

## Оглавление

Введение.....	2
1. Краткая справка по языку Си и разработке консольных приложений в среде Visual C++ 2008.....	4
Вопросы по самопроверке.....	23
2. Программы линейной структуры .....	24
2.1. Средства разработки программ линейной структуры .....	24
Целый тип данных.....	24
Вещественные типы данных .....	25
Стандартные функции для обработки числовых данных .....	26
Оператор присваивания и его сокращенные формы .....	28
Арифметические выражения.....	29
Вывод десятичных чисел в окно программы .....	32
Ввод десятичных чисел с клавиатуры.....	36
2.2 Приемы, используемые для минимизации вычислений.....	38
2.3 Примеры выполнения задания.....	39
Пример 2.1 выполнения задания.....	39
Пример 2.2 выполнения задания.....	40
2.4. Задания А для самостоятельной работы .....	42
2.5. Задания Б для самостоятельной работы.....	45
Вопросы по самопроверке .....	50
3. Программы разветвляющейся структуры.....	51
3.1 Средства разработки программ разветвляющейся структуры.....	51
Условные операторы.....	51
Сложные логические выражения.....	54
Условное выражение (тернарный оператор).....	55
3.2. Примеры выполнения задания .....	55
3.3. Задания для самостоятельной работы .....	60
Вопросы по самопроверке .....	66
4. Программы циклической структуры.....	67
4.1. Средства разработки программ циклической структуры.....	67
Цикл с параметром (for).....	67
Цикл с предусловием (while).....	70
Цикл с постусловием (do while).....	70
4.2. Вычисление и вывод данных в виде таблицы .....	72
4.3. Пример выполнения задания с использованием цикла while .....	76
4.4. Пример выполнения задания с использованием цикла for .....	79
4.5. Задания для самостоятельной работы .....	81
4.6. Сохранение результатов вычислений в массиве .....	95
4.7. Пример выполнения задания.....	97
4.8. Задания для самостоятельной работы .....	99
Вопросы по самопроверке .....	102
Список рекомендуемой литературы.....	104

## Введение

В связи с возрастанием роли информатики в жизни современного общества существенное внимание уделяется и преподаванию аналогичной дисциплины в ВУЗах страны. По сложившейся традиции большое место в курсе информатики в технических университетах занимает раздел, связанный с изучением языков программирования и реализацией на изучаемом языке алгоритмов решения важнейших инженерных задач.

Изучение алгоритмов решения основных инженерных задач (характерных приемов программирования) рассматривается как база для дальнейшего освоения дисциплины, в ходе которого студенты учатся работать с различными типами и структурами данных, разрабатывать алгоритмы решения более сложных задач. Конкретный язык программирования, изучаемый студентами и на котором реализуются рассматриваемые алгоритмы, выступает в этом случае как конкретный инструмент для практического воплощения основных теоретических положений.

В ходе последующего изучения дисциплины при решении более сложных задач и обработке различных типов и структур данных показывается значение простейших алгоритмов как своего рода строительных блоков, на базе которых разрабатывается алгоритм решения поставленной задачи. По сути, уяснив постановку задачи и разрабатывая алгоритм ее решения, студенты должны выделить основные этапы ее решения, которые чаще всего будут представлять собой ранее рассмотренные приемы программирования и алгоритмические конструкции. Разработка и реализация алгоритмов решения задач позволяет попутно добиться еще одного важного результата – формирования основ логического мышления.

Многолетняя практика преподавания курса “Информатика” студентам-первокурсникам свидетельствует о том, что уровень начальной подготовки студентов различается существенным образом, у многих из них отсутствуют умения и навыки логического построения алгоритма решения поставленной задачи. Проблема усугубляется в дальнейшем еще и тем, что в силу разных причин студенты должным образом не осваивают раздел, посвященный разработке и реализации основных типов алгоритмов и характерных приемов программирования.

Написание данного учебного пособия преследовало цель изложить в краткой форме основные принципы и правила построения и программирования алгоритмов различных типов (линейных, разветвляющихся, циклических, в том числе и с вложенными циклами) и характерных приемов программирования, показать использование этих алгоритмов

при решении практических задач. Изложение этого материала сопровождается примерами программ, а также заданиями для выполнения лабораторных работ. В отличие от подобных пособий прошлых лет авторы стремились составить более сложные и интересные задания, причем в рамках одной темы выдержать одинаковый уровень сложности для разных вариантов заданий. Систематическое выполнение предлагаемых заданий позволит студентам подготовиться и успешно решить задачи, предлагаемые при проведении рубежных контролей и на экзамене.

