Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Институт повышения квалификации
и переподготовки

Кафедра «Профессиональная переподготовка»

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ НА ЯЗЫКАХ ВЫСОКОГО УРОВНЯ

ПОСОБИЕ
для слушателей специальности переподготовки
1-40 01 73 «Программное обеспечение
информационных систем»
заочной формы обучения

Гомель 2021

УДК 004.921(075.8)
ББК 32.973-018.2я73
О-75

Рекомендовано кафедрой «Профессиональная переподготовка»
ИПКиП ГГТУ им. П. О. Сухого
(*протокол № 9 от 28.05.2021 г.*)

Рецензент: доц. каф. «Информатика» ГГТУ им. П. О. Сухого
канд. техн. наук, доц. *Т. А. Трохова*

О-75 **Основы** алгоритмизации и программирования на языках высокого уровня : пособие для слушателей специальности переподготовки 1-40 01 73 «Программное обеспечение информационных систем» заоч. формы обучения / сост. Е. И. Гридина – Гомель : ГГТУ им. П. О. Сухого, 2021. – 65 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://elib.gstu.by. – Загл. с титул. экрана.

Данное пособие написано в соответствии с требованиями, предъявленными к оформлению методических пособий, доступным языком и содержит множество примеров, позволяющих в кратчайшие сроки овладеть основными принципами алгоритмизации и реализации алгоритмов на языках высокого уровня.

Для слушателей специальности переподготовки 1-40 01 73 «Программное обеспечение информационных систем» ИПКиП.

УДК 004.921(075.8)
ББК 32.973-018.2я73

Содержание

Понятие алгоритмов. Основные принципы алгоритмизации

Понятие алгоритма занимает центральное место в современной математике и программировании.

Алгоритмизация – сведение задачи к последовательным этапам действий так, что результаты предыдущих действий используются при выполнении последующих.

При разработке алгоритма выделяют основные положения:

1. Действие – это некоторая операция, имеющая конкретную продолжительность и приводящая к совершенно конкретному результату.

2. Каждое действие предполагает наличие некоторых данных, над которыми это действие совершается, и по изменению состояния которых определяют результат этого действия.

3. Каждое действие должно быть таким, чтобы его можно было описать при помощи какого-либо языка (или набора формул); такое описание называют инструкция.

4. Если действие можно разложить на составные части, то его называют процессом (или вычислением).

5. Описание характера проведения процесса, т.е. последовательности выполняемых действий без привязки к какому-то конкретному процессору, называют алгоритмом.

Числовой алгоритм – детально описанный способ преобразования числовых входных данных в выходные при помощи математических операций. Существуют нечисловые алгоритмы, которые используются в экономике, технике и научных исследованиях.

Алгоритм – строгий и четкий набор правил, определяющий последовательность действий, приводящих к достижению поставленной цели.

Определим основные свойства алгоритмов.

Дискретность – значения новых величин (данных) вычисляются по определенным правилам из других величин с уже известными значениями.

Определенность (детерминированность) – каждое правило из набора однозначно, а сами данные однозначно связаны между собой, т.е. последовательность действий алгоритма строго и точно определена.

Результативность (конечность) – алгоритм решает поставленную задачу за конечное число шагов.

Массовость – алгоритм разрабатывается так, чтобы его можно было применить для целого класса задач, например, алгоритм вычисления определенных интегралов с заданной точностью.

Выполнение любого алгоритма требует определенного объема памяти компьютера для размещения данных и программы, а также времени по обработке этих данных – эти ресурсы ограничены и, следовательно, правомочен вопрос об эффективности их использования.

Таким образом, в самом широком смысле понятие эффективности связано со всеми вычислительными ресурсами, необходимыми для работы алгоритма.

Однако обычно под «самым эффективным» понимается алгоритм, обеспечивающий наиболее быстрое получение результата, поэтому рассмотрим именно временну́ю сложность алгоритмов.

Время работы алгоритма удобно выражать в виде функции от одной переменной, характеризующей «размер» конкретной задачи, т.е. объем входных данных, необходимых для ее решения. Тогда сравнительная сложность задач и может оцениваться через ее размер.

Поскольку описание задачи, предназначенной для решения посредством вычислительного устройства, можно рассматривать в виде слова конечной длины, представленной символами конечного алфавита, в качестве формальной характеристики размера задачи можно принять длину входного слова. Например, если стоит задача определения максимального числа в некоторой последовательности из n элементов, то и размер задачи будет n, поскольку любой вариант входной последовательности можно задать словом из n символов.

Графическое изображение алгоритма – это представление его в виде схемы, состоящей из последовательности блоков (геометрических фигур), каждый из которых отображает содержание очередного шага алгоритма. А внутри фигур кратко записывают действие, выполняемое в этом блоке. Такую схему называют блок-схемой или структурной схемой алгоритма, или просто схемой алгоритма.

Правила изображения фигур сведены в единую систему программной документации (дата введения последнего стандарта ГОСТ 19.701.90 – 01.01.1992).

По данному ГОСТу графическое изображение алгоритма – это схема данных, которая отображает путь данных при решении задачи и определяет этапы их обработки.

Схема данных состоит из следующих элементов:

– символов данных (символы данных могут отображать вид носителя данных);

– символов процесса, который нужно выполнить над данными;

– символов линий, указывающих потоки данных между процессами и носителями данных;

– специальных символов, которые используют для облегчения чтения схемы алгоритма.

Основные символы для изображения схемы алгоритма представлены в приложении А.

Виды алгоритмов

Любую программу можно разбить на блоки, реализованные в виде алгоритмов (процессов), которые можно разделить на три вида:

1) линейные (единственное направление выполнения);

2) разветвляющиеся (направление выполнения определяет условие);

3) циклические (отдельные участки вычислений выполняются многократно).

Любой циклический процесс включает в себя участок с разветвлением и может быть простым и сложным (вложенным).

Для решения вопроса о том, сколько раз нужно выполнить цикл, используется анализ переменной, которую называют параметром цикла.

Циклический процесс, в котором количество повторений заранее известно, называется циклом по счетчику, а циклический процесс, в котором количество повторений заранее неизвестно и зависит от получаемого в ходе вычислений результата, называют итерационным.

Базовые виды алгоритмов можно классифицировать на :

– линейные;

– разветвляющиеся;

– циклические.

Линейным называется алгоритм, каждая команда которого выполняется за один проход алгоритма. Общая графическая схема линейного алгоритма приведена на рис.1.1.

Рис. 1.1. Линейный алгоритм

Разветвляющийся алгоритм – это алгоритм, выполнение команд которого зависит от выполнения или невыполнения некоторого условия. Разветвляющийся алгоритм может иметь полную и краткую формы (рис. 1.2).

Циклическим называется алгоритм, в котором группа команд записывается один раз, а выполняться может множество раз.

Различают:

– циклы с предусловием, когда условие повторения цикла проверяется раньше, чем выполняется рабочая часть цикла;

– циклы с постусловием, когда проверка условия повторения цикла выполняется после выполнения рабочей части цикла.

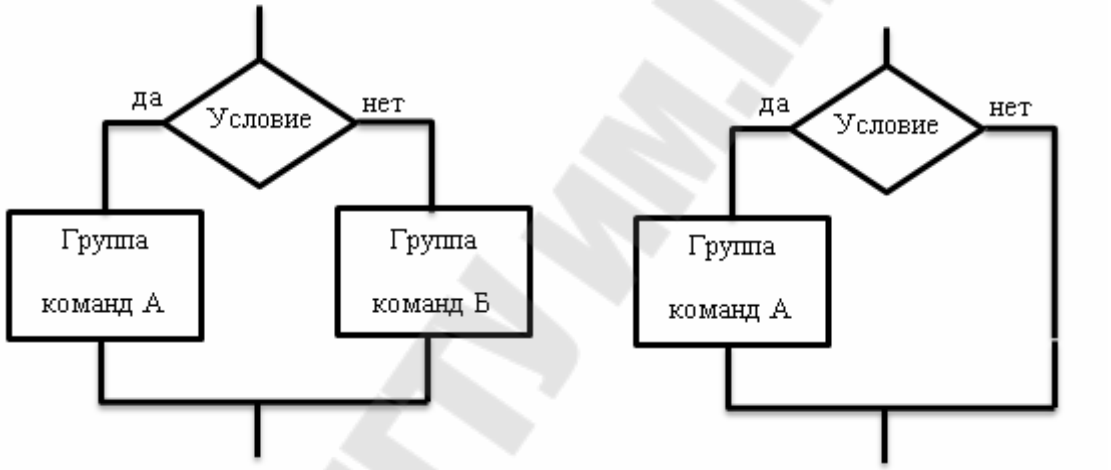


Рис. 1.2. Формы разветвляющегося алгоритма (полная и краткая)

Порядок выполнения циклического алгоритма с предусловием таков:

– проверяется выполнение условия повторения цикла, и, пока оно выполняется, то выполняется рабочая часть цикла;

– если условие не выполняется, то выполняется следующий за циклом шаг алгоритма.

Для циклов с постусловием порядок выполнения алгоритма таков:

– выполняется рабочая часть цикла;

– проверяется выполнение условия повторения цикла, и, пока оно выполняется, выполняется рабочая часть цикла;

– если условие не выполняется, то выполняется следующий за циклом шаг алгоритма.

Из описания процесса выполнения циклических алгоритмов видно, что цикл с постусловием всегда выполняется хотя бы один раз, а цикл с предусловием может не выполниться ни разу. На рис. 1.3 приведены графические схемы циклических алгоритмов.

Часто встречаются циклические алгоритмы, в которых присутствует величина, значение которой изменяет ход выполнения алгоритма, такая величина называется переменной цикла.

При решении научных и инженерных задач чаще всего используют смешанные алгоритмы. Смешанными называются алгоритмы, которые содержат сочетания линейных, разветвляющихся и циклических алгоритмов.

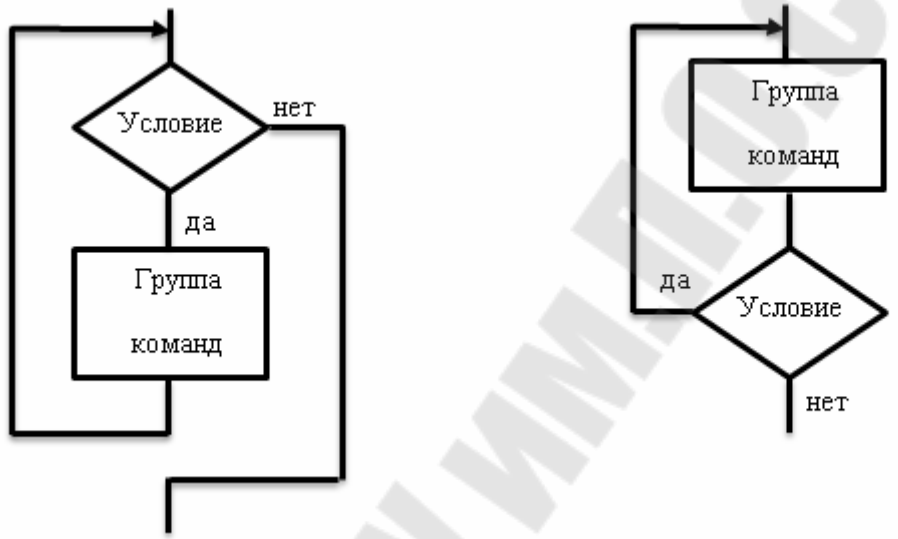


Рис. 1.3. Формы циклического алгоритма
(с предусловием и постусловием)

Среди многообразия типовых алгоритмов решения инженерных задач можно условно выделить такие виды алгоритмов, как:

- алгоритмы вычисления с накоплением;
- поисковые алгоритмы;
- алгоритмы формирования и преобразования данных.

К алгоритмам вычисления с накоплением условно можно отнести алгоритмы поиска суммы, произведения, количества элементов массивов, отвечающих определенным условиям, например, положительных, отрицательных, равных заданному числу и т.д.

К поисковым алгоритмам относятся алгоритмы поиска элементов, соответствующих определенному условию или критерию в массиве данных, например, поиск минимального или максимального элемента массива, поиск элемента, большего или меньшего заданного порогового значения или элемента, попадающего в интервал заданных числовых значений.

К алгоритмам формирования и преобразования данных можно отнести алгоритмы, решающие задачи о табулировании функций,

формировании новых массивов из уже существующих, изменения части массива, дополнения массивов и т.д.

Наиболее часто в практике программирования требуется организовать расчет некоторого арифметического выражения при различных исходных данных. Например, такого (1):

$$z = \frac{tg^2 x}{\sqrt{x^2 + m^2}} + x^{(m+1)} \sqrt{x^2 + m^2} \ ,$$

где x > 0 – вещественное, m – целое.

