котором элементы вычисляются как: - произведение элементов соответствующих строк; - среднее арифметическое соответствующих столбцов; - разность наибольших и наименьших элементов соответствующих строк; - значения первых отрицательных элементов в столбце.

9. Дано вещественное число r и массив размера n. Найти элемент массива, который наиболее и наименее близок к данному числу.

10. Если в массиве присутствуют отрицательные элементы, заменить их значения средним арифметическим массива. Подсчитать и вывести количество совпадающих элементов массива.

11. Дан двухмерный массив A[1..m,1..n]. Написать программу построения одномерного массива B[1..m], элементы которого соответственно равны а) суммам элементов строк, б) произведениям элементов строк, в) наименьшим средних арифметических элементов строк.

12. Расположить элементы массива в обратном порядке (первый элемент меняется с последним, второй - с предпоследним и т.д. до середины; если массив содержит нечетное количество элементов, то средний остается без изменения).

13. Найти в массиве наибольшее число повторений элементов. Вывести значение элемента и количество его повторений.

14. В массиве поменять местами элементы, стоящие на нечетных местах, с элементами, стоящими на четных местах.

15. Заменить положительные элементы матрицы нулями, отрицательные – единицами. Переписать массив таким образом, чтобы нулевые элементы стояли в его конце, но при этом сохранялась очередность других.

16. Дан двумерный массив. В каждой строке все его положительные элементы переписать (сохраняя порядок) в начало строки, а отрицательные элементы - в конец массива. Дополнительный массив не использовать.

Контрольные вопросы

1. Каким образом определяются переменные типа массив (одномерный и двумерный)?
2. Как осуществляется доступ к отдельному элементу одномерного и двумерного массива?
3. Каким образом вводятся и выводятся элементы массива на экран?

Контрольная работа 5

Программирование с использованием строк

Цель работы: познакомить с понятием строка и выработать навыки работы с символьной информацией в языке программирования C/C++, научиться использовать строки символов при решении задач.

Общие сведения

Переменные типа string аналогичны массивам типа char. Их отличием является то, что число символов (длина строки) может динамически меняться в интервале от единицы до заданного верхнего значения.

Перед выполнением работы необходимо ознакомиться с правилами описания и использования строк, допустимых операций над ними, соответствующими стандартными процедурами и функциями.

Задания

Обработка текста: В следующих заданиях под словом "текст" понимается строка символов, слова в которой, разделены пробелами, специальными символами .,:;?!.

1. Дан текст. Требуется напечатать все слова с удвоенной буквой "н". Вывести самое длинное и короткое слово из строки.

2. Дан текст. а) Подсчитать количество слов в данной строке. б) Подсчитать количество букв "а" в последнем слове данной строки. в) Найти количество слов, начинающихся с буквы "б". г) Найти количество слов, у которых первый и последний символы совпадают между собой.

3. Составить программу циклической перестановки букв в словах текста так, что i-я буква слова становится i+1-ой, а последняя - первой.

4. В каждом слове текста замените "а" на букву "е", если "а" стоит на четном месте, и заменить букву "б" на сочетание "ак", если "б" стоит на нечетном месте.

5. Дан текст, содержащий от 2 до 30 слов, в каждом из которых от 2 до 10 латинских букв; между соседними словами - не менее одного пробела. Вывести все слова, отличные от последнего слова, предварительно преобразовав каждое из них по следующему правилу: 1) перенести первую букву в конец слова; 2) перенести последнюю букву в начало слова.

6. Отредактировать заданное предложения текста, удаляя из него все слова с нечетными номерами и переворачивая слова с четными номерами. Например, HOW DO YOU DO -> OD OD

7. Дан текст. Напечатать все слова, отличные от последнего слова, предварительно преобразовав каждое из них по следующему правилу: 1) оставить в слове только первые вхождения каждой буквы; 2) если слово нечетной длины, то удалить его среднюю букву

8. Составить таблицу слов данного текста, начинающихся с буквы "а", с указанием числа повторений каждого слова.

9. Составить программу для вычеркивания из слов текста всех букв, стоящих на нечетных местах после буквы "а".

10. Посчитать количество английских букв a и b в слове. В случае если a>b, то необходимо удалить все b. Перевести строчные буквы в заглавные.

11. Вывести из строки все слова, где количество гласных букв равно согласным. Найти длину самого короткого слова.

12. Вывести из строки слова, которые являются числами, кратными заданному числу m.

13. В тексте все повторяющиеся рядом стоящие символы удалить. Подсчитать количества вхождений каждого символа в строку.

14. Изменить строку таким образом, чтобы все различные символы остались по одному разу, причем остаются последние вхождения этих символов в строку.

15. Удалить слова, которые начинаются и заканчиваются одной буквой, остальные продублировать. Подсчитать число слов в тексте.

16. Даны два числа: N1 и N2, и две строки: s1 и s2. Получить из этих строк новую строку, объединив N1 первых символов строки s1 и N2 последних символов строки s2.

Контрольные вопросы

1. Как описываются строковые переменные?
2. Какая максимальная длина строки допустима в C/C++?
3. Какие операции допустимы над строковыми данными?
4. В чем отличие строковой переменной от массива символов?
5. Какие стандартные процедуры и функции для работы со строками вы знаете?