В качестве инструментального средства программной реализации рассматриваемых алгоритмов используется язык С среды программирования MS Visual Studio, который все шире изучается на разных кафедрах университета. В связи с этим авторы сочли необходимым включить раздел, содержащий основные сведения о среде программирования Visual Studio и подготовке в ней консольных приложений, поскольку именно эта среда используется при преподавании курса «Информатика». Данное пособие представляет собой лишь первую часть, представляющую собой фактически введение в программирование. В дальнейшем авторы планируют представить последующие части, охватывающие другие разделы преподаваемой дисциплины.

## 1. Краткая справка по языку Си и разработке консольных приложений в среде Visual C++ 2008

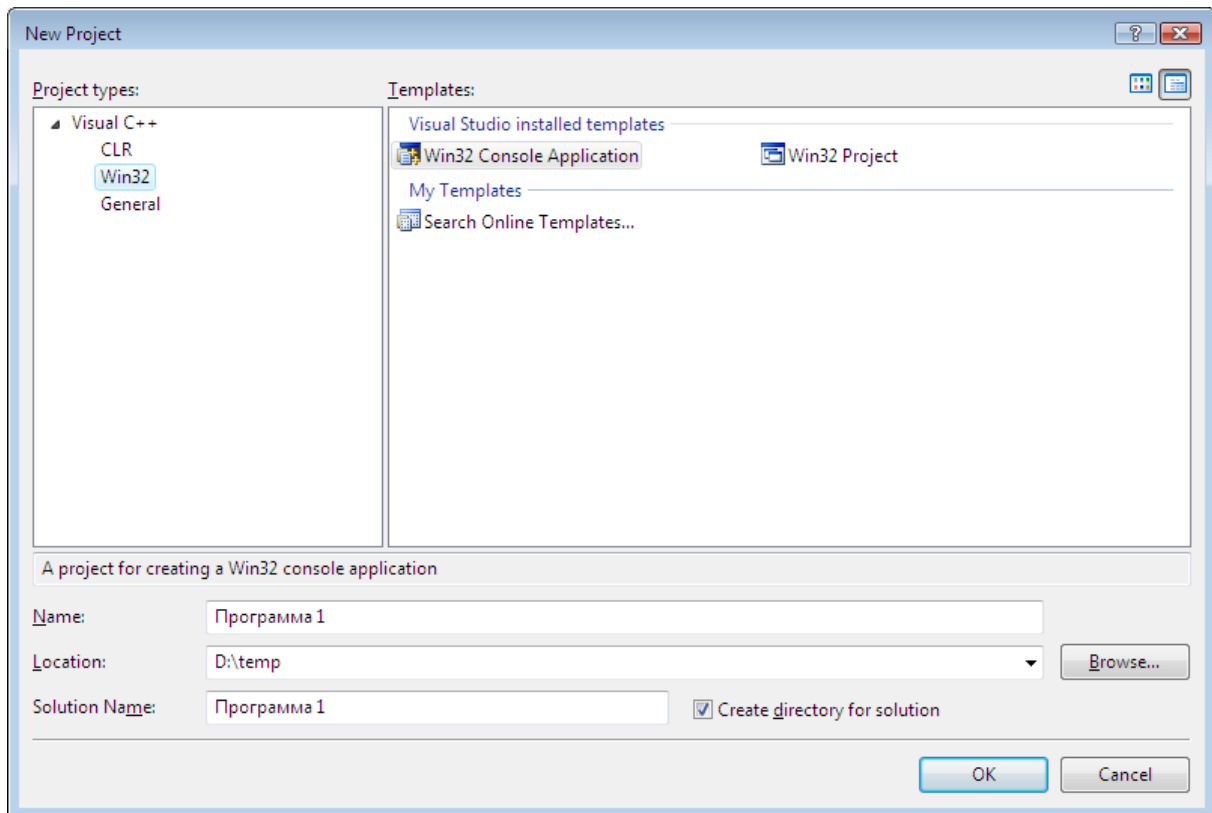
Система программирования Visual C++ предоставляет возможность разработки и отладки различных программных продуктов, в том числе приложений, работающих как с использованием графического интерфейса пользователя, так и в консольном режиме. Последние имеют интерфейс пользователя в виде текстового окна, называемого *окном программы*, в котором последовательно, строка за строкой отображаются данные, вводимые пользователем с клавиатуры, и данные, выводимые программой. Позицию начала ввода или вывода в окне программы указывает *курсор* - мигающий символ, имеющий вид подчеркивания в режиме вставки или прямоугольника – в режиме замены. По умолчанию длина строки равна 80, а количество строк – 50. Изменить эти и другие параметры окна программы позволяет диалог, открывающийся при вводе команды *Свойства* в системном меню окна программы при ожидании ввода данных.