Разработка алгоритма обычно начинается с составления схемы. Продумывается оптимальная последовательность вычислений, при которой, например, отсутствуют повторения. При написании алгоритма рекомендуется переменным присваивать те же имена, которые фигурируют в заданном арифметическом выражении либо иллюстрируют их смысл.

Для того чтобы не было «длинных» операторов, исходное выражение полезно разбить на ряд более простых. В нашей задаче предлагается схема вычислений, представленная на рис. 1.4.

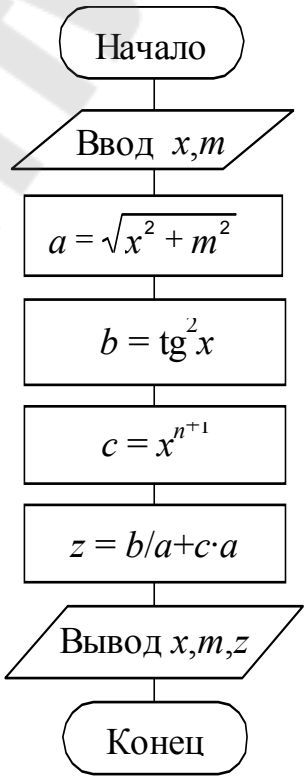


Рис. 1.4. Схема линейного процесса

Она содержит ввод и вывод исходных данных, линейный вычислительный процесс, вывод полученного результата. Заметим, что выражение $\sqrt{x^2+m^2}$ вычисляется только один раз. Введя дополнительные переменные *a*, *b*, *c*, мы разбили сложное выражение на ряд более простых.

Задачи линейной алгоритмизации

Теоретические сведения по изучению темы:

1.Сущность алгоритма линейной структуры.
2.Типы данных и объявления данных.
3.Операции. Приоритет операций.
4.Функции getchar() и putchar().
5.Понятие буферизации.
6.Функция printf().
7.Функция scanf().
8.Оператор присваивания.

Изложение вопросов:

Под алгоритмом линейной структуры понимают алгоритм, включающий в себя прямую конечную последовательность действий Алгоритм линейной последовательности является наиболее простым.

В языке С различают следующие типы данных :

int – задает значения, к которым относятся целые числа.

short – в 16-разрядных системах short=int ; в 32-х-short=16 бит.

long – для 16-разрядных систем диапазон значений лежит от – 2147483648 до 2147483647; занимает переменная 32 бита; для 32-разрядных ЭВМ long = int.

char – задает значения, которые представляют различные символы. Например: w, y, 4, ! и т. д. Такими символами могут быть почти все знаки из кодовой таблицы ЭВМ .

Исключение составляют лишь специальные управляющие знаки, не имеющие графического изображения. И в 16-разрядных и в 32-разрядных ЭВМ данные типа char занимают в памяти 8 бит или 1 байт.

unsigned – в языке С можно задавать некоторые типы как unsigned . К этим типам относятся char, short, int, long.

Такое задание означает, что соответствующие переменные не будут иметь отрицательных значений. В результате они могут принимать большие положительные значения, чем переменные обычных типов. Например, задание unsigned int может принимать значения от 0 до 65535 при том же размере, что и int (16 бит).

Замечание: в случае типа int запись вида unsigned int a; можно записывать более сжато: unsigned a .

float – задает значения, к которым относятся вещественные числа, имеющие дробную часть. Например, 5.27; -1.58e+2; 3.61e-4. И в 16-ти и в 32-разрядных системах занимает 32 бита. Значения находятся в диапазоне от ±3.4e-38 до ±3.4e+38.

double – тип float с двойной точностью .

Занимают в два раза больше места, чем тип float, то есть 64 бита.

Диапазон представления от ±1.7e-308 до ±1.7e+308.

enum – предназначен для описания объектов из некоторого заданного множества , например , {весна, лето, осень, зима}.

void – используется для обозначения величин , имеющих нулевую длину и не имеющих значения (не определенных).

В языке Си все данные должны быть объявлены раньше, чем будут использоваться. Объявление данных осуществляется в операторах объявления, которые определяют тип и список, состоящий из одной или более переменных этого типа . Например:

```
int lower, upper, step;
char ch , line[100];
enum tip _ year{весна, лето, осень, зима} a, b, c;
```

При объявлении переменная может быть инициализирована, например:

```
char esc =’\\’;
int i=0 ;
int limit = MAXLINE+1;
float eps = 1.0e-5;
```

К любой переменной при объявлении может быть применен квалификатор const для указания того, что ее значение далее в программе изменяться не будет. Например: const double e = 2.71828182845905;

const char msg [] = “Предупреждение”;

Замечание: Реакция на попытку изменить переменную, помеченную квалификатором const оставляется обычно на усмотрение разработчиков компилятора.

Правила при объявлении переменных:

1. Внешние и статические переменные, по умолчанию инициализируются нулем .
2. Автоматические переменные, явным образом не инициализированные, содержат неопределенные значения (мусор).
3. В Си необходимо различать порядок вычисления операций и приоритет (старшинство) операций.

Порядок – это последовательность вычислений (слева направо, справа налево). Приоритет – это то, какая операция выполняется в выражении первой, какая второй и т. д. В табл. 1 перечислены все операции Си и Си++ с указанием их приоритета, начиная с высшего и

кончая низшим, и показан порядок вычисления (слева направо или справа налево). Все операции, указанные внутри строк, имеют одинаковый приоритет.

Функция getchar() получает один символ, поступающий с устройства стандартного ввода-вывода (клавиатура) и передает его выполняющейся в данный момент программе.

Функция putchar () получает один символ, поступающий из программы и пересылает его для вывода на экран.

Функция getchar() аргументов не имеет. Она получает символ и возвращает его программе. Функция putchar() имеет один аргумент, который представляет собой символ, выводимый на печать. Аргументом функции putchar() могут быть:

– одиночный символ, включая знаки, представляемые управляющими последовательностями;

– переменная, значением которой является одиночный символ;

– функция, значением которой является одиночный символ.

Буфер – это некоторая область оперативной памяти, организуемая временно для помещения в нее информации. Если буфер заполнится, содержимое его передается по назначению, и процесс буферизации начинается снова.

Функции стандартного ввода-вывода – это функции с буферизацией, то есть не сразу осуществляют ввод-вывод, а через буфер.

Функции printf() и scanf() дают возможность взаимодействовать с программой на уровне стандартного ввода-вывода.

Обычно функции printf() и scanf() работают во многом одинаково – каждая использует управляющую строку и список аргументов. Вначале рассмотрим работу функции printf(),а затем scanf().

Функция printf() осуществляет форматный вывод на устройство стандартного вывода (дисплей). Общий вид обращения к функции printf() следующий: printf(управляющая строка, аргумент 1, аргумент 2,…);

Где аргумент 1, аргумент 2 и т.д. – выводимые параметры, в качестве которых могут быть: переменные; константы; выражения, которые вычисляются перед выводом.

Управляющая строка - строка символов, показывающая, как должны быть выведены параметры. Управляющая строка обрамляется кавычками.

Пример:

printf(“%d женщин съели %d плиток шоколада.\n”, number, shok);

Управляющая строка содержит информацию двух различных видов: символы, выводимые текстуально; идентификаторы данных, которые называются также спецификациями преобразования.

Правило: каждому аргументу из списка, следующего за управляющей строкой, должна соответствовать одна спецификация преобразования.

Замечание: поскольку символ % используется в функции printf() в спецификации преобразования, то если нужно вывести сам символ %, просто надо написать два символа % подряд.

Пример: printf("Только %d %% стряпни Анны было съедено.\n",pe);

Спецификации преобразования могут содержать внутри себя модификаторы. Они помещаются между знаком % и символом, определяющим тип преобразования.

Каждому типу выводимой информации соответствует своя спецификация преобразования.

Функция scanf() во многом идентична функции printf(), с той лишь разницей, что она осуществляет форматное чтение с устройства стандартного ввода (клавиатуры). Так же, как и функция printf() она имеет управляющую строку и список аргументов. Главное различие двух этих функций заключается в особенностях списка аргументов. Функция prihtf() использует в качестве аргументов переменные, константы, выражения, в то время как scanf() – только указатели на переменные. При применении функции scanf() надо соблюдать два правила:

1) при вводе значения некоторой переменной в качестве аргумента используется не имя переменной, а адрес этой переменной в памяти, то есть указатель на переменную, поэтому перед именем переменной ставится символ операции получения адреса &;

2) если вводится значение строковой переменной, использование символа & необязательно, ибо строка сама указывает на себя в памяти.

Функция scanf() использует некоторые специальные знаки (пробелы, табуляции, "enter") для разбиения входного потока символов на отдельные поля. Она согласует последовательность спецификаций преобразования с последовательностью полей, опуская упомянутые специальные знаки между ними. Единственным исключением из этого является спецификация %c, которая обеспечивает чтение каждого следующего символа даже в том случае, если это пустой символ.

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

1) отсутствует спецификация %g;

2) спецификации %f и %e эквиваленты;

3) для чтения целых чисел типа short применяется %h;

Оператор присваивания – это основная рабочая сила большинства программ. С помощью этого оператора переменной присваивается некоторое значение.

Общий вид оператора присваивания:
N=T; где N-имя переменной ;
T-выражение.

Задачи

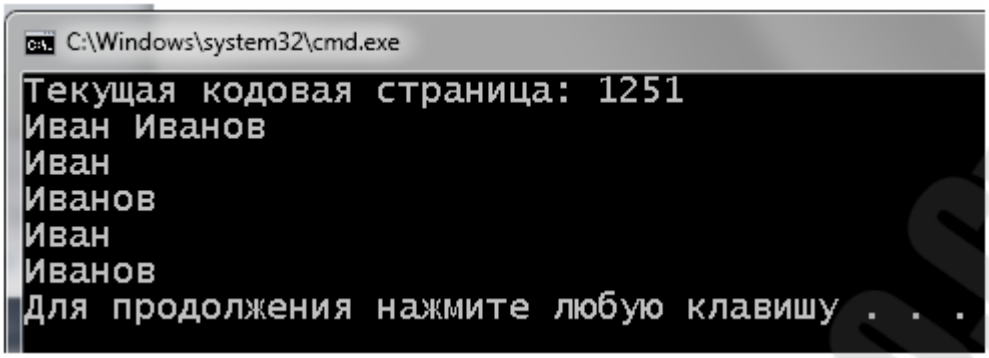
1. Напишите программу, которая использует вызов функции printf () для печати вашего имени и фамилии в одной строке, использует второй вызов функции printf(), чтобы напечатать ваше имя и фамилию в двух строках, и использует два вызова функции printf () для печати вашего имени и фамилии в одной строке. Выходные данные должны иметь следующий вид (при этом используются ваши персональные данные):

Иван Иванов ← Первый оператор печати
Иван ← Второй оператор печати
Иванов ← Все еще второй оператор печати
Иван Иванов ← Третий и четвертый операторы печати

Фрагмент программы

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

int main(int argc, _TCHAR* argv[])
{
        system("chcp 1251");            //Смена кодировки страницы
        printf("Иван Иванов \n");   //Первый оператор печати
        printf("Иван \nИванов \n"); //Второй оператор печати
        printf("Иван \n");              //Третий оператор печати
        printf("Иванов \n");            //Четвертый оператор печати

        return 0;
}
```

Результат работы программы

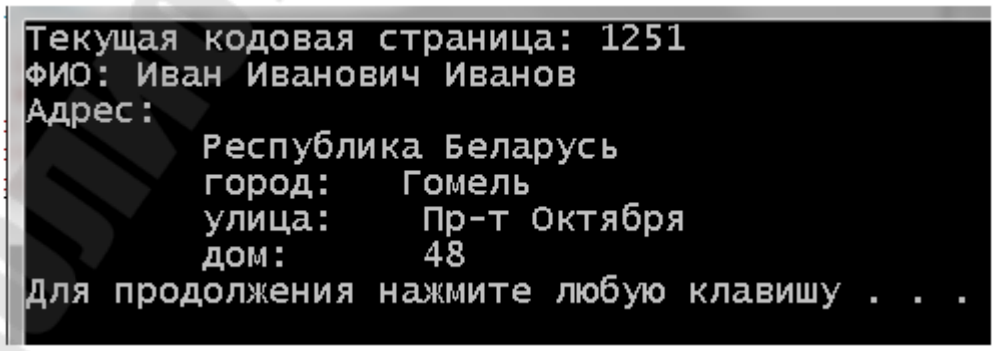


2. Напишите программу, печатающую ваше имя и адрес.

Фрагмент программы

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

int main(int argc, _TCHAR* argv[])
{
system("chcp 1251");            //Смена кодировки страницы
printf("ФИО: Иван Иванович Иванов  \n");   //Первый оператор печати
printf("Адрес: \n"); //Второй оператор печати
printf("\tРеспублика Беларусь \n");       //Третий оператор печати
printf("\tгород:\t Гомель \n");            //Четвертый оператор печати
printf("\tулица:\t  Пр-т Октября \n");     //Четвертый оператор печати
printf("\tдом:\t  48 \n");
return 0;
}
```

Результат работы программы



3. Напишите программу, которая преобразует ваш возраст в годах в количество дней и отображает на экране оба значения. На этой стадии можно учитывать только прожитые годы и не учитывать високосные года.