Контрольная работа 6

Программирование с использованием процедур и функций

Цель работы: познакомиться с понятиями процедура и функция в языке программирования C/C++, рассмотреть их сходства и различия, закрепить практические навыки работы с интегрированной средой разработки (IDE) на языке C/C++ на примере реализации алгоритмов при помощи процедур и функций, научиться применять метод последовательной детализации в практическом программировании; применять процедуры и функции при решении задач.

Общие сведения

Часто в программе обнаруживаются однотипные участки, которые выполняют одни и те же вычисления, но с различными данными. Такие части программы целесообразно оформлять в виде подпрограмм. Перед выполнением данной работы необходимо изучить правила описания процедур и функций, механизм передачи параметров, ознакомиться с понятием локальной и глобальной переменной.

Задания

1. Найти сумму положительных элементов в массиве. Использовать процедуру для ввода элементов массива и функцию для подсчета суммы.
2. Используя функцию нахождения длины отрезка (по теореме Пифагора), найти длину периметра и площадь треугольника, заданного координатами своих вершин.
3. Даны действительные числа x1, y1, x2, y2, …,x10, y10. Найти периметр десятиугольника, вершины которого имеют соответственно координаты (x1, y1), (x2, y2), …, (x10, y10). (Определить процедуру вычисления расстояния между двумя точками, заданными своими координатами.)

4. Найти среднее арифметическое положительных элементов в массиве. Использовать процедуру для ввода элементов массива и функцию для подсчета среднего арифметического.

5. Даны действительные числа a, b, c, d, e - стороны пятиугольника. Найти площадь пятиугольника. (Определить процедуру вычисления площади треугольника по его сторонам.)

6. Даны три символьные матрицы.
a) ту матрицу, где есть хотя бы одна гласная - транспонировать;
b) в той матрице, на главной диагонали которой все цифры, найти наименьшую и удалить соответствующую строку.

7. Подсчет количества чисел в массиве, которые больше заданного числа.

8. Даны отрезки a,b,c и d. Для каждой тройки этих отрезков, из которых можно построить треугольник, напечатать площадь данного треугольника. Определить процедуру Plo(x,y,z), выводящую площадь треугольника со сторонами x,y и z, если такой треугольник существует.

9. Даны длины a,b и c сторон некоторого треугольника. Найти медианы треугольника, сторонами которого являются медианы исходного треугольника. Длина медианы, проведенной к стороне a, равна $\frac{1}{2}\sqrt{2b^2+2c^2-a^2}$.

10. Вывести все четырехзначные числа, в которых цифры не повторяются.

11. Даны координаты вершин двух треугольников. Определить, какой из них имеет большую площадь.

12. Даны координаты вершин треугольника и координаты некоторой точки внутри него. Найти расстояние от данной точки до ближайшей стороны треугольника. (При определении расстояний учесть, что площадь треугольника вычисляется и через три его стороны, и через основание и высоту.).

13. Произведение элементов двух массивов, стоящих на нечетных позициях.

14. Определить максимальное число из четырех введенных, путем сравнения их сначала попарно, а затем результат между собой.

15. вычислить функцию $z(x)=x^2$ и сумму членов ряда $y=\pi^2-4(\cos x-\frac{\cos 2x}{2^2}+\frac{\cos 3x}{3^2}-...)$ на диапазоне $[-\pi;\pi]$. Очередной элемент включается в сумму, если его значение по модулю превышает некоторое заранее заданное число P, определяемое с требуемой точностью вычислений. P=0.001

16. Подсчет количества чисел в массиве, которые меньше заданного числа.

Контрольные вопросы

1. Для чего нужны в программе процедуры и функции?
2. В чем отличие между процедурой и функцией?
3. Чем отличаются формальные и фактические параметры?
4. Чем отличаются параметры-значения и параметры-переменные?
5. Как объявляются глобальные и локальные переменные? Каково правило видимости этих переменных?
6. Почему при обращении к процедуре, аргумент, передаваемый параметру-переменной, может быть только переменной, а не константой или выражением?

Контрольная работа 7

Работа с файлами

Цель работы: познакомить с понятием файлового типа данных (типизированные, текстовые и нетипизированные файлы); выработать навыки работы с файловым типом данных в языке программирования C/C++. научиться считывать информацию из файлов, записывать информацию в файл; научится решать задачи с использованием файлов.

Общие сведения

Файл представляет собой структурированный тип данных, содержащий последовательность компонентов одного типа и одной длины. Число элементов в файле (длина файла) не фиксировано. Это является основным отличием файла от массива.

Файл можно представить как ленту, у которой есть начало, а конец не фиксирован. Элементы файла записываются на эту ленту последовательно, друг за другом с помощью некоторого устройства - указателя файла. При чтении или записи этот указатель перемещается к следующему элементу и делает его доступным для обработки. В каждый момент доступен для чтения или записи только тот элемент файла, на который установлен указатель.