При вводе пользователь имеет возможность редактировать последние вводимые данные, используя клавиши с печатными символами, а также клавиши BackSpace (удаление последнего введённого символа), Delete (удаление символа справа от курсора), Insert (переключение режимов вставки и замены), Стрелка вверх (удаление всех введённых символов), Стрелка влево (перемещение курсора в предыдущую позицию), Стрелка вправо (перемещение курсора в следующую позицию). Если в окне диалога, открывающегося при вводе команды *Свойства* системного меню окна программы, установить на вкладке *Общие* флажок *Выделение мышью*, то становится возможным выделять части текста буксировкой мыши, копировать выделенное в буфер обмена щелчком её правой клавиши и затем вставлять в позицию курсора щелчком правой клавиши. Завершается ввод нажатием клавиши Enter, при этом курсор перемещается в начало новой строки.

Вывод данных из программы выполняется в виде текста, символ за символом при автоматическом перемещении курсора в очередную позицию строки, а при достижении её конца – в начало новой строки.

Консольный режим обычно используется в тех случаях, когда основными требованиями к программе являются минимизация времени счёта и расхода оперативной памяти. На подготовку таких программ требуется меньше времени, поэтому консольный режим удобно использовать для быстрой проверки и отладки отдельных алгоритмов. Именно поэтому, имея в виду, что данный сборник задач ориентирован на развитие на-

чальных навыков алгоритмизации и отладки небольших программ, предполагается использование консольного режима.



**Рис. 1.1**

Для создания программы, работающей в консольном режиме, следует после запуска Visual C++ ввести команду File/New/Project.... В открывшемся окне New Project (см. рис. 1.1) выбрать в левом списке строку Win32, в правом списке – Win32 Console Application, ввести в поле Location: (или выбрать в диалоге, нажав кнопку Browse...) папку для размещения проекта, ввести в поле Name: имя проекта и нажать кнопку ОК, а затем - кнопку Finish во вновь появившемся окне Win32 Application Wizard. При этом изменится набор окон, как показано на рис. 1.2.

В окне редактирования программы (см. рис. 1.2) будет представлен подготовленный системой исходный текст простейшей программы - *шаблон консольной программы* (или просто *шаблон программы*).

Элементами шаблона программы являются

\* *заголовок программы*

`//Программа 1.cpp : Defines the entry point for the console application –`  
комментарий (начинается с пары символов `//`) указывающий, в каком файле проекта находится описание исходного текста программы и что это приложение консольное.