.

Фрагмент программы

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

int main(int argc, _TCHAR* argv[])
{
	system("chcp 1251");		//Смена кодировки страницы

	int age;			// Переменная целого типа
	printf("Введите количество полных лет ");
	scanf_s("%d",&age);				// Ввод данных
	printf("Вам %d лет, что составляет %d дней \n",age,age*365);
	return 0;
}
```

Результат работы программы

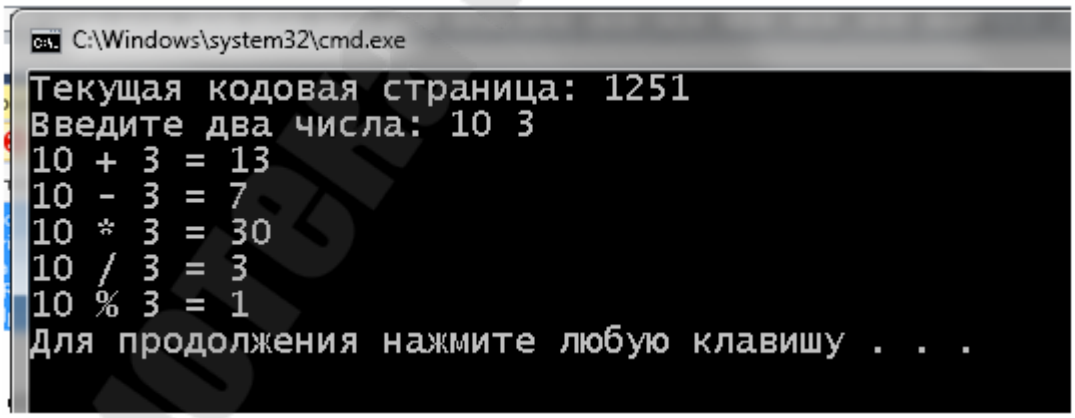
```
Текущая кодовая страница: 1251
Введите количество полных лет 25
Вам 25 лет, что составляет 9125 дней
Для продолжения нажмите любую клавишу . . .
```

4. Ввести с клавиатуры два числа, а затем вывести на экран результат вычисления суммы, разницы, умножения, деления и взятия остатка от деления первого на второе.

Фрагмент программы

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

int main(int argc, _TCHAR* argv[])
{
	system("chcp 1251");		//Смена кодировки страницы

	int a, b;
	printf("Введите два числа: ");
	scanf_s("%d %d", &a, &b);
	printf("%d + %d = %d\n", a, b, a + b);
	printf("%d - %d = %d\n", a, b, a - b);
	printf("%d * %d = %d\n", a, b, a * b);
	printf("%d / %d = %d\n", a, b, a / b);
	printf("%d %% %d = %d\n", a, b, a % b);
	return 0;
}
```

Результат работы программы

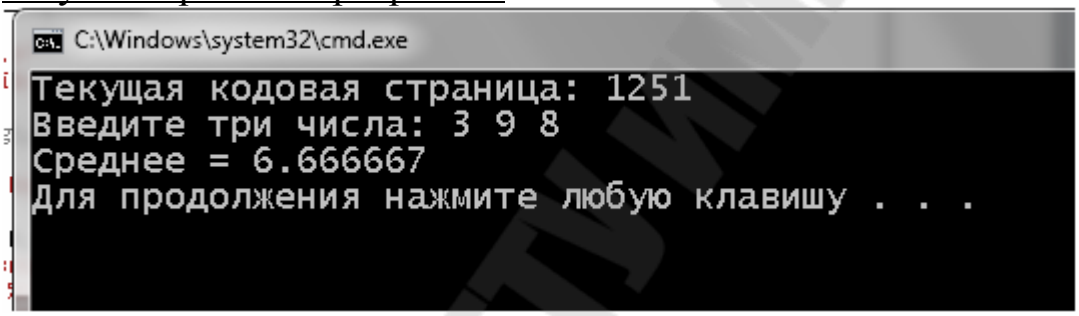


5. Ввести три числа, а затем вывести их среднее арифметическое.

Фрагмент программы

```
#include "stdio.h"
#include <cstdlib>
int main(int argc, _TCHAR* argv[])
{
	system("chcp 1251");		//Смена кодировки страницы
	float a, b, c;
	printf("Введите три числа: ");
	scanf_s("%f %f %f", &a, &b, &c);
	printf("Среднее = %f\n", (a + b + c) / 3);
	return 0;
}
```

Результат работы программы

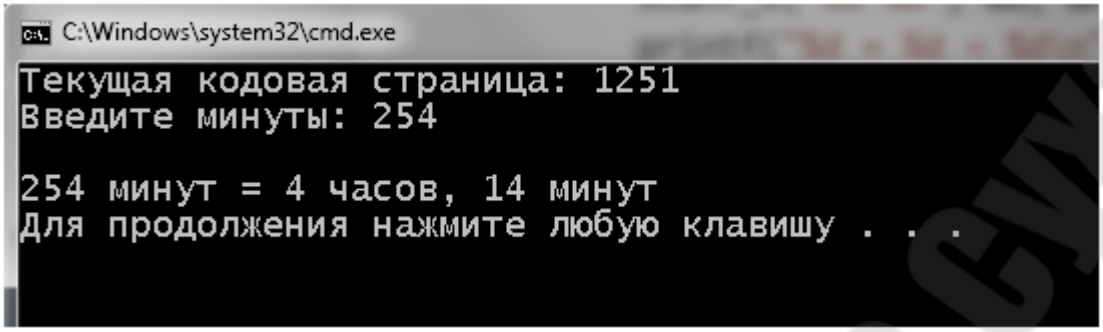


6. Ввести с клавиатуры количество минут и напечатать количество целых часов и оставшихся минут. Например, 69 минут => 1 час и 9 минут, 45 минут => 0 часов и 45 минут, 254 минуты => 4 часа и 14 минут.

Фрагмент программы

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

int main(int argc, _TCHAR* argv[])
{
	system("chcp 1251");		//Смена кодировки страницы

	int minutes;
	printf("Введите минуты: ");
	scanf_s("%d", &minutes);
	printf("\n%d минут = %d часов, %d минут\n", minutes,minutes / 60, minutes % 60);
	return 0;
}
```

Результат работы программы



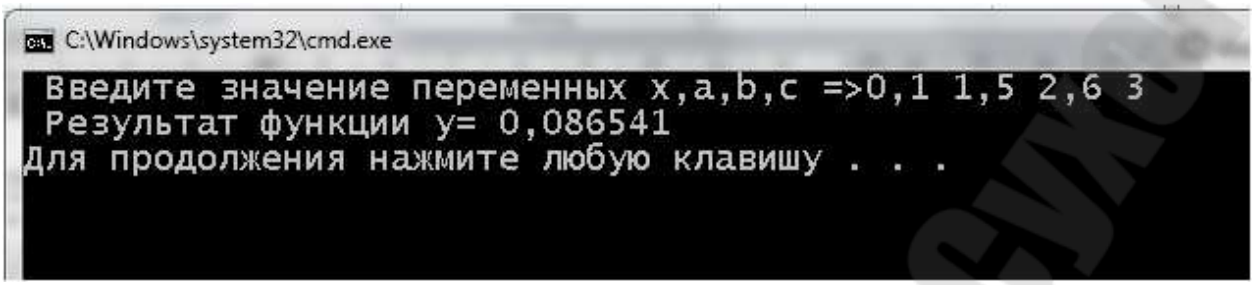
7. Выполнить решение данного уравнения.

$y = \left\| \frac{\cos^2 x}{bx - abc} \right\|$	x =58,6 a = 3,8 b = 0,14 c =4,13

Фрагмент программы

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

int main(void)
{
	setlocale(LC_ALL, "RU");
	float x = 0.0, a = 0.0, b = 0.0, c = 0.0, y = 0.0;
	printf(" Введите значение переменных x,a,b,c =>");


	scanf("%f%f%f%f", &x, &a, &b, &c);

	y = fabs(pow(cos(x),2)/(b*x-a*b*c));
	printf(" Результат функции y= %f \n", y);

	return 0;
}
 return 0;
}
```

Результат работы программы



Задачи условной алгоритмизации

Теоретические сведения по изучению темы:

1. Сущность алгоритма разветвляющейся структуры.
2. Условный оператор if – else.
3. Переключатель switch.

Изложение вопросов:

Под алгоритмом разветвляющейся структуры понимают алгоритм, в котором направление процесса выбирается в зависимости от выполнения или невыполнения того или иного условия.

Условный оператор дает возможность осуществлять разветвление выполнения программы. В качестве условного оператора в С используется конструкция if – else. Обычно используется три формы записи конструкции if – else.

Форма записи 1.

```
If  (условное выражение)        Оператор  выполняется
        оператор;                если выражение истинно.
```

Форма записи 2.

```
If  (условное выражение)        Если выражение истинно,
        оператор 1;              то выполняется оператор 1,
else                            в противном случае оператор 2
        оператор 2;
```

Форма записи 3.

```
If (условное выражение 1)       Если выражение истинно, то
        оператор 1;              выполняется оператор 1.
 else                           Если выражение 1 ложно, но
 If (условное выражение 2)       выражение 2 истинно,
        оператор 2;              выполняется в случае, когда
 else                           оба выражения ложны,
        оператор 3;              выполняется оператор 3.
```

В каждой из этих трех форм оператором может быть либо простой оператор, либо составной. Рассмотрим следующий пример:

```
if (legs =  = 4)
    printf (“Это возможно лошадь. \n”);
  else
    if (legs > 4)
      printf (“Это не лошадь.\n”);
   else
    {
      legs ++;
      printf (“Теперь животное имеет еще одну ногу.\n”);
     }
```

Часто в программе необходимо произвести выбор одного из нескольких вариантов. Это возможно осуществить, используя конструкцию if – else. Но во многих случаях это удобнее сделать с помощью оператора switch (переключателя).

Оператор switch имеет следующий общий вид:

```
switch (выражение)
  {
    case 1 : оператор 1;
             break;
    case 2 : оператор 2;
             break;
          . . .
    case n : оператор n;
             break;
   default : оператор;
             break;
  }
```

Порядок работы:

– определяется значение выражения;

– затем управление передается оператору, у которого в качестве метки используется значение вычислительного выражения;

– осуществляется выход из выбранного case по break и соответственно со всего switch, либо осуществляется “провал” на следующий case если не предусмотрен break;

– если значения не совпало ни с одним из case, то при наличии метки default выполняется оператор, помеченный этой меткой; если default отсутствует, то происходит переход к оператору, расположенному за оператором switch.

Замечание:

1. Выражение должно иметь значение целого типа (включая тип char).
2. Метки должны быть константами или константными выражениями.
3. Присутствие default необязательно, но является хорошим стилем программирования.
4. Наличие break во всех случаях case – хороший стиль программирования.