Задания

1. Сведения об ученике состоят из его имени и фамилии и названия класса (года обучения и буквы), в котором он учится. Дан типизированный файл

f, содержащий сведения об учениках школы:
а) выяснить, имеются ли в школе однофамильцы; б) выяснить, имеются ли в школе однофамильцы, у которых совпадает и имя и фамилия;
в) выяснить на сколько человек в восьмых классах больше, чем в десятых;
г) собрать в файле g сведения об учениках 9-х и 10-х классов, поместив вначале сведения об учениках класса 9а, затем 9б и т.д., затем 10а, 10б и т.д.

2. Дан текстовый файл f, компоненты которого являются целыми числами. Получить в файле g все компоненты файла f: а) являющимися четными числами; б) делящиеся на 3 и не делящиеся на 7; в) являющимися точными квадратами.

3. Дан текстовый файл f, компоненты которого являются целыми числами. Никакая из компонент файла не равна нулю. Файл f содержит столько же отрицательных чисел, сколько и положительных. Используя вспомогательный файл h, переписать компоненты файла f в файл g так, чтобы в файле g:
а) не было двух соседних чисел с одинаковым знаком;
б) вначале шли положительные, затем отрицательные числа;
в) числа шли в следующем порядке: два положительных, два отрицательных, два положительных, два отрицательных и т.д. (предполагается, что число компонент в файле f делится на 4).

4. Дан типизированный файл f, содержащий сведения о кубиках: размер каждого кубика (длина ребра в сантиметрах), его цвет (красный, зеленый, желтый или синий) и материал (деревянный, металлический, картонный). Найти:
а) количество кубиков каждого из перечисленных цветов и их суммарный объем;
б) количество деревянных кубиков с ребром 3 см и количество металлических кубиков с ребром, большим 5 см.

5. Дан текстовый файл f, компоненты которого являются целыми числами. Получить файл g, образованный из файла f исключением повторных вхождений одного и того же числа.

6. Дан типизированный файл f, содержащий различные даты. Каждая дата - это число, месяц и год. Найти: а) год с наименьшим номером; б) все весенние даты; в) самую позднюю дату. Найденные данные записать в файл g.

7. Дан текстовый файл f, компоненты которого являются целыми числами. Никакая из компонент файла не равна нулю. Числа в файле идут в следующем порядке: десять положительных, десять отрицательных, десять положительных, десять отрицательных и т.д. Переписать компоненты файла f в файл g так, чтобы в файле g числа шли в следующем порядке: а) пять положительных, пять отрицательных, пять положительных, пять отрицательных

и т.д.; б) двадцать положительных, двадцать отрицательных, двадцать положительных, двадцать отрицательных и т.д. (предполагается, что число компонент в файле f делится на 40).

8. Сведения об ученике состоят из его имени и фамилии и названия класса (года обучения и буквы), в котором он учится. Дан типизированный файл f, содержащий сведения об учениках школы:
а) выяснить, имеются ли однофамильцы в каких-либо параллельных классах;
б) выяснить, имеются ли однофамильцы в каком-нибудь классе; в) выяснить, имеются ли однофамильцы в каких-либо параллельных классах у которых совпадает и имя и фамилия;
г) выяснить, имеются ли однофамильцы в каком-нибудь классе, у которых совпадает и имя и фамилия;
д) выяснить, в каких классах насчитывается более 35 учащихся;

9. Дан текстовый файл f, компоненты которого являются целыми числами. Записать в файл g наибольшее значение первых пяти компонент файла f, затем - следующих пяти компонент и т.д. Если в последней группе окажется менее пяти компонент, то последняя компонента файла g должна быть равна наибольшей из компонент файла f, образующих последнюю (неполную) группу.

10. Дан типизированный файл f, содержащий сведения о книгах. Сведения о каждой из книг - это фамилия автора, название и год издания. 1) Найти названия книг данного автора, изданных с 1960 г. 2) Определить, имеется ли книга с названием "Программирование на Pascal". Если да, то сообщить фамилию автора и год издания. Если таких книг несколько, то сообщить имеющиеся сведения обо всех книгах.

11. Дан символьный файл f: а) подсчитать число вхождений в файл сочетаний 'ab'; б) определить входит ли в файл сочетание 'abcdefgh'; в) подсчитать число вхождений в файл каждой из букв 'a','b','c','d','e','f' и вывести результат в виде таблицы

a --> Na b --> Nb c --> Nc
d --> Nd e --> Ne f --> Nf

где Na, Nb, Nc, Nd, Ne, Nf - числа вхождений соответствующих букв.

12. Сведения об автомобиле состоят из его марки, номера и фамилии владельца. Дан типизированный файл f, содержащий сведения о нескольких автомобилях. Найти: а) фамилии владельцев и номера автомобилей данной марки; б) количество автомобилей каждой марки. Найденные данные записать в файл g.

13. Дан символьный файл f. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Удалить из файла все однобуквенные слова и лишние пробелы. Результат записать в файл g. Переписать из файла g в файл h строки в перевернутом виде, порядок строк должен быть обратным.