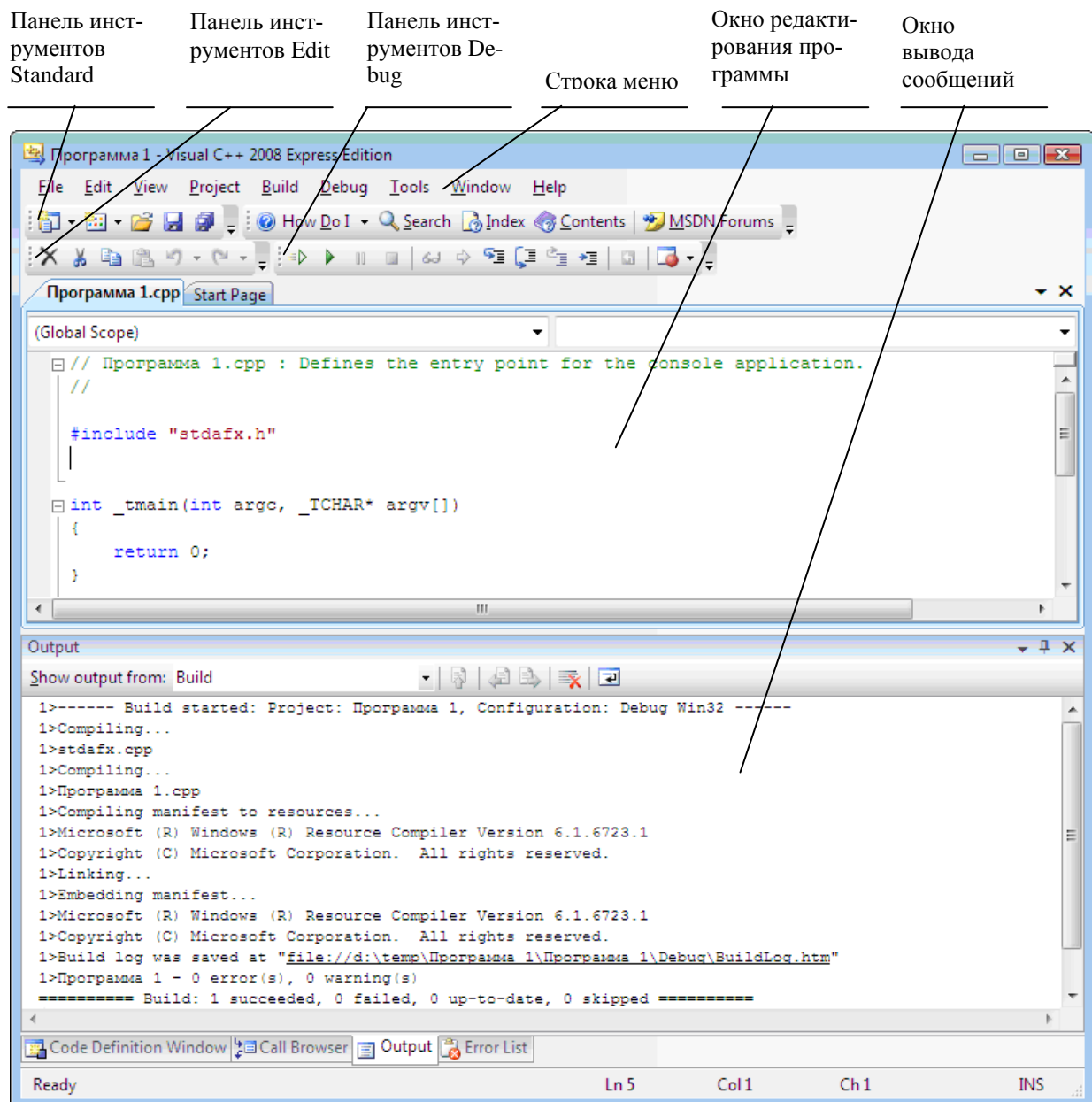


Рис. 1.2

\* *директива #include "stdafx.h", подключающая к шаблону заголовочный файл stdafx.h, необходимый для использования минимального набора стандартных (библиотечных) функций, обеспечивающих работу проекта,*

\* *главная функция (primary function), включающая*


- *заголовок функции*

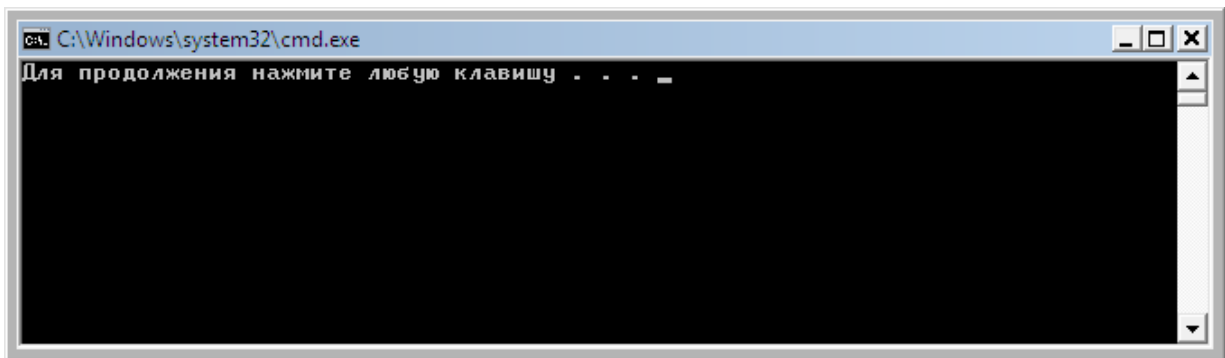
```
int _tmain(int argc, _TCHAR* argv[])
```