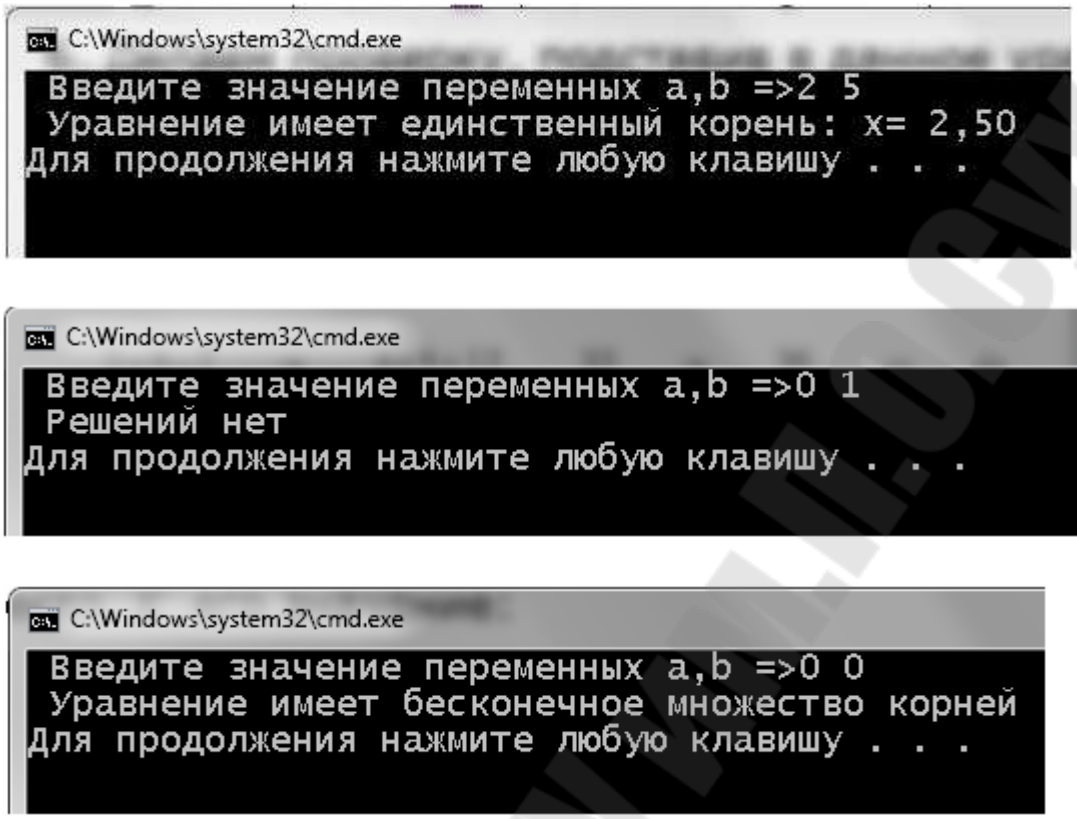
Задачи

1. Решить уравнение x=b/a

Фрагмент программы

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

int main(void)
{
	setlocale(LC_ALL, "RU");
	double x = 0.0, a = 0.0, b = 0.0, c = 0.0, y = 0.0;
	printf(" Введите значение переменных a,b =>");
	scanf("%lf%lf",  &a, &b);
	if (a != 0)
	{
		x = b / a;
	printf(" Уравнение имеет единственный корень: x= %.2lf \n", x);
	}
	else
	{
		if ((a == 0) && (b != 0))
		printf(" Решений нет \n");
		else
	printf(" Уравнение имеет бесконечное множество корней\n");
	}

	return 0;
}
```

Результат работы программы



Задачи циклической алгоритмизации

Алгоритм циклических структур называется такой алгоритм, который повторяет какое-то действие до тех пор, пока верно некоторое условие.

В языке Си существует три оператора цикла- while, for и do-while.

Оператор цикла while имеет следующий синтаксис:

While(выражение)

<оператор;>

Оператор цикла while позволяет выполнять оператор до тех пор, пока значение выражения неравно нулю.

Оператор while является оператором цикла с предусловием, т.е. в начале вычисляется выражение, а затем выполняется оператор. Если выражение ложно, т.е. равно нулю – оператор ни разу не выполняется. Перед каждым следующим выполнением оператора выражение выполняется заново.

Если в теле цикла присутствует оператор break, то при выполнении этого оператора происходит выход из цикла. Если в теле цикла имеется

оператор-контейнер (continue), то при выполнении этого оператора начинается выполняться следующая итерация.

Цикл for имеет следующий синтаксис:

For ([инициирующее выр-е]; [условие]; [выр-е приращение])

<оператор;>

Оператор for выполняется следующим образом:

1. Вначале вычисляется инициирующее выражение. Если инициирующее выражение отсутствие, то никаких действий не выполняется.

2. Вычисляется выражение условие. Если оно истинно, то переходит к шагу 3, если значение выражения равно нулю, то управление передается на следующий за for оператор. Если условие отсутствует, то считается, что условное выражение истинно.

3. Выполняется тело цикла оператора for. Если в теле цикла присутствует оператор разрыва break, то цикл завершает вне зависимости от условного выражения. Если в теле цикла встречается оператор continue, то управление сразу передается на шаг 4, причем если оператор составной, то все операторы от continue, до конца тела цикла не выполняются.

4. Вычисляется выражение приращения, и затем переходит к шагу 2.

Цикл вида for (;;); – является бесконечным циклом.

В инициализирующем выражении и выражении приращения можно указывать несколько выражений, разделяя их запятой.

Цикл do while имеет следующий синтаксис:

do

<оператор>

while (<выражение>);

Тело оператора цикла do while выполняется один или несколько раз до тех пор, пока значение <выражения> не станет ложным (равным нулю). Вначале выполняется тело цикла − <оператор>, затем вычисляется условие – <выражение>. Если выражение ложно, то оператор цикла do while завершается и управление передается следующему за оператором while оператору программы. Если значение выражения истинно (не равно нулю), то тело цикла выполняется снова, и снова вычисляется выражение. Выполнение тела оператора цикла do while повторяется до тех пор, пока выражение не станет ложным. Оператор do while может также завершиться при выполнении в своем теле операторов break, goto, return.

Задачи

1. Вычислить значение функции, при действительном x:

$$Y=\begin{cases}\sin X, \text{если } X > \frac{\pi}{2}, X \neq \pi \\ \cos X, \text{если } \frac{\pi}{3} < X < \frac{\pi}{2} \\ X^2, \text{в остальных случаях}\end{cases}$$

Фрагмент программы

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

#define M_PI 3.14159265358979323846

int main(void)
{
        setlocale(LC_ALL, "RU");
        double x=0.0,xn = 0.0, xk=0, dx=0, y = 0.0;
        int f = 0;
        printf(" Введите начальное значение x =>");
        scanf("%lf",  &xn);
        printf("\n Введите конечное значение x =>");
        scanf("%lf", &xk);
        printf(" \nВведите шаг x =>");
        scanf("%lf", &dx);

        printf("\n     x       y         формула\n", x, y, f);
        for (x = xn; x <= xk;x+=dx)
        {
                if ((x > M_PI / 2) && (x != M_PI))
                {
                        y = sin(x);
                        f = 1;
                }
                else
                {
                        if ((x > M_PI / 3) && (x < M_PI/2))
                        {
                                y = cos(x);
```

```
                        f = 2;
                }
                else
                {
                        y = x*x;
                        f = 3;
                }
        }

        printf("\n%7.2lf   %7.2lf   %7d", x, y, f);
    }
    return 0;
}
```

Результат работы программы

```
 Введите начальное значение x =>0,8

 Введите конечное значение x =>2

Введите шаг x =>0,05

     x         y          формула

   0,80      0,64          3
   0,85      0,72          3
   0,90      0,81          3
   0,95      0,90          3
   1,00      1,00          3
   1,05      0,50          2
   1,10      0,45          2
   1,15      0,41          2
   1,20      0,36          2
   1,25      0,32          2
   1,30      0,27          2
   1,35      0,22          2
   1,40      0,17          2
   1,45      0,12          2
   1,50      0,07          2
   1,55      0,02          2
   1,60      1,00          1
   1,65      1,00          1
   1,70      0,99          1
   1,75      0,98          1
   1,80      0,97          1
   1,85      0,96          1
   1,90      0,95          1
   1,95      0,93          1Для продо
```

2. Вычислить:

$$\sum_{i=1}^{100} i^2 \prod_{j=1}^{20} \frac{1}{i^2 + j^2}.$$

Фрагмент программы

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



int main(void)
{
	setlocale(LC_ALL, "RU");
	double S=0, p=1;
	float i=0,j=0;
	for (i = 1; i <= 100; i++)
	{
		for (j = 1, p = 1; j <= 20; j++)
		{
			p *= 1 / (i*i + j*j);
		}
		S += i*i*p;
   }
	printf("Сумма S = %lf\n", S);
	return 0;
}
```

Результат работы программы

```
Сумма S = 0,000000
Для продолжения нажмите любую клавишу . . .
```

3. Задача. Найти сумму всех элементов последовательности:

$S = \frac{1}{2} - \frac{2}{4} + \frac{3}{8} - \frac{4}{16} + \cdots,$, которые по модулю не меньше, чем 0,001.

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



int main(void)
{
	setlocale(LC_ALL, "RU");
	float S, z, c, d, a=0.5;
	S = 0; z = 1; c = 1; d = 2;	// начальные значения
					//
первыйэлементпоследовательности
	while (a>=0.001)
	{
		S = S + z*a;		// добавить элемент к сумме
		z = - z;		// изменить переменные z, c, d
		c ++;
		d = d * 2;
		a = c / d;		// модуль следующего элемента
	}
	printf("Сумма S = %f\n", S);
	return 0;
}
```

Результат работы программы

```
Сумма S = 0,222778
Для продолжения нажмите любую клавишу . . .
```

Функции

Как и любой объект программы на языке Си, пользовательские функции необходимо декларировать. Объявление функции пользователя, т.е. ее декларация, выполняется в двух формах – в форме описания (объявления) и в форме определения, т.е. любая пользовательская функция должна быть объявлена и определена.

Описанием функции является декларация ее прототипа, который сообщает компилятору о том, что далее будет приведено ее полное определение (текст), т.е. реализация.

Объявление функции (прототип, заголовок) задает ее свойства – идентификатор, тип возвращаемого значения (если такое имеется), количество и типы параметров.

В стандарте языка используется следующий формат декларации (объявления) функций:

тип_результата ID_функции (список);

В списке перечисляются типы параметров данной функции, причем идентификаторы переменных в круглых скобках прототипа указывать необязательно, т.к. компилятор языка их не обрабатывает.

Описание прототипа дает возможность компилятору проверить соответствие типов и количества параметров при фактическом вызове этой функции.

Пример объявления функции fun, которая имеет три параметра типа int, один параметр типа double и возвращает результат типа double:

```
double fun(int, int, int, double);
```

Каждая функция, вызываемая в программе, должна быть определена (только один раз). Определение функции – это ее полный текст, включающий заголовок и код.

Полное определение (реализация) функции имеет следующий вид:

```
тип_результата ID_функции(список параметров)
{
код функции
return выражение;
}
```

Рассмотрим составные части определения пользовательской функции.

Тип результата определяет тип выражения, значение которого возвращается в точку ее вызова при помощи оператора return выражение; (возврат). Выражение преобразуется к типу_результата, указанному в заголовке функции и передается в точку вызова. Тип возвращаемого функцией значения может быть любым базовым типом, а также указателем на массив или функцию. Если функция не должна возвращать значение, указывается тип void. В данном случае оператор return можно не ставить. Из функции, которая не описана как void, необходимо возвращать значение, используя оператор return. Если тип функции не указан, то по умолчанию устанавливается тип int.

Список параметров состоит из перечня типов и идентификаторов параметров, разделенных запятыми. Список параметров определяет объекты, которые требуется передать в функцию при ее вызове.

В определении и в объявлении одной и той же функции типы и порядок следования параметров должны совпадать. Тип возвращаемого значения и типы параметров совместно определяют тип функции.

Функция может не иметь параметров, но круглые скобки необходимы в любом случае. Если у функции отсутствует список параметров, то при декларации такой функции желательно в круглых скобках указать void. Например, void main(void){ ... }.

В функции может быть несколько операторов return, но может и не быть ни одного (тип void – это определяется потребностями алгоритма). В последнем случае возврат в вызывающую программу происходит после выполнения последнего оператора кода функции.

Пример функции, определяющей наименьшее значение из двух целочисленных переменных:

```
int  min (int x, int y)
{
return (x<y) ? x : y;
}
```

Функции, возвращающие значение, желательно использовать в правой части выражений языка Си, иначе возвращаемый результат будет утерян.

В языке Си каждая функция – это отдельный блок программы, вход в который возможен только через вызов данной функции.

Для вызова функции в простейшем случае нужно указать ее имя, за которым в круглых скобках через запятую перечислить список передаваемых ей аргументов. Вызов функции может находиться в любом месте программы, где по синтаксису допустимо выражение того типа, который формирует функция.

Простейший вызов функции имеет следующий формат:

ID_функции (список аргументов);

где в качестве аргументов можно использовать константы, переменные, выражения (их значения перед вызовом функции будут определены компилятором).

Аргументы в списке вызова должны совпадать со списком параметров вызываемой функции по количеству и порядку следования, а типы аргументов при передаче в функцию будут преобразованы, если это возможно, к типу соответствующих им параметров.

Связь между функциями осуществляется через аргументы и возвращаемые функциями значения. Ее можно осуществить также через внешние, глобальные переменные.

Глобальные переменные доступны всем функциям, где они не описаны как локальные переменные. Использовать их для передачи данных между функциями довольно просто, но тем не менее этого делать не рекомендуется. Необходимо стремиться к тому, чтобы функции в программе были максимально независимыми и чтобы их интерфейс полностью определялся прототипами этих функций.

Функции могут располагаться в исходном файле в любом порядке, при этом исходная программа может размещаться в нескольких файлах.