14. Сведения об ученике состоят из его имени и фамилии, названия класса (года обучения и буквы), в котором он учится и отметки. Дан типизированный файл f, содержащий сведения об учениках школы: а) выяснить, сколько учеников школы не имеют отметок ниже четырех; б) собрать в файле g сведения о лучших учениках школы, т.е. об учениках, не имеющих отметок ниже четырех и по сумме баллов не уступающих другим ученикам своего и параллельных классов.

15. Дан текстовый файл f. Записать в файл g компоненты файла f в обратном порядке. Даны текстовые файлы m и n. Записать в файл h сначала компоненты файла m, затем - компоненты файла n с сохранением порядка.

16. Дан типизированный файл f в который вводятся имена, пол, возраст и рост человека. Программа считывает данные из файла и выдает совпадения а) есть ли в нем мужчины одного роста; б) женщины с одинаковыми именами; в) мужчины и женщины одного возраста. Полученные данные записать в файл g.

Контрольные вопросы

1. Что такое файл? Какие существуют виды файлов?
2. Какими стандартными процедурами и функциями располагает Pascal для работы с файлами?
3. Каковы особенности работы с текстовыми файлами?
4. Каковы особенности работы с типизированными файлами?

Приложение 1

ТРЕБОВАНИЯ К ОФОРМЛЕНИЮ ОТЧЕТА ПО КОНТРОЛЬНОЙ РАБОТЕ

1. Отчет по контрольной работе сдается в бумажном виде после обязательной регистрации. Преподавателю отправляется в электроном виде.
2. Имя файла отчета «**Фамилия_Номер_группы_Номер_варианта**».

Пример: Иванов_4017412_15.docx

3. Версия редактора Word не менее Word 2010. Общие требования к оформлению текста представлены ниже.
4. Содержание отчета

- титульный лист
- содержание
- задание
- исходный код
- скриншоты работы приложения
- заключение
- список использованной литературы

Требования к разделам: Титульный лист должен обязательно содержать – фамилию, имя, отчество, номер группы, номер варианта. Исходный код, представленный в отчете, должен **обязательно** содержать комментарии к ключевым строкам программы (подписать классы, методы). Скриншоты работы приложения должны полностью раскрывать все функциональные возможности разработанного приложения.

ОБЩИЕ ТРЕБОВАНИЯ ПО ОФОРМЛЕНИЮ КОНТРОЛЬНОЙ РАБОТЫ

Параметры страницы

Формат листа – А4 (размер 210 × 290 мм). Перед набором текста настроить параметры **Microsoft Word**:

- поля – 20 мм;
- номер страницы ставится **снизу**, по центру;
- ориентация – книжная.

Основной текст

- абзац: первая строка – отступ **1,25** мм, междустрочный интервал – «**одинарный**», выравнивание – «**по ширине**»;
- шрифт – **Times New Roman**, **14 пт**;
- перенос слов – **автоматический**;
- выделять (жирным или курсивом) отдельные слова, словосочетания и предложения следует исходя из важности терминов.

ТРЕБОВАНИЯ К ИСХОДНОМУ КОДУ ПРОГРАММ

1. Программа должна быть разработана в **Visual Studio 2017-2022** или в **SharpDevelop** версии 4.3.0 на языке программирования C#, версия .NET 4.0.
2. Программа должна быть скомпилирована и иметь файл с расширением **.exe** для запуска приложения.
3. Имя архива «**Фамилия_Номер_группы_Номер_варианта**»

Пример: Иванов_25412_15.rar

РАБОТЫ, ОФОРМЛЕННЫЕ С НАРУШЕНИЕМ ТРЕБОВАНИЙ, ПРИНЯТЫ НЕ БУДУТ.

РАЗДЕЛ 4. ВСПОМОГАТЕЛЬНЫЙ

ПРОГРАММА ДИСЦИПЛИНЫ

Учебная программа по учебной дисциплине «Основы алгоритмизации и программирования» разработана для специальности 6-05-0612-01 «Программная инженерия».

Целью изучения учебной дисциплины является подготовка специалиста, уверенно владеющего возможностями, предоставляемыми современными компьютерными технологиями в среде программирования на алгоритмическом языке высокого уровня, а также программирования вычислительных алгоритмов. Изучение данной дисциплины является необходимым этапом в профессиональном развитии специалиста в области информационных технологий и позволяет в дальнейшем совершенствовать навыки разработки профессиональных программных средств, отвечающих современному этапу развития компьютерной техники.

Основными задачами преподавания учебной дисциплины являются: усвоение понятия алгоритма, его основных свойств, способов построения и записи алгоритмов, перевода их в конструкции языка программирования, а так же способов представления и анализа алгоритмов; изучение языка программирования высокого уровня, а также приобретение практических навыков составления и отладки программ на персональных компьютерах; приобретение навыков алгоритмизации на примерах решения вычислительных задач и их закрепление на основе программирования алгоритмов обработки структур данных и алгоритмов вычислительной математики; приобретение знаний об эффективности разрабатываемых алгоритмов, оценке их временных и вычислительных ресурсов.