- *тело функции – блок из пары фигурных скобок, содержащих лишь один оператор return 0;*


```
return 0;
```

*выхода из программы с кодом завершения 0.*

Для создания файла исполняемой программы шаблона (как и создаваемых на его основе консольных программ), который будет размещён в подпапке Debug папки приложения, следует ввести команду Build/Build Solution. Работающая программа шаблона будет представлять только *окно консольного приложения (окно программы)*. Для её запуска в среде Visual C++ необходимо ввести команду Debug/Start Without Debugging (*старт без отладки*), или щелкнуть кнопку , или нажать клавиши Ctrl+F5. В результате откроется окно программы (см. рис 1.3) с предложением нажать любую клавишу, будет выполнен единственный оператор шаблона `return 0;`, завершающий работу программы, а нажатие любой клавиши приведёт к закрытию её окна.



**Рис 1.3.**

При запуске программы шаблона командой Debug/Start Debugging (*старт в режиме отладки*), или щелчком на кнопке , или нажатием клавиши F5, или при запуске из папки окно программы также откроется, но сразу же закроется, и работа программы будет завершена. Чтобы задержать окно программы до нажатия клавиши, в программу следует добавить перед оператором `return 0;` оператор `getch();` и директиву `#include "conio.h"` как показано ниже

```
#include "stdafx.h"
#include "conio.h"

int _tmain(int argc, _TCHAR* argv[])
{
    getch();
    return 0;
}
```

Впрочем, закрыть окно и завершить работу программы можно и любым другим способом закрытия окна Windows.

В дальнейшем при разработке программы в шаблон потребуется добавлять операторы, объявления переменных и именованных констант, комментарии, директивы, создаваемые пользователем функции, а также, при необходимости, директивы `#include`, подключающих к проекту иные наборы библиотечных и пользовательских функций.

*Комментарии* никак не влияют на создание исполняемой программы и просто пропускаются компилятором. Назначение же комментариев состоит в том, чтобы облегчить понимание алгоритма программы при чтении её исходного текста. Оформить текст в виде комментария можно одним из следующих способов:

\* поместив символы `//` перед текстом строки, например,  
`//оператор getch();` заставит программу  
`getch();` `//ждать нажатия клавиши`

\* заключив текст в скобки вида `/*` и `*/`, например,  
 текст, занимающий несколько строк,  
`/* составить программу вычисления времени движения тела`  
`с ускорением 9,8 м/с^2 на пути заданной длины,`  
`вводимой с клавиатуры */`

или поясняющий текст внутри конструкции языка, если комментарий не искажает их смысл и правила построения

```
int _tmain(int argc, _TCHAR* argv[/*массив аргументов*/])
```

или

```
int _tmain(int argc/*количество аргументов*/, _TCHAR* argv[])
```

Нельзя использовать этот комментарий, например, после знака `,` внутри задаваемых пользователем имён, внутри строковых констант (текстов, начинающихся и заканчивающихся символом `"`, например, `"conio.h"`) и других таких же комментариев.

*Операторы* программы служат для описания её алгоритма. Конструкция оператора определяет, какие действия должны быть выполнены по обработке данных и\или передаче управления от оператора к оператору. В соответствии с правилами построения различают *составные операторы* или *блоки*, *структурные операторы* и *простые операторы*.

В этом разделе ограничимся рассмотрением некоторых простых операторов и блоков.

Простые операторы могут выражать некоторое законченное действие и завершаться знаком `;` (точка с запятой), например, `getch();` или `return 0;` или `q++;` (увеличить



значение переменной `q` на 1), а также быть частью выражений, например, `s=s+q++` (увеличить значение переменной `s` на значение переменной `q`, а затем увеличить значение переменной `q` на 1) или частью списков с разделителем «запятая» внутри структурных операторов (см. один из примеров организации цикла `for` в разделе 4.1).