Все величины, описанные внутри функции, являются локальными. Областью их действия является функция. При вызове функции, как и при входе в любой блок, в стеке выделяется память под локальные автоматические переменные. Кроме того, в стеке сохраняется содержимое регистров процессора на момент, предшествующий вызову функции, и адрес возврата из функции, для того чтобы при выходе из нее можно было продолжить выполнение вызывающей функции. При выходе из функции соответствующий участок стека освобождается, поэтому значения локальных переменных между вызовами одной и той же функции не сохраняются. Если этого требуется избежать, при объявлении локальных переменных используется модификатор static.

Указатель – это группа ячеек, в которых может храниться адрес.

В Си имеется унарная операция & – операция получения адреса. Запись вида pch=&ch; присваивает адрес ячейки, где находится ch переменной pch, которая является переменной указателя. В этом случае принято говорить: pch указывает на ch, или, что одно и то же, pch ссылается на ch.

Указатель на переменную содержит адрес памяти расположения этой переменной.

Объявление указателя имеет следующее формальное описание:
тип_переменной *имя_переменной_адреса;

Инициализация указателя выполняется следующим образом:
тип_переменной имя_переменной_содержания;
имя_переменной_адреса = &имя_переменной_содержания;

Объявление указателя может быть выполнено с одновременной инициализацией:
тип_переменной *имя_переменной_адреса = &имя_переменной_содержания;

Доступ к значению переменной по указателю имеет следующее формальное описание:

имя_переменной_содержания1=*имя_переменной_адреса;

При работе с указателями действуют следующие правила:

- при объявлении переменной-указателя перед именем переменной указывается операция *;
- если одним оператором объявляется несколько переменных-указателей, то перед каждой такой переменной следует указывать операцию *;
- после объявления указателя его следует инициализировать адресом значения того же типа, что и тип указателя;
- для получения адреса переменной перед ее именем указывается операция взятия адреса &;

– для получения значения переменной по указателю на нее перед указателем ставится операция разыменования * (называемая иногда операцией взятия значения);

– указатель строки содержит адрес первого символа строки;

– при увеличении указателя на единицу значение, содержащееся в переменной-указателе, увеличивается на число байт, которое отведено под переменную данного типа.

Правило: операция получения адреса & применяется только к объектам, расположенным в памяти: к переменным и элементам массивов. Её операндом не может быть ни выражение, ни константа, ни регистровая переменная.

Унарная операция * есть операция раскрытия ссылки. (другие названия: операция разадресации, операция косвенной разадресации, наиболее часто используется термин операция разадресации). Эта операция применяется к указателю и выдает обьект, на который данный указатель ссылается.

Задачи

1. Необходимо вычислить y1 и y2 в двух точках – x1 и x2. Вычисление y1 и y2 оформить в виде функции. Обратиться к функции при разных значениях аргумента x, равных x1 и x2, вывести результаты – y1 и y2 и значение x, при котором они получены. $y1 = \cos\frac{\sqrt{tgx}}{e^x} - \frac{1}{2.1^x}$ $y2 = \sqrt[3]{x} + tg^2 x^3$ x1=0.5, x2=1.1.

Фрагмент программы

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

double fun_y1(double x);
double fun_y2(double x);

int main(void)
{
      setlocale(LC_ALL, "RU");
      double x1=0.5,x2=1.1;

printf("\nРезультат функции y1=%.3lf при x=%.2lf\n", fun_y1(x1), x1);
printf("\nРезультат функции y1=%.3lf при x=%.2lf\n", fun_y1(x2), x2);
printf("\nРезультат функции y2=%.3lf при x=%.2lf\n", fun_y2(x1), x1);
printf("\nРезультат функции y2=%.3lf при x=%.2lf\n", fun_y2(x2), x2);
```

```
        return 0;
}
 // определение функций
double fun_y1(double x){
        return(cos(sqrt(tan(x))/exp(x))-1/pow(2.1,x));
}
double fun_y2(double x){
        return(pow(x, 0.33)+pow(tan(pow(x,3.0)),3.0));
}
```

Результат работы программы

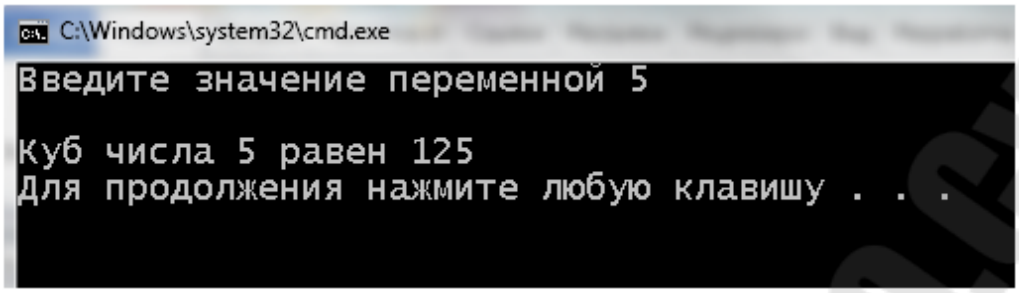
```
Результат функции y1=0,211 при x=0,50
Результат функции y1=0,451 при x=1,10
Результат функции y2=0,798 при x=0,50
Результат функции y2=69,448 при x=1,10
```

1. Написать функцию, которая возвращает куб числа.

Фрагмент программы

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

int cube(int x);  //объявление функции

int main(void)
{
        setlocale(LC_ALL, "RU");
        int a=0;

        printf("Введите значение переменной ");
        scanf("%d", &a);
        printf("\nКуб числа %d равен %d\n",a,cube(a));  //вызов функции
        return 0;
}

int cube(int x)  // определение функции
{
        return(x*x*x);
}
```

Результат работы программы

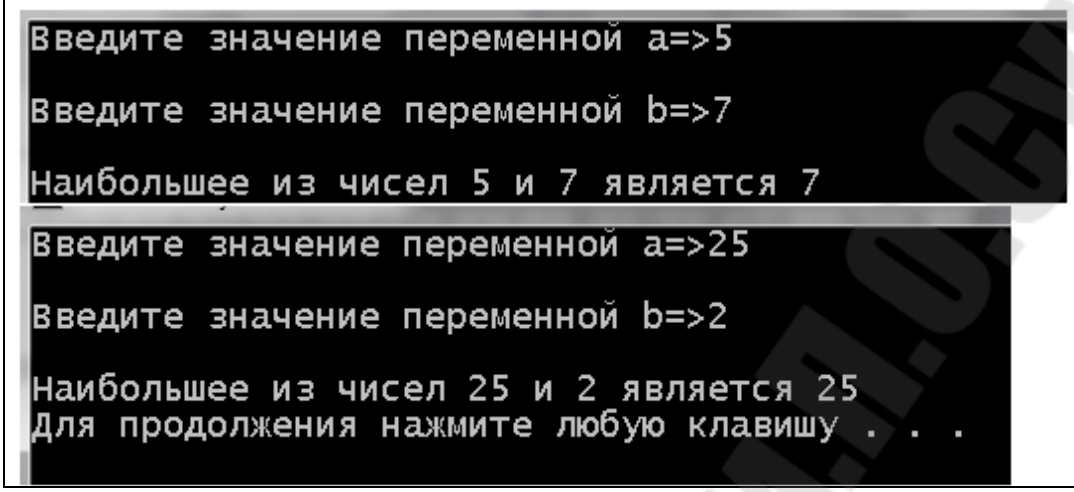


2. Написать функцию для нахождения наибольшего из двух чисел.

Фрагмент программы

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

int max(int ,int );          //объявление функции

int main(void)
{
        setlocale(LC_ALL, "RU");
        int a=0,b=0;

        printf("Введите значение переменной a=>");
        scanf("%d", &a);
        printf("\nВведите значение переменной b=>");
        scanf("%d", &b);
        printf("\nНаибольшее из чисел %d и %d является
%d\n",a,b,max(a,b)); //вызов функции
        return 0;
}

int max(int a,int b)   // определение функции
{
        return(a > b ? a:b);
}
```

Результат работы программы



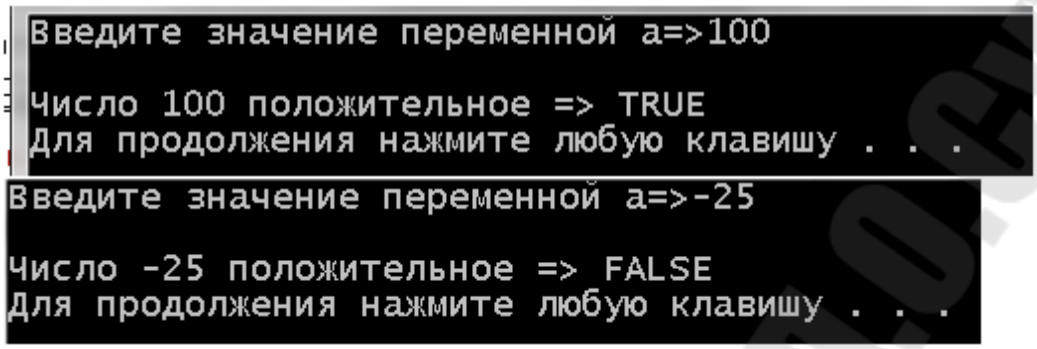
4. Написать функцию, которая возвращает истину, если передаваемое значение положительное и ложь, если отрицательное.

Фрагмент программы

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

char* positive(int);                    //объявление функции

int main(void)
{
        setlocale(LC_ALL, "RU");
        int a=0;

        printf("Введите значение переменной a=>");
        scanf("%d", &a);
        printf("\nЧисло %d положительное => %s\n",a,positive(a));
//вызов функции
        return 0;
}

char* positive(int a)  // определение функции
{
        return(a > 0 ? "TRUE":"FALSE");
```

```
}
```

Результат работы программы



5. Написать функцию, которая в зависимости от выбора пользователя вызывает функции сложения, произведения, вычитания, деления двух чисел (эти функции тоже нужно написать самостоятельно).

Фрагмент программы

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

void menu(float,float); //объявление функции
float add(float x, float y);
float mult(float x, float y);
float sub(float x, float y);
float div(float x, float y);

int main(void)
{
	setlocale(LC_ALL, "RU");

	float a, b;
	printf("\nВведите первое число  ");
	scanf("%f",&a);
	printf("\nВведите второе число  ");
	scanf("%f",&b);
	menu(a,b);
	return 0;
}
```

```
void menu(float x,float y)  // определение функции
{
        char f;
        int a=1;
        float z=0;
        do{
                printf("\nУкажите вид функции:\n\t + - сложение\n");
                printf("\t * - произведение\n");
                printf("\t - - вычитание\n");
                printf("\t / - деление\n");
                fflush(stdin);
                scanf("%c", &f);
                switch(f){
                case '+':z=add(x,y); break;
                case '*':z=mult(x,y); break;
                case '-':z=sub(x,y); break;
                case '/':z=div(x,y); break;
                default:printf("\nТакая операция не предусмотрена!\n");
                }
                printf("Результат %5.2f\n", z);
                printf("Продолжить ? (1-Да, 0-Нет)  ");
                scanf("%d", &a);
        } while (a);
}

float add(float x, float y)
{
        return(x + y);
}
float mult(float x, float y)
{
        return(x * y);
}
float sub(float x, float y)
{
        return(x - y);
}
float div(float x, float y)
{
        return(x/ y);
}
```

Результат работы программы

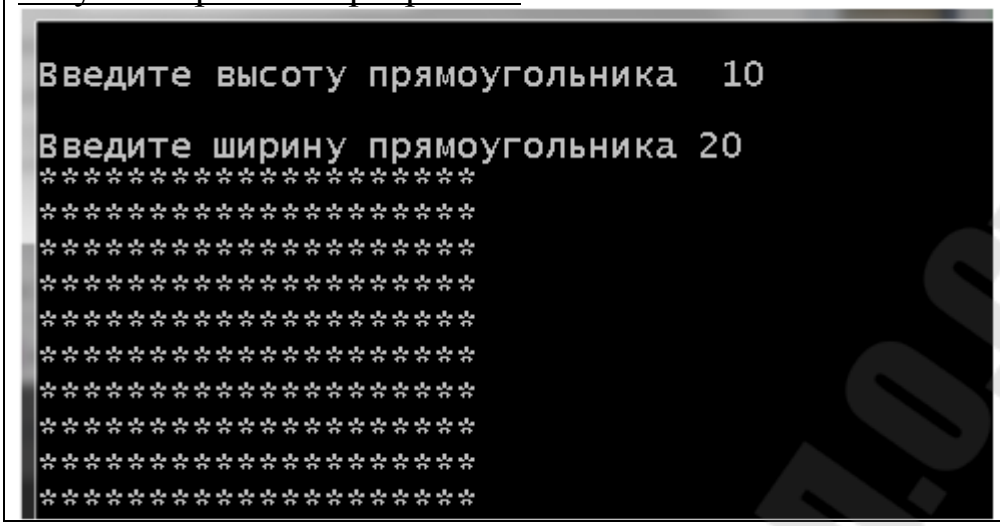
```
Введите первое число  5

Введите второе число  2

Укажите вид функции:
         + - сложение
         * - произведение
         - - вычитание
         / - деление
+
Результат  7,00
Продолжить ? (1-Да, 0-Нет)  1

Укажите вид функции:
         + - сложение
         * - произведение
         - - вычитание
         / - деление
*
Результат 10,00
Продолжить ? (1-Да, 0-Нет)  1

Укажите вид функции:
         + - сложение
         * - произведение
         - - вычитание
         / - деление
-
Результат  3,00
Продолжить ? (1-Да, 0-Нет)  1

Укажите вид функции:
         + - сложение
         * - произведение
         - - вычитание
         / - деление
/
Результат  2,50
Продолжить ? (1-Да, 0-Нет)
```

6. Написать функцию, выводящую на экран прямоугольник с высотой N и шириной K.

Фрагмент программы

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

void rectangle(int h, int w);   //объявление функции


int main(void)
{
	setlocale(LC_ALL, "RU");

	int n, k;
	printf("\nВведите высоту прямоугольника  ");
	scanf("%d",&n);
	printf("\nВведите ширину прямоугольника ");
	scanf("%d",&k);
	rectangle(n,k);
	return 0;
}

void rectangle(int h, int w)  // определение функции
{
	for (int i = 0; i < h;i++)
	{
		for (int j = 0; j < w; j++)
			printf("*");
		printf("\n");
	}

}
```

Результат работы программы



7. Написать функцию, которая проверяет, является ли переданное ей число простым? Число называется простым, если оно делится без остатка только на себя и на единицу.