Базовыми учебными дисциплинами по курсу «Основы алгоритмизации и программирования» являются «Математика» (в объеме уровня общего среднего образования), «Информатика» (в объеме уровня общего среднего образования). Знания и умения, полученные студентами при изучении данной дисциплины, необходимы для освоения последующих специальных дисциплин и дисциплин специализаций, связанных с разработкой программ, изучением технологий программирования и обработкой информации.

В результате изучения учебной дисциплины обучаемый должен:

знать:

– основы и современное состояние одного из алгоритмических языков высокого уровня;

– способы построения и представления алгоритмов;

– основные динамические структуры данных и алгоритмы их обработки;

– вычислительные алгоритмы решения инженерных задач;

– теоретические основы алгоритмизации и проектирования программ;

– принципы оценки вычислительной сложности и эффективности алгоритмов;

уметь:

– выполнять алгоритмизацию инженерных задач;

– реализовывать разработанный алгоритм в виде собственной программы на алгоритмическом языке или с использованием стандартных программ;

– применять разработанные программы в профессиональной деятельности;

владеть:

– современными средствами программирования;

– навыками анализа исходных и выходных данных решаемых задач и формами их представления;

– навыками отладки программ.

Освоение данной учебной дисциплины обеспечивает формирование следующих компетенций:

УК-2. Решать стандартные задачи профессиональной деятельности на основе применения информационно-коммуникационных технологий;

БПК-11. Быть способным применять научные подходы, концепции и методы, выработанные в рамках современных технических наук, для самостоятельного анализа теоретических проблем;

БПК-12. Быть способным анализировать источники информации, выделять наиболее существенные факты, давать им собственную оценку и интерпретацию, использовать на практике понятийно-категориальный аппарат, принятый в среде специалистов.

Согласно учебному плану учреждения высшего образования на изучение дисциплины отведено:

для заочной (дистанционной) формы получения высшего образования всего 216 часов, в том числе 56 часов аудиторных занятий;

для заочной (дистанционной) формы получения высшего образования, интегрированной со средним специальным образованием всего 108 часов, в том числе 28 часов аудиторных занятий;

Распределение аудиторных часов по курсам, семестрам и видам занятий приведено в таблицах 1 и 2.

Таблица 1.

Заочная (дистанционная) форма получения высшего образования					
Курс	Семестр	Лекции, ч.	Лабораторные занятия, ч.	Консультации по расписанию, ч.	Форма текущей аттестации
1	1	14	14		Экзамен
1	2	14	14		Экзамен

Таблица 2.

Заочная (дистанционная) форма получения высшего образования, интегрированная со средним специальным образованием					
Курс	Семестр	Лекции, ч.	Лабораторные занятия, ч.	Консультации по расписанию, ч	Форма текущей аттестации
1	2	14	14		Экзамен

СОДЕРЖАНИЕ УЧЕБНОГО МАТЕРИАЛА

Тема 1. Общие сведения об алгоритмах

Основные определения и понятия алгоритмизации вычислительных процессов. Алгоритм и его свойства. Разновидности структур алгоритмов. Способы описания алгоритмов. Стандартизация графического представления алгоритмов. Методы разработки и анализа алгоритмов. Общие сведения о структурном программировании. Представление структурированных схем. Примеры вычислительных алгоритмов.

Псевдокоды. Машина Тьюринга и вычислимость. Понятие универсальной машины Тьюринга. Тезис Тьюринга. Связь машин Тьюринга и вычислимости функций. Определение и виды вычислительной сложности. Невычислимые функции. Алгоритмически неразрешимые проблемы.

Тема 2. Системы программирования

Принципы и технология разработки программного обеспечения. Назначение и состав системы программирования. Возможности среды программирования. Подготовка программы к исполнению. Создание консольного приложения. Классификация языков программирования. Жизненный цикл программы. Примитивы, синтаксис, семантика.

Тема 3. Основные элементы языка

Основные понятия языка. Структура программы. Алфавит языка. Простые типы данных. Операции и их приоритет. Выражения. Основные операторы. Основные возможности организации ввода/вывода. Стандартные потоки ввода/вывода (ошибок). Примеры вычислительных алгоритмов. Операции отношения, логические, битовые операции. Условная операция, операция сдвига. Приоритет операций. Операции приведения типов. Условный оператор. Оператор перехода. Пустой оператор. Составной оператор. Операторы организации циклов. Оператор выбора, break, return, exit, continue. Вычисление сумм, рядов. Среда разработки.

Тема 4. Структуры данных

Массивы. Работа с массивами. Строки. Работа со строками. Структуры данных различного типа. Работа со структурами. Специфические типы данных. Указатели. Основные возможности работы с динамической памятью. Примеры вычислительных алгоритмов. Задачи поиска и сортировки.

Тема 5. Подпрограммы

Модульность в программировании. Понятие и структура подпрограммы. Описание подпрограмм в языках высокого уровня (процедуры, функции).

Организация вызова подпрограммы. Типы параметров подпрограммы; локальные и глобальные переменные. Формальные и фактические параметры. Передача массивов в качестве параметров подпрограмм. Процедурные типы. Внешние модули. Примеры вычислительных алгоритмов. Рекурсивные алгоритмы.