Блоки (составные операторы) начинаются с открывающей фигурной скобки (`{`) и заканчиваются парной закрывающей фигурной скобкой (`}`). Они используются в качестве тел функций (например, тело функции `_tmain`, см. выше), могут быть частями структурных операторов или иметь самостоятельное значение. Их особенностью является то, что они могут содержать, помимо вложенных в них других операторов, в том числе и составных, объявления локальных переменных, используемых («видимых») только внутри этих блоков. Операторы, входящие в составной, могут описывать сложный алгоритм, выполнение которого начинается при входе в составной оператор. Завершение алгоритма происходит либо при выполнении оператора, меняющего естественный порядок их выполнения (в порядке их следования в тексте программы), например, оператором `return` выхода из функции, либо при достижении парной закрывающей фигурной скобки.

Обрабатываемые данные хранятся в ячейках оперативной памяти, а *переменные* и *константы* представляют их в исходном тексте программы. Отличие переменной от константы состоит в том, что переменным можно присваивать новые значения, а значения констант задают только в исходном тексте программы и нельзя изменять их при её выполнении. В простейшем случае для объявления переменной используется конструкция

```
<Имя типа> < > <Имя переменной>;
```

где

<Имя типа> - один из предопределенных типов: `int` (целый), `float` (вещественный), `double` (вещественный удвоенной точности), `char` (символьный),

< > - пробел,

<Имя переменной> - последовательность латинских букв, цифр, знаков `$` и `_` (подчерк), начинающаяся с буквы, знака `$` или `_`.

Вместо одной можно описать несколько переменных, записав их имена через запятую.

При описании переменных им можно задавать *начальные значения*, которые будут иметь переменные в момент начала выполнения программы. Начальное значение

переменной должно записываться вслед за знаком = после её имени (знак «равно» имеет в Си смысл присваивания переменной значения).

Помимо обычных констант, значения и типы которых определяются по их форме записи, можно использовать и *именованные константы* (иногда их называют переменными-константами), которые представляют значения своими именами. Именованные константы объявляются подобно переменным с начальными значениями, но перед их именами необходимо ставить слово `const`.

```
int x, y=-5; /* Объявление двух переменных типа int.
             Переменная y будет иметь начальное
             значение -5*/
int const n = 10; /* Объявление именованной константы типа
                  int */
```

Именованные константы в ряде случаев позволяют облегчить понимание алгоритма программы и избежать ошибок при необходимости изменения значения. Например, если число 10 используется для объявления размеров нескольких массивов и в операторах обработки этих массивов, то желательно использовать именованную константу со значением 10. Тогда, при необходимости замены числа 10 другим, достаточно будет сделать эту замену только в объявлении именованной константы (см. пример программы в разделе 4.6).

Рассмотрим следующий пример программы, вычисляющей время движения тела с ускорением  $9,8 \text{ м/с}^2$  на пути заданной длины, вводимой с клавиатуры.

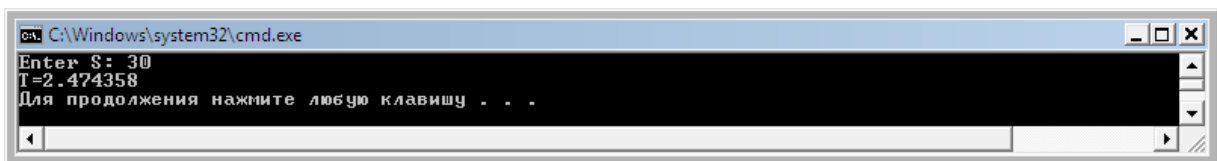
```
#include "stdafx.h"
#include "conio.h" /*подключение библиотечных подпрограмм
                  консольного ввода-вывода, в частности getch(); */
#include "math.h" /* подключение библиотечных подпрограмм
                  вычисления математических функций
                  */
//главная функция программы
int _tmain(int argc, _TCHAR* argv[])
{
    // ОБЪЯВЛЕНИЯ ПЕРЕМЕННЫХ
```

```

float const g=9.8; /*Именованная константа,
                    представляющая ускорение. */
float T; /*Переменная вещественного типа, которая будет
          представлять искомое время. */
int S; //Переменная целого типа, представляющая длину пути.
      // ОПЕРАТОРЫ
//Вывод приглашения к вводу значения переменной S
printf("Enter S: ");
//Ввод значения переменной S
scanf("%d",&S);
//Вычисление времени движения тела на заданном пути
T=sqrt(S*2/g);
//Вывод результата с пояснениями
printf("T=%f\n", T);
//Задержка закрытия окна программы до нажатия клавиши
getch();
return 0;
}

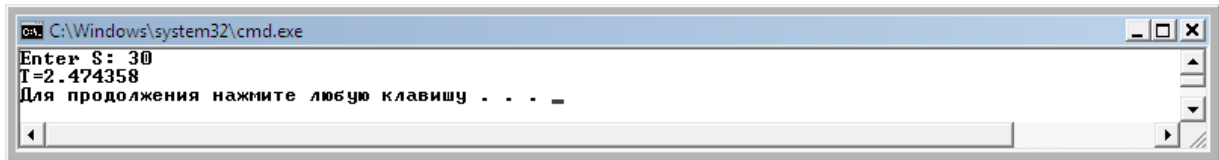
```