Фрагмент программы

```
#include <stdio.h>
#include <locale.h>
int simple(int);    //объявление функции
int main(void)
{
        setlocale(LC_ALL, "RU");
        int n;
        printf("\nВведите число  ");
        scanf("%d",&n);
        if (simple(n))
                printf("\nЧисло %d - простое\n",n);
        else
                printf("\nЧисло %d - не является простым\n",n);
        return 0;
}
int simple(int n)  // определение функции
{
        for (int i = 2; i < n;i++)
        {
                if (n%i == 0) return(0);
        }
        return(1);
```

```
}
```

Результат работы программы

```
Введите число  13

Число 13 - простое

Введите число  21

Число 21 - не является простым
```

8. Написать функцию, вычисляющую факториал переданного ей числа.

Фрагмент программы

```
#include <stdio.h>
#include <locale.h>
int factorial(int);  //объявление функции


int main(void)
{
	setlocale(LC_ALL, "RU");
	int n;
	printf("\nВведите число  ");
	scanf("%d",&n);
	printf("\nФакториал числа %d равен %d\n", n, factorial(n));
	return 0;
}

int factorial(int n)  // определение функции
{
	int f = 1;
	for (int i = 2; i <=n;i++)
	{
		f*=i;
	}
	return(f);
}
```

Результат работы программы

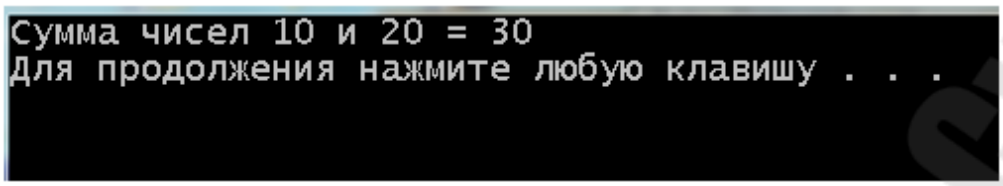
```
Введите число  3

Факториал числа 3 равен 6
```

9. Написать программу, вычисляющую значения сумму двух чисел. Расчет должен производиться в функции, которая получает данные по указателю и возвращает результат по указателю.

Фрагмент программы

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

int* sumXY(int* x, int* y);

int main(void)
{
	setlocale(LC_ALL, "RU");
	char* locale = setlocale(LC_ALL, "");

	int x = 10;
	int z = 20;
	int sum = *sumXY(&x, &z);
	printf("Сумма чисел %d и %d = %d \n",x,z, sum);

	return 0;
}
 // определение функций
int* sumXY(int* x, int *y)
{
	static int sum;
	int *ptr;
	sum = *x + *y;
	ptr = &sum;
	return ptr;
}
```

Результат работы программы



10. Описать 2 указателя на символьный тип. Выделить динамическую память по одному символу для каждого указателя. Ввести значения в выделенную память с клавиатуры. Вывести прямоугольник из символов первой переменной.

Фрагмент программы

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

int main(void)
{
	setlocale(LC_ALL, "RU");
	char* locale = setlocale(LC_ALL, "");
	char *ptr1, *ptr2;
	ptr1 = (char *) calloc(1, sizeof(char));
	ptr2 = (char *) calloc(1, sizeof(char));
	printf("Введите символ 1 \n");
	scanf("%c", ptr1);
	fflush(stdin);
	printf("Введите символ 2 \n");
	scanf("%c", ptr2);
	for (int i = 0; i < 10; i++){
		for (int i = 0; i < 10; i++)
			printf("%c ",*ptr1);
		printf("\n");
	}

	return 0;
```

```
}
```

Результат работы программы

Структурированные типы данных

В языке Си для этой цели используется сложный тип данных – массив, представляющий собой упорядоченную конечную совокупность элементов одного типа. Число элементов массива называют его размером. Каждый элемент массива определяется идентификатором массива и своим порядковым номером – индексом. Индекс – целое число, по которому производится доступ к элементу массива. Индексов может быть несколько. В этом случае массив называют многомерным, а количество индексов одного элемента массива является его размерностью.

Описание массива в программе отличается от описания простой переменной наличием после имени квадратных скобок, в которых задается количество элементов массива. Например, double a [10]; – описание массива из 10 вещественных чисел.

При описании массивов квадратные скобки являются элементом синтаксиса, а не указанием на необязательность конструкции.

Размеры массивов предпочтительнее вводить с клавиатуры как значения целочисленных переменных или задавать с помощью именованных констант, поскольку при таком подходе для ее изменения достаточно скорректировать значение константы всего лишь в одном месте программы.

В программе одномерный массив объявляется следующим образом:

тип ID_массива [размер] = {список начальных значений};

тип – базовый тип элементов массива (целый, вещественный, символьный); размер – количество элементов в массиве.

Список начальных значений используется при необходимости инициализировать данные при объявлении, он может отсутствовать.

При декларации массива можно использовать также атрибуты «класс памяти» и const.

Размер массива вместе с типом его элементов определяет объем памяти, необходимый для размещения массива, которое выполняется на этапе компиляции, поэтому размер массива задается только константой или константным выражением. Нельзя задавать массив переменного размера, для этого существует отдельный механизм – динамическое выделение памяти.

Пример объявления массива целого типа: int a[5];

Индексы массивов в языке Си начинаются с 0, т.е. в массиве а первый элемент: a[0], второй – a[1], … пятый – a[4].

Обращение к элементу массива в программе на языке Си осуществляется в традиционном для многих других языков стиле – записи операции обращения по индексу [] (квадратные скобки), например:

```
a[0]=1;
a[i]++;
a[3]=a[i]+a[i+1];
```

Пример объявления массива целого типа с инициализацией начальных значений:

```
int a[5]={2, 4, 6, 8, 10};
```

Если в группе {…} список значений короче, то оставшимся элементам присваивается 0.

Внимание. В языке Си с целью повышения быстродействия программы отсутствует механизм контроля выхода за границы индексов массивов. При необходимости такой механизм должен быть запрограммирован явно.

Структура – это одна или несколько переменных (возможно, различных типов), которые для удобства работы с ними сгруппированы под одним именем. Структуры помогают в организации сложных данных (особенно в больших программах), поскольку позволяют группу связанных между собой переменных трактовать не как множество отдельных элементов, а как единое целое.

Объявление структуры начинается с ключевого слова struct и содержит список объявлений, заключенный в фигурные скобки. За словом struct может следовать имя, называемое тегом структуры.

Структурный шаблон сам по себе является схемой без содержания. Он сообщает компилятору как делать что-то, но действий никаких не

вызывает в программе. Для того чтобы структура заработала необходимо создать структурную переменную.

Объявление структуры, не содержащей списка переменных, не резервирует памяти; оно просто описывает шаблон, или образец структуры. Объявление структуры определяет тип. За правой фигурной скобкой, закрывающей список элементов, могут следовать переменные точно так же, как они могут быть указаны после названия любого базового типа.

Объединение – это переменная, которая может содержать (в разные моменты времени) объекты различных типов и размеров. Все требования относительно размеров и выравнивания выполняет компилятор. Объединения позволяют хранить разнородные данные в одной и той же области памяти без включения в программу машинно-зависимой информации.

Цель введения в программу объединения – иметь переменную, которая бы на законных основаниях хранила в себе значения нескольких типов.

Задачи

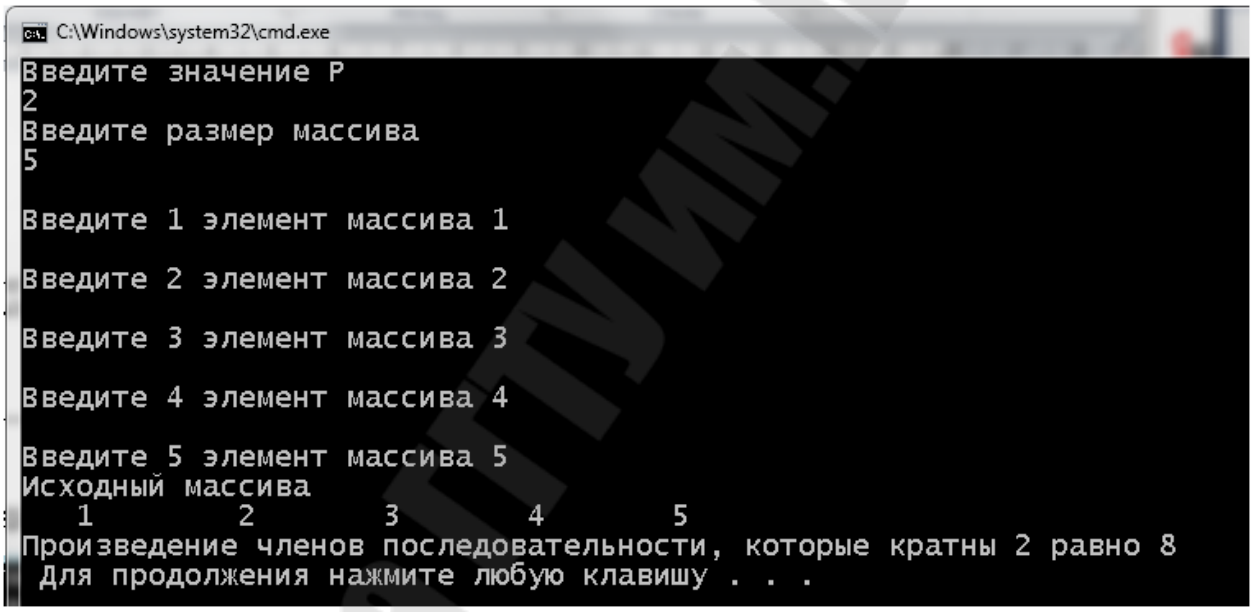
1.1. Даны натуральные числа N, P, целые числа A1……An. Получить произведение членов последовательности, которые кратны P. Первый вариант решения – обычная адресация элементов массива.