Тема 6. Файлы

Файлы. Основные возможности языка программирования для работы с файлами. Способы представления информации в файлах. Физическая и логическая организация файла. Прямой и последовательный доступ. Программная реализация алгоритмов работы с файлами. Библиотечные функции для работы с файлами. Примеры вычислительных алгоритмов.

Тема 7. Динамические структуры данных

Динамическое распределение памяти. Указатели и массивы. Работа с массивами при помощи указателей. Указатели и структуры. Динамическое распределение памяти. Указатели и функции. Организация динамических структур данных. Списки. Стеки. Очереди. Кольца. Организация данных в виде древовидных динамических структур. Двоичные деревья. Алгоритмы обработки динамических структур данных. Алгоритмы хеширования.

Тема 8. Алгоритмы вычислительной математики

Алгоритмы решения систем линейных алгебраических уравнений. Численное дифференцирование и интегрирование. Способы отыскания корней уравнений. Аппроксимация функций. Решение задач оптимизации. Алгоритмы поиска и сортировки. Пузырьковая сортировка. Сортировка выбором наименьшего элемента. Сортировка вставками. Рекурсивная реализация алгоритма быстрой сортировки. Вычисление определителя матрицы. Методы нахождения экстремума функций. Обыкновенные дифференциальные уравнения. Задачи Коши. Понятие метода сеток. Одношаговый метод Эйлера и его модификации, методы Рунге-Кутта. Решение систем дифференциальных уравнений первого порядка. Методы прогноза и коррекции, семейство многошаговых методов Адамса. Сравнение точности и скорости вычислений на основе различных вычислительных методов. Работа с матрицами (умножение, вычитание и т.д.). Динамическое программирование. Жадные алгоритмы. Решение NP-полных задач. Теория структурного программирования. Реализация основ структурного программирования в языках программирования.

УЧЕБНО-МЕТОДИЧЕСКАЯ КАРТА УЧЕБНОЙ ДИСЦИПЛИНЫ

заочная (дистанционная) форма получения высшего образования[1]

Номер раздела, темы	Название раздела, темы, занятия	Количество аудиторных часов					Количество часов УСР	Форма контроля знаний
		Лекции	Практические занятия	Лабораторные занятия	Консультации	Иное		
1	2	3	4	5	6	7	8	9
	1 семестр							
1.	Общие сведения об алгоритмах.	2						
	Лабораторное занятие №1. Графическое представление различных типов вычислительных процессов.			2				
3.	Основные элементы языка.	6						
	Лабораторное занятие №2. Ввод-вывод данных.			2				
	Лабораторное занятие №3. Программирование разветвляющихся вычислительных процессов.			2				
	Лабораторное занятие №4. Программирование циклических вычислительных процессов.			2				
4.	Структуры данных.	2						
	Лабораторное занятие №5. Программирование с использованием массивов.			2				
5.	Подпрограммы.	2						Контрольная работа
	Лабораторное занятие №6. Программирование с использованием подпрограмм.			2				
6.	Файлы.	2						
	Лабораторное занятие №7. Библиотечные функции для работы с файлами (открытие, закрытие, ввод/вывод, организация прямого доступа).			2				

[1] Темы учебного материала, не указанные в Учебно-методической карте, отводятся на самостоятельное изучение студента.

	Итого за семестр	14		14				Экзамен
	Всего аудиторных часов	28						

Номер раздела, темы	Название раздела, темы, занятия	Количество аудиторных часов					Количество часов УСР	Форма контроля знаний
		Лекции	Практические занятия	Лабораторные занятия	Консультации	Иное		
1	2	3	4	5	6	7	8	9
	2 семестр							
7.	Динамические структуры данных.	6						
	Лабораторное занятие №8. Работа с массивами при помощи указателей			2				
	Лабораторное занятие №9. Реализация хранения и преобразования информации представленной в виде стека.			2				
	Лабораторное занятие №10. Реализация хранения и преобразования информации представленной в виде кольца (одно/двунаправленного)			2				
8.	Алгоритмы вычислительной математики.	8						Контрольная работа
	Лабораторное занятие №11. Алгоритмы поиска и сортировки. Пузырьковая сортировка			2				
	Лабораторное занятие №12. Рекурсивная реализация алгоритма быстрой сортировки.			2				
	Лабораторное занятие №13. Работа с матрицами.			2				
	Лабораторное занятие №14. Решение задач оптимизации. Методы нахождения экстремума функций.			2				
	Итого за семестр	14		14				Экзамен
	Всего аудиторных часов	28						

УЧЕБНО-МЕТОДИЧЕСКАЯ КАРТА УЧЕБНОЙ ДИСЦИПЛИНЫ
заочная (дистанционная) форма получения высшего образования, интегрированная со средним специальным образованием