В блоке главной функции `_tmain` программы используется объявление именованной константы (константа `g`), объявления переменных `T` и `S`, а также операторы, представляющие алгоритм решения задачи. Протокол ввода-вывода программы представлен в окне программы на следующем рисунке.



Результат выводится с поясняющим кратким текстом, набранным в латинском алфавите.

Цвет фона (чёрный) и символов (белый) в окне программы можно изменить следующим образом: раскрыть системное меню окна щелчком на кнопке , выбрать пункт Свойства, в появившемся окне диалога на вкладке Цвета из представленной палитры выбрать для фона и символов желаемые цвета, нажать кнопку ОК. Например, при выборе белого цвета для фона и чёрного для символов окно программы примет вид



Используемая в Visual C++ кодировка символов не обеспечивает правильный вывод букв кириллицы, что представляет неудобство для русскоязычных пользователей. Наиболее простым способом обеспечения вывода текстов на русском языке является включение в программу объявления функции Ruc перекодирования символов как показано ниже.

```
#include "stdafx.h"
#include "conio.h"
// Подключение библиотечных подпрограмм
// вычисления математических функций.
#include "math.h"

//заголовок функции Ruc, описание см. ниже.
char* Ruc(char s[]);

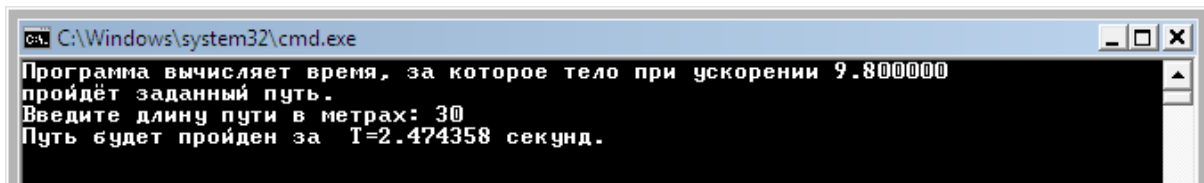
int _tmain(int argc, _TCHAR* argv[])
{
    // ОБЪЯВЛЕНИЯ ПЕРЕМЕННЫХ
    float const g=9.8; /*Именованная константа,
    представляющая ускорение. */
    float T; /*Переменная вещественного типа, которая будет
    представлять искомое время. */
    int S; //Переменная целого типа, представляющая длину пути.
    // ОПЕРАТОРЫ
    //Вывод приглашения к вводу значения переменной S.
    //Знак \ в конце строки первой строки оператора обозначает
    //перенос текста константы на начало новой строки.
    printf(Ruc("Программа вычисляет время, за которое тело \
    при ускорении %f\n пройдёт заданный путь.\n"), g);
    printf(Ruc("Введите длину пути в метрах: "));
    //Ввод значения переменной S.
    scanf("%d",&S);
```

```

//Вычисление времени движения тела на заданном пути.
//Математическая функция sqrt извлекает квадратный корень.
T=sqrt(S*2/g);
//Вывод результата с пояснениями.
printf("%s T=%f", Ruc("Путь будет пройден за "), T);
printf("%s\n", Ruc(" секунд."));
//Задержка закрытия окна программы до нажатия клавиши.
getch();
return 0;
}
char* Ruc(char s[])
{ //Функция перекодирования русских букв
  //для вывода в окно программы.
  int i;
  static char ss[257];
  for (i=0;s[i] != '\0'; i++)
  {
    if (s[i]>= -64 && s[i] <= -17)
      ss[i]=(-64+s[i]); //А..п
    else if (s[i]>= -16 && s[i] <= 0)
      ss[i]=(char)(-16+s[i]); //р..я
    else if (s[i] == -72)
      ss[i]=(char)(-15); //ё
    else if (s[i] == -88)
      ss[i]=(char)(-16); //Ё
    else
      ss[i]=s[i];
  }
  ss[i]='\0';
  return ss;
}