Фрагмент программы

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

void main(void)
{
	setlocale(LC_ALL, "RU");
	char* locale = setlocale(LC_ALL, "");
	int i;
	int n,p,mult=1;
	int mas[25];
	printf("Введите значение P\n");
	scanf_s("%d", &p);
	printf("Введите размер массива\n");
	scanf_s("%d", &n);
	for (i = 0; i < n; i++)
	{
		printf("\nВведите %d элемент массива ", i + 1);
		scanf_s("%d", &mas[i]);
```

```
            if (mas[i] % p == 0) mult *= mas[i];
        }

        printf("Исходный массива\n");
        for (i = 0; i < n; i++)
            printf(" %3d \t ",mas[i]);

printf("\nПроизведение членов последовательности, которые кратны %d равно %d\n ", p,mult);

}
```

Результат работы программы



Для инициализации элементов массива очень часто используют генерацию случайных чисел. Выше приведенный пример программы доработаем с учетом инициализации элементов массива случайными значениями в диапазоне от 10 до 100.

Функция rand() возвращает случайное целое число в диапазоне от нуля до RAND_MAX. RAND_MAX это специальная константа языка Си, в которой содержится максимальное целое число, которое может быть возвращено функцией rand().

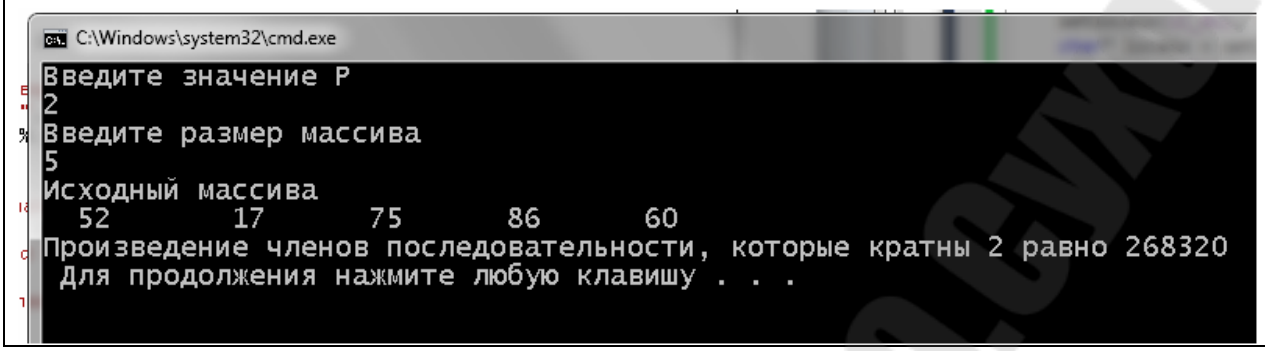
Функция rand() определена в заголовочном файле stdlib.h.

Но, если используется только функция rand(), генерируются всё время одинаковые числа. На это можно повлиять используя функцию srand(), которая также определена в заголовочном файле stdlib.h.

Фрагмент программы

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

void main(void)
{
	setlocale(LC_ALL, "RU");
	char* locale = setlocale(LC_ALL, "");
	srand(time(NULL));

	int i;
	int n,p,mult=1;
	int mas[25];

	printf("Введите значение P\n");
	scanf_s("%d", &p);

	printf("Введите размер массива\n");
	scanf_s("%d", &n);
	for (i = 0; i < n; i++)
	{
		/* генерируем пять случайных целых чисел из отрезка [10;100] */
		mas[i] = rand() % (100 - 10 + 1);
		if (mas[i] % p == 0) mult *= mas[i];
	}

	printf("Исходный массива\n");
	for (i = 0; i < n; i++)
		printf(" %3d \t ",mas[i]);

	printf("\nПроизведение членов последовательности, которые кратны %d равно %d\n ", p,mult);

}
```

Результат работы программы



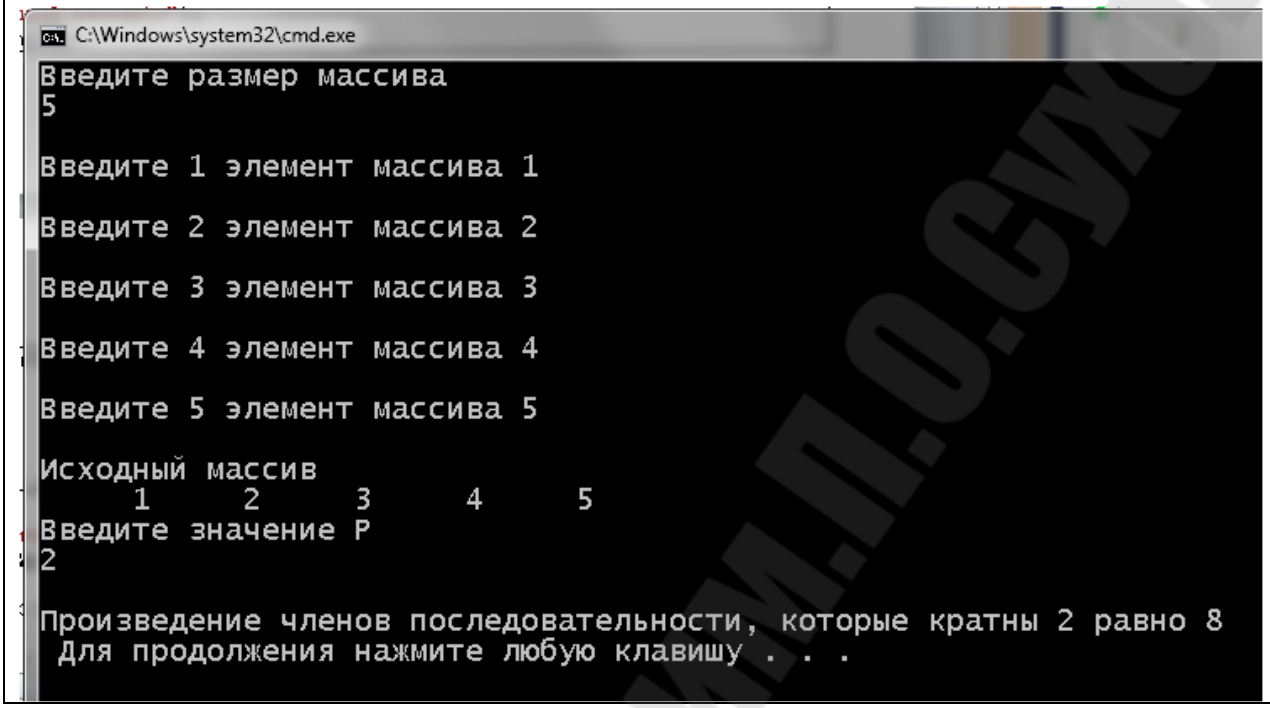
1.2. Даны натуральные числа N, P, целые числа A1……An. Получить произведение членов последовательности, которые кратны P. Второй вариант – адресация через указатели и использование функций.

Фрагмент программы

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

int n;
int *InputArray()
{
	int i;

	printf("Введите размер массива\n");
	scanf_s("%d", &n);
	int *mas = (int*)malloc(n * sizeof(int));
	for (i = 0; i < n; i++)
	{
		printf("\nВведите %d элемент массива ", i + 1);
		scanf_s("%d", (mas + i));
	}

	return (mas);

}

void OutputArray(int *mas)
{
	int i;
```

```
        printf("\nИсходный массив\n");
        for (i = 0; i<n; i++)
                printf(" %5d", *(mas + i));
}
void Multi(int *mas)
{
        int i, mult = 1,p;

        printf("\nВведите значение P\n");
        scanf_s("%d", &p);

        for (i = 1; i < n; i++)
        {
                if (*(mas + i) % p == 0)
                        mult *= *(mas + i);
        }

printf("\nПроизведение членов последовательности, которые кратны %d
равно %d\n ", p, mult);
}
void main(void)
{
        setlocale(LC_ALL, "RU");
        char* locale = setlocale(LC_ALL, "");
        int *mas;

        setlocale(LC_CTYPE, "");
        mas = InputArray();//ввод массива
        OutputArray(mas);//вывод массива
        Multi(mas);//произведение элементов, кратных P
        free(mas);

}
```

Результат работы программы



Задание 2

В упорядоченный по неубыванию числовом массиве все числа, неменьшие заданного, поставить в начале массива, не меняя их взаимного расположения переставляемых чисел.

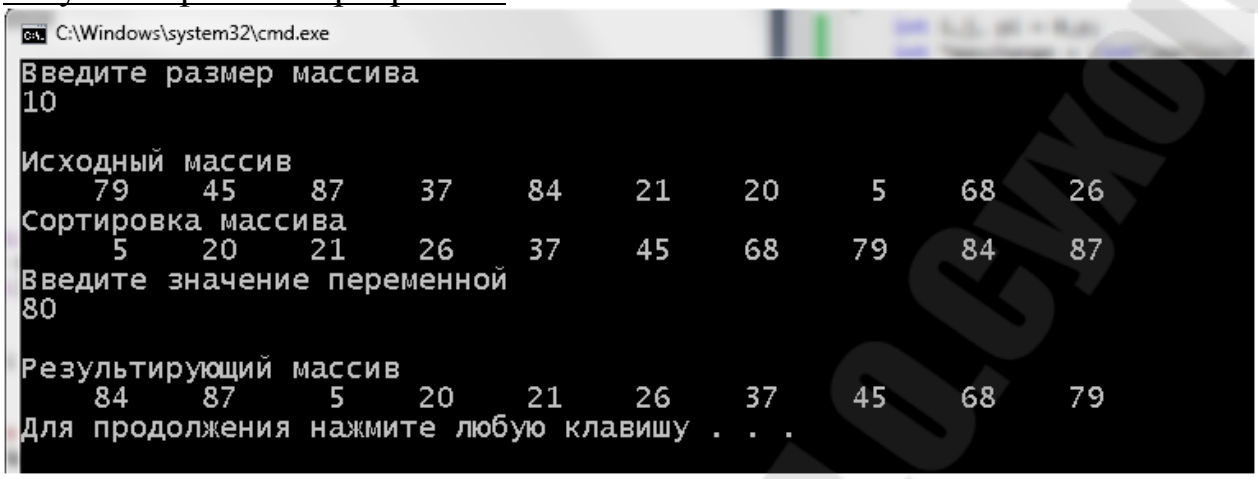
Фрагмент программы

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


int n;
int *InputArray()
{
	int i;

	printf("Введите размер массива\n");
	scanf_s("%d", &n);
	int *mas = (int*)malloc(n * sizeof(int));
	for (i = 0; i < n; i++)
```

```
            *(mas+i) = rand() % (100 - 10 + 1);

        return (mas);
}

void OutputArray(int *mas)
{
        int i;

        for (i = 0; i<n; i++)
                printf(" %5d", *(mas + i));
}

int* Change(int *mas)
{
        int i,j, pi = 0,p;
        int *maschange = (int*)malloc(n * sizeof(int));

        printf("\nВведите значение переменной\n");
        scanf_s("%d", &p);

        for (i = 0; i < n; i++)
                if (*(mas + i) > p)
                {       pi = i;
                        break;
                }

        for (i = 0; i <= (n-pi); i++)
                (*(maschange + i)) = *(mas + pi + i);
        for (i = (n-pi),j=0; i <n; i++,j++)
                (*(maschange + i)) = *(mas + j);

        return(maschange);
}


void Sort(int *mas)
{
        for (int i = 0; i < n - 1; i++)
        {
                //Поиск минимального элемента среди mas[i],...,mas[n-1]
                int minIndex = i;
```

```
            for (int j = i + 1; j < n; j++)
            {
                if (mas[j] < mas[minIndex])
                    minIndex = j;
            }
            //Обмен местами элементов mas[i] и mas[minIndex]
            int tmpValue = mas[i];
            mas[i] = mas[minIndex];
            mas[minIndex] = tmpValue;
        }

}

void main(void)
{
        setlocale(LC_ALL, "RU");
        char* locale = setlocale(LC_ALL, "");
        int *mas;

        srand(unsigned int(time(NULL)));

        setlocale(LC_CTYPE, "");
        mas = InputArray();//ввод массива
        printf("\nИсходный массив\n");
        OutputArray(mas);//вывод массива
        Sort(mas);//сортировка массива
        printf("\nСортировка массива\n");
        OutputArray(mas);//вывод массива
        mas=Change(mas);//обработка массива
        printf("\nРезультирующий массив\n");
        OutputArray(mas);//вывод массива
        printf("\n");
        free(mas);
}
```

Результат работы программы



Задание 3

В матрице M x N переставить строки таким образом, чтобы получилась последовательность s1≤s2≤…≤sm, где si – сумма абсолютных значений всех элементов i – строки.

Программа должна выполнять ввод и вывод матрицы и дополнительных данных, выполнять необходимые действия и выводить результаты. В решении предусмотрена адресация через указатели с использованием функций.