Номер раздела, темы	Название раздела, темы, занятия	Количество аудиторных часов					Количество часов УСР	Форма контроля знаний
		Лекции	Практические занятия	Лабораторные занятия	Консультации	Иное		
1	2	3	4	5	6	7	8	9
	2 семестр							
7.	Динамические структуры данных.	6						
	Лабораторное занятие №8. Работа с массивами при помощи указателей			2				
	Лабораторное занятие №9. Реализация хранения и преобразования информации представленной в виде стека.			2				
	Лабораторное занятие №10. Реализация хранения и преобразования информации представленной в виде кольца (одно/двунаправленного)			2				
8.	Алгоритмы вычислительной математики.	8						Контрольная работа
	Лабораторное занятие №11. Алгоритмы поиска и сортировки. Пузырьковая сортировка			2				
	Лабораторное занятие №12. Рекурсивная реализация алгоритма быстрой сортировки.			2				
	Лабораторное занятие №13. Работа с матрицами.			2				
	Лабораторное занятие №14. Решение задач оптимизации. Методы нахождения экстремума функций.			2				
	Итого за семестр	14		14				Экзамен
	Всего аудиторных часов	28						

ИНФОРМАЦИОННО-МЕТОДИЧЕСКАЯ ЧАСТЬ

Список литературы

Основная литература

1. Программирование на языке Delphi : учебное пособие / А. Н. Вальвачев, К. А. Сурков, Д. А. Сурков, Ю. М. Четырько. – [Электронный ресурс]. – Режим доступа: https://www.bsuir.by/m/12_103607_1_90135.pdf. – Дата доступа: 16.03.2022.
2. Вирт, Н. Алгоритмы и структуры данных / Н. Вирт. – Санкт-Петербург : Невский Диалект, 2001. – 352 с.
3. Голицына, О. Л. Основы алгоритмизации и программирования: учеб. пособие / О. Л. Голицына, И. И. Попов. – Москва : ФОРУМ, 2008. – 432 с.
4. Златопольский, Д. М. Сборник задач по программированию / Д. М. Златопольский. – Санкт-Петербург : БХВ-Петербург, 2007. – 304 с.
5. Керниган, Б. Язык программирования С / Б. В. Керниган, Д. Ритчи. – 2-е изд., перераб. и доп. – Москва : Вильямс, 2009. – 304 с.
6. Котов, В. М. Структуры данных и алгоритмы. Теория и практика : учеб. пособие / В. М. Котов, Е. П. Соболевская. – Минск : БГУ, 2004. – 267 с.
7. Кочан, С. Программирование на языке С / С. Кочан. – 3-е изд. – Москва : Вильямс, 2007. – 496 с.
8. Уилсон, С. Принципы проектирования и разработки программного обеспечения. Учебный курс. – Санкт-Петербург : Питер, 2003.
9. Фаронов, В. В. Турбо Паскаль 7.0. Учебный курс / В. В. Фаронов. – Москва : Кнорус, 2011. – 368 с.
10. Хусаинов, Б. С. Структуры и алгоритмы обработки данных. Примеры на языке Си / Б. С. Хусаинов. – Москва : Финансы и статистика, 2004. – 464 с.
11. ГОСТ 19.701-90 – Единая система программной документации – Схемы алгоритмов, программ, данных и систем – Условные обозначения и правила выполнения.
12. Руководство по Object Pascal для Delphi 10.4 Sydney Марко Канту [Электронный ресурс]. – Режим доступа: https://lp.embarcadero.com/RU-ObjectPascalEbook?utm_source=whitepaper-RU&utm_medium=Partner&utm_ content=ObjectPascalHandbook2021-RU. – Дата доступа: 08.04.2021.
13. Алгоритмы: построение и анализ / Т. Кормен [и др.]. – Москва : Вильямс, 2019. – 1328 с.

14. Навроцкий, А. А. Основы алгоритмизации и программирования в среде Visual C++ : учебно-метод. пособие / А. А. Навроцкий. – Минск : БГУИР, 2014. – 160 с. : ил.

Дополнительная литература

1. Архангельский, А. Я. Программирование в C++ Builder 6 / А. Я. Архангельский. – 2-е изд. – Москва : Бином, 2005. – 1168 с.
2. Батура, М. П. Основы алгоритмизации и программирования. Язык Си : учебное пособие [доп. МО РБ] / М. П. Батура [и др.]. – 2-е изд. – Минск : БГУИР, 2008. – 240 с.
3. Демидович, Е. М. Основы алгоритмизации и программирования. Язык Си / Е. М. Демидович. – Санкт-Петербург : БХВ-Петербург, 2006. – 440 с.
4. Шупляк, В. И. C++. Практический курс : учеб. пособие / В. И. Шупляк. – Минск : Новое знание, 2008. – 576 с.
5. Шилдт, Г. Искусство программирования на C++ / Г. Шилдт. – Санкт-Петербург : БХВ-Петербург, 2005. – 496 с.
6. Страуструп, Б. Язык программирования C++ / Б. Страуструп. – Москва : Бином, 2012. – 1104 с.
7. Род, С. Delphi. Готовые алгоритмы / С. Род. – Москва : ДМК-Пресс, 2001. – 745 с.
8. Колосов, С. В. Программирование в среде Delphi : учеб. пособие / С. В. Колосов. – Минск : БГУИР, 2005. – 166 с.
9. Кнут, Д. Искусство программирования. Т. 1–3 / Д. Кнут. – Москва : Вильямс, 2004. – 486 с.
10. Гленн Брукшир, Дж. Введение в компьютерные науки / Дж. Гленн Брукшир. – Москва ; Санкт-Петербург ; Киев : Вильямс, 2001. – 688 с.
11. Бахвалов, Н. С. Численные методы в задачах и упражнениях / Н. С. Бахвалов, А. В. Лапин, Е. В. Чижонков. – Москва : Высшая школа, 2000. – 190 с.
12. Соловьев, В. П. Основы численных методов : учеб.-метод. пособие / В. П. Соловьев, Т. М. Кривоносова, В. Л. Смирнов. – Минск : БГУИР, 2011. – 131 с.
13. Бхаргава, А. Грокаем алгоритмы / А. Бхаргава. – Санкт-Петербург : Питер, 2017. – 288 с.
14. Луцик, Ю. А. Основы алгоритмизации и программирования [+ электр. вариант] : язык Си : учебно-методическое пособие / Ю. А. Луцик, А. М. Ковальчук, Е. А. Сасин. – Минск : БГУИР, 2015. – 169 с.
15. Лафоре, Р. Объектно-ориентированное программирование в C++ [+ электр. вариант] / Р. Лафоре. – 4-е изд. – Санкт-Петербург : Питер, 2016. – 928 с.