```

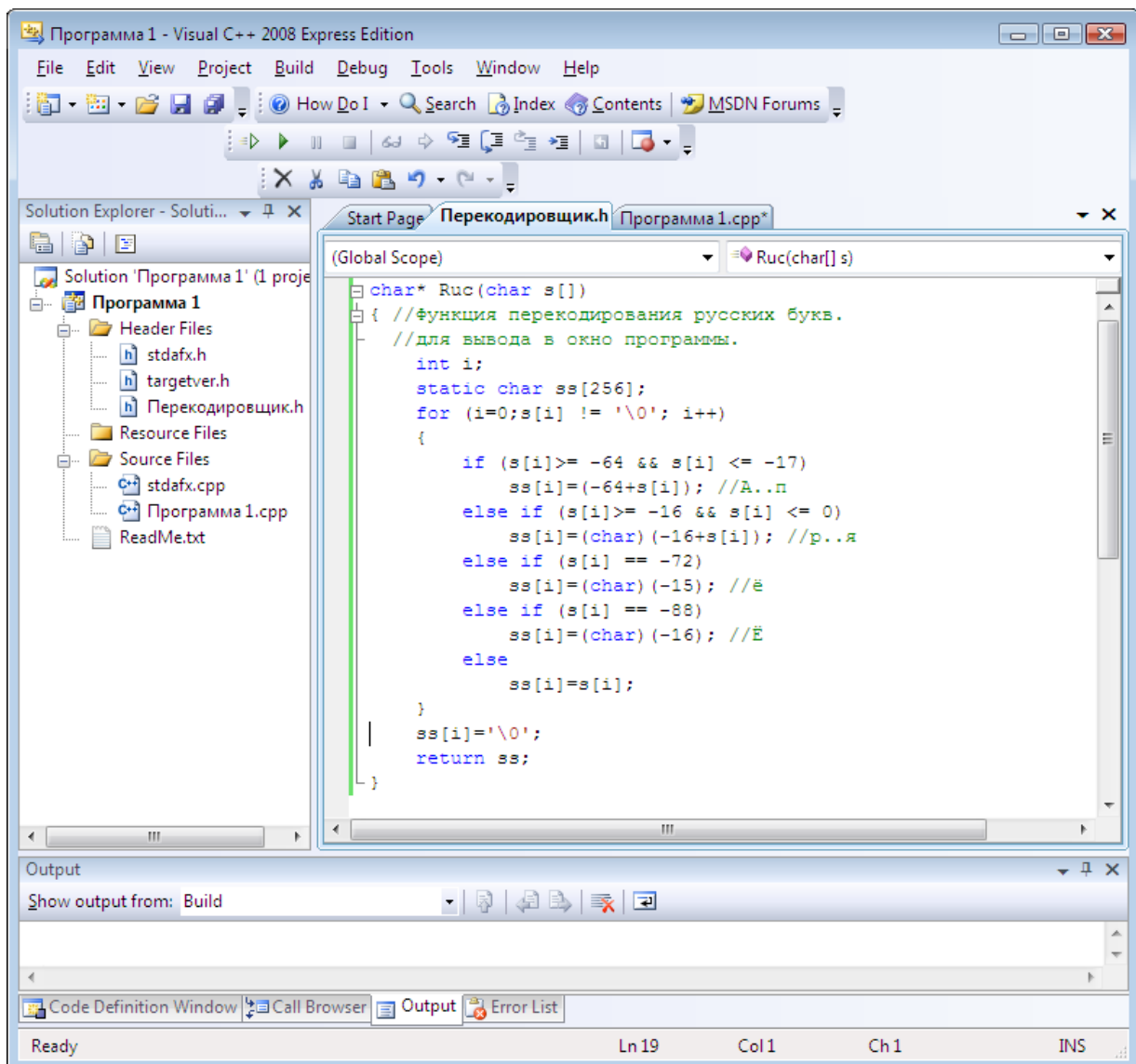