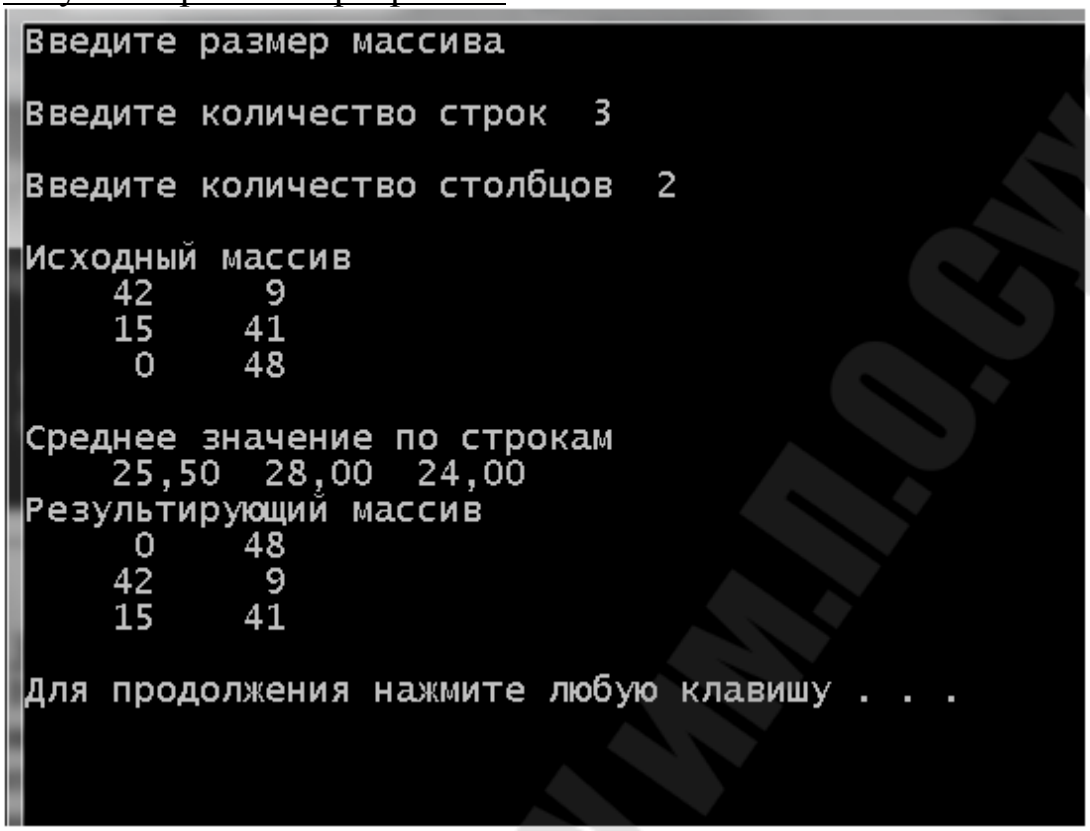
Фрагмент программы

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

int n,m;
int **InputArray()
{
	int i, j;
	printf("Введите размер массива\n");
	printf("\nВведите количество строк  ");
	scanf_s("%d", &n);
	printf("\nВведите количество столбцов  ");
	scanf_s("%d", &m);
	int **mas = (int**)malloc(n * sizeof(int));
	for (i = 0; i < n; i++)
```

```
            mas[i] = (int*)malloc(m * sizeof(int));
        for (i = 0; i < n; i++)
            for (j = 0; j < m; j++)
                mas[i][j] = rand() % (100 - 10 + 1);

        return (mas);
}
void OutputArray(int **mas)
{
        int i,j;

        for (i = 0; i<n; i++)
        {
            for (j = 0; j < m; j++)
                printf(" %5d", mas[i][j]);
            printf("\n");
        }
}
int** Change(int **mas)
{
        int i,j, pi = 0,p;
        float *s = (float*)malloc(n * sizeof(float));
        int *t = (int*)malloc(n * sizeof(int));

        for (i = 0; i<n; i++)
        {
            s[i] = 0;
            for (j = 0; j < m; j++)
                s[i] += (float)mas[i][j];
            s[i] /= (float)m;

        }

        printf("\nСреднее значение по строкам\n  ");
        for (i = 0; i < n; i++)
            printf("%7.2f", s[i]);

        for (i = 0; i < n - 1; i++)
        {
            //Поиск минимального элемента среди mas[i],...,mas[n-1]
            int minIndex = i;
            for (j = i + 1; j < n; j++)
```

```
            {
                if (s[j] < s[minIndex])
                    minIndex = j;
            }
            //Обмен местами элементов mas[i] и mas[minIndex]
            t = mas[i];
            mas[i] = mas[minIndex];
            mas[minIndex] = t;

            float tmpValue = s[i];
            s[i] = s[minIndex];
            s[minIndex] = tmpValue;
        }
        return(mas);
}

void main(void)
{
        setlocale(LC_ALL, "RU");
        char* locale = setlocale(LC_ALL, "");
        int **mas;
        srand(unsigned int(time(NULL)));
        setlocale(LC_CTYPE, "");
        mas = InputArray();//ввод массива
        printf("\nИсходный массив\n");
        OutputArray(mas);//вывод массива
        mas=Change(mas);//обработка массива
        printf("\nРезультирующий массив\n");
        OutputArray(mas);//вывод массива
        printf("\n");

}
```

Результат работы программы



Задание 4

В задаче задается строка текста, состоящая из нескольких слов. Слова отделяются последовательностью пробелов. Разработать программу, которая должна вводить строку, печатать ее, производить преобразования или вычисления, указанные в задании и выводить результат. Данные для отладки выбирать самостоятельно.

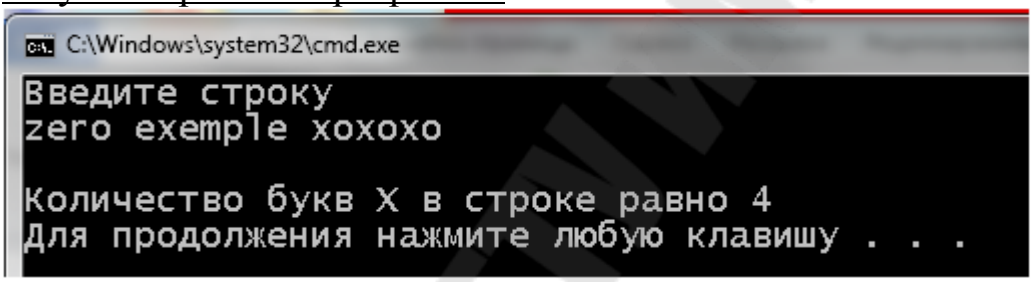
Подсчитать, сколько раз среди символов строки встречается буква x.

Фрагмент программы

```
#include <stdio.h>
#include <locale.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

void main(void){
        char str[25];
```

```
    int i, n, countX=0;
    setlocale(LC_ALL, "RU");
    srand(unsigned int(time(NULL)));
    printf("Введите строку\n");
    gets(str);
    n = strlen(str);
    for (i = 0; i < n; i++)
    {
        if (str[i] == 'x')
            countX++;
    }
    printf("\nКоличество букв X в строке равно %d\n", countX);


}
```

Результат работы программы



Задание 5

В каждом варианте задания необходимо обеспечить выполнение следующих действий (операций):

1) ввод информации в массив структур;

2) просмотр на экране содержимого массива структур в виде таблицы;

3) одним из элементов структуры является объединение;

4) вывод содержимого на печать.

Отладку программ производить с самостоятельно выбираемым набором данных. Число записей - произвольное.

Меню	• Название блюда • Стоимость блюда • Объединение	• Калорийность блюда (для тех, кто следит за фигурой) • Отсутствие вредных веществ (для тех, кто следит за здоровьем)

Фрагмент программы

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

int* sumXY(int* x, int* y);

int main(void)
{
	setlocale(LC_ALL, "RU");
	char* locale = setlocale(LC_ALL, "");

	int x = 10;
	int z = 20;
	int sum = *sumXY(&x, &z);
	printf("Сумма чисел %d и %d = %d \n",x,z, sum);

	return 0;
}

 // определение функций
int* sumXY(int* x, int *y)
{
	static int sum;
	int *ptr;
	sum = *x + *y;
	ptr = &sum;
	return ptr;
}
```

Результат работы программы

```
Сумма чисел 10 и 20 = 30
Для продолжения нажмите любую клавишу . . .
```

Литература

1. Информатика. Базовый курс / 2-е изд. под ред. С. В. Симоновича. - СПб.: Питер, 2007. – 640 с.
2. Касаткин, А. И. Профессиональное программирование на языке Си: управление ресурсами / А. И. Касаткин. - Минск : Вышэйшая школа, 1992. - 432 с.
3. Котлинская Г. П. Программирование на языке СИ : справ. пособие. - Минск : Выш. шк., 1991. – 155 с.
4. Макогон В. С. Язык программирования Си для начинающих : учеб.пособие. – Одесса : астропринт, 1993. – 96 с.
5. Структуры данных в языке СИ [Электронный ресурс] : пособие по курсам "Модели и структуры данных" и "Основы алгоритмизации и программирования" для студентов специальностей 1-40 01 02 "Информационные системы и технологии (по направлениям)" и 1-36 04 02 "Промышленная электроника" дневной и заочной форм обучения / О. А. Кравченко. – Гомель : ГГТУ им. П. О. Сухого, 2010. – 149 с.
6. Основы алгоритмизации и программирования : курс лекций по одноименной дисциплине для студентов специальности 1-40 01 02 "Информационные системы и технологии (по направлениям)" дневной формы обучения / О. А. Кравченко, С. М. Мовшович, Е. В. Коробейникова. - Гомель : ГГТУ им. П. О. Сухого, 2010. - 111 с.
7. Программирование на языке С. Массивы : пособие по выполнению контрольных и лабораторных работ по дисциплине "Вычислительная техника и программирование" для студентов технических специальностей дневной и заочной форм обучения / О. А. Кравченко, Д. А. Литвинов ; кафедра "Информационные технологии". - Гомель : ГГТУ им. П. О. Сухого, 2007. – 38 с.
8. Костюкова, Н. И. Язык Си и особенности работы с ним : учебное пособие : [16+] / Н.И. Костюкова, Н.А. Калинина ; Национальный Открытый Университет "ИНТУИТ". – Москва : Интернет-Университет Информационных Технологий (ИНТУИТ) : Бином. Лаборатория знаний, 2006. – 207 с. – (Основы информационных технологий). – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=233309 (дата обращения: 10.02.2021). – ISBN 5-9556-0026-4. – Текст: электронный.
9. Фридман, А.Л. Язык программирования Си++ : [16+] / А. Л. Фридман. – 2-е изд., исправ. – Москва : Национальный Открытый Университет «ИНТУИТ», 2016. – 219 с. – (Основы информационных технологий). – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=578114 (дата обращения: 10.02.2021). – Текст : электронный.

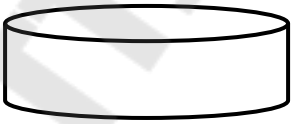
10. Царев, Р.Ю. Программирование на языке Си : учебное пособие / Р.Ю. Царев ; Сибирский федеральный университет. – Красноярск : Сибирский федеральный университет (СФУ), 2014. – 108 с. : табл., схем. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=-book&id=364601. – Дата обращения: 10.02.2021.– Текст : электронный.

ПРИЛОЖЕНИЕ А

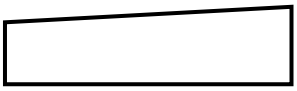
(справочное)

Обозначение и назначение основных символов схем

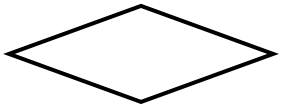
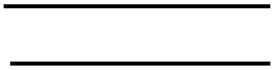
Таблица А.1.

Название символа	**Обозначение**	**Значение**
1	**2**	**3**
Символы данных		
Данные		Ввод или вывод данных; носитель данных не определен
Запоминаемые данные		Хранимые данные в виде, пригодном для обработки, носитель данных не определен
Оперативное запоминающее устройство		Данные, хранящиеся в оперативном запоминающем устройстве
Запоминающее устройство с прямым доступом		Данные, хранящиеся в запоминающем устройстве с прямым доступом (магнитный диск)
Документ		Данные, представленные на носителе в удобочитаемой форме

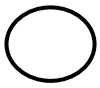
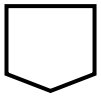
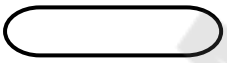
Продолжение таблицы А.1

1	2	3
Ручной ввод		Данные, вводимые вручную во время обработки
Дисплей		Данные, представленные в человекочитаемой форме на носителе в виде отображающего устройства
Символы процесса		
Процесс		Обработка данных любого вида, приводящее к изменению значения, формы или размещения информации
Предопределенный процесс		Использование подпрограммы (или модуля)
Подготовка		Модификация команды или группы команд с целью воздействия на некоторую последующую функцию

Продолжение таблицы А.1

1	2	3
Решение		Проверка условия и выбор одного из нескольких альтернативных выходов
Параллельные действия		Синхронизация двух или более параллельных операций
Границы цикла		Отображают начало и конец циклического процесса
Символы линий		
Линия		Отображает поток данных и управления. При необходимости могут быть добавлены стрелки - указатели
Канал связи		Передачу данных по каналу связи

Окончание таблицы А.1

1	**2**	3
Соединитель (на одной странице)		Обрыв линии и продолжение ее в другом месте. Соответствующие символы – соединители должны иметь одно и то же уникальное обозначение.
Межстраничный соединитель		
Терминатор		Начало или конец схемы программы
Комментарий		Пояснения к выполняемым действиям. Располагается около ограничивающей фигуры

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ НА ЯЗЫКАХ ВЫСОКОГО УРОВНЯ

ПОСОБИЕ
для слушателей специальности переподготовки 1-40 01 73 «Программное обеспечение информационных систем» заочной формы обучения

Составитель **Гридина** Елена Ивановна

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 08.10.21.
Рег. № 42Е.
http://www.gstu.by