Средства диагностики результатов учебной деятельности

Оценка уровня знаний студента производится по десятибалльной шкале в соответствии с критериями, утвержденными Министерством образования Республики Беларусь.

Для оценки достижений студента рекомендуется использовать следующий диагностический инструментарий:

- коллоквиум;
- устный опрос;
- письменная самостоятельная работа;
- контрольная работа;
- отчеты по аудиторным/домашним практическим упражнениям с их устной защитой;
- сдача экзамена по дисциплине.

Перечень контрольных вопросов и заданий для самостоятельной работы студентов

1. Алгоритм. Свойства алгоритма.
2. Алфавит языка.
3. Структура программы.
4. Идентификаторы. Переменные.
5. Константы. Виды констант.
6. Типы данных языка.
7. Порядковые типы данных. Стандартные подпрограммы, обрабатывающие порядковые типы данных.
8. Операции. Арифметические операции. Арифметические выражения.
9. Стандартные арифметические функции. Порядок вычислений.
10. Преобразование типов данных. Стандартные функции преобразования типов данных.
11. Ввод с консоли. Вывод на консоль. Форматированный вывод.
12. Комментарии. Простейшие операторы языка.
13. Условный оператор if-else.
14. Оператор выбора switch.
15. Метки и безусловный переход. Операторы break и continue.
16. Оператор циклов for.
17. Операторы циклов while, do-while.

18. Массивы. Описание переменных размерностей. Обращение к компонентам массива.
19. Одномерные массивы.
20. Многомерные массивы. Задание массива константой.
21. Символы. Символ-константа. Операции с символами. Стандартные функции для работы с символами.
22. Строки. Строка-константа. Операции со строками. Стандартные функции и процедуры обработки строк.
23. Множества. Множество-константа. Операции со множествами.
24. Функции. Объявление функции. Описание функции. Возвращаемые значения. Вызов функции.
25. Функции. Способы постановки аргументов.
26. Передача параметров по значению и по ссылке.
27. Функции с переменным числом параметров.
28. Параметры функции main.
29. Перегрузка функций.
30. Шаблоны функций.
31. Процедуры. Объявление процедуры. Описание процедуры. Возвращаемое значение. Вызов процедуры.
32. Процедуры. Способы постановки аргументов.
33. Записи. Описание. Задание записей константой. Доступ к полям записи.
34. Записи. Оператор with.
35. Файлы. Разновидность файлов. Описание файлов.
36. Текстовые файлы. Назначение текстовых файлов. Открытие и закрытие файла.
37. Текстовые файлы. Назначение текстовых файлов. Запись в файл. Считывание из файла.
38. Типизированные файлы. Описание типизированных файлов. Назначение типизированных файлов. Открытие и закрытие файлов.
39. Типизированные файлы. Назначение типизированных файлов. Запись в файл. Считывание из файла. Поиск в типизированном файле.
40. Нетипизированные файлы. Описание нетипизированных файлов. Назначение нетипизированных файлов. Открытие и закрытие файлов.

41. Нетипизированные файлы. Назначение нетипизированных файлов. Запись в файл. Считывание из файла. Поиск в нетипизированном файле.
42. Модульность программ. Стандартные модули языка Си. Подключение модулей.
43. Структура модуля. Секция внешних связей. Секция реализации. Секция инициализации.
44. Взаимодействие модулей. Компиляция модулей.
45. Указатели в Си.
46. Ссылки в Си.
47. Связь указателей и массивов в Си.
48. Структура как тип данных в Си.
49. Указатели и структуры в Си.
50. Структура как параметр функции.
51. Объединения в Си.
52. Перечисления в Си.
53. Функции динамического распределения памяти.

Методические рекомендации по организации и выполнению самостоятельной работы студентов

При изучении дисциплины рекомендуется использовать следующие формы самостоятельной работы:

- работа с учебной и справочной литературой;
- составление конспектов;
- решение задач и выполнение упражнений;
- работа с раздаточным материалом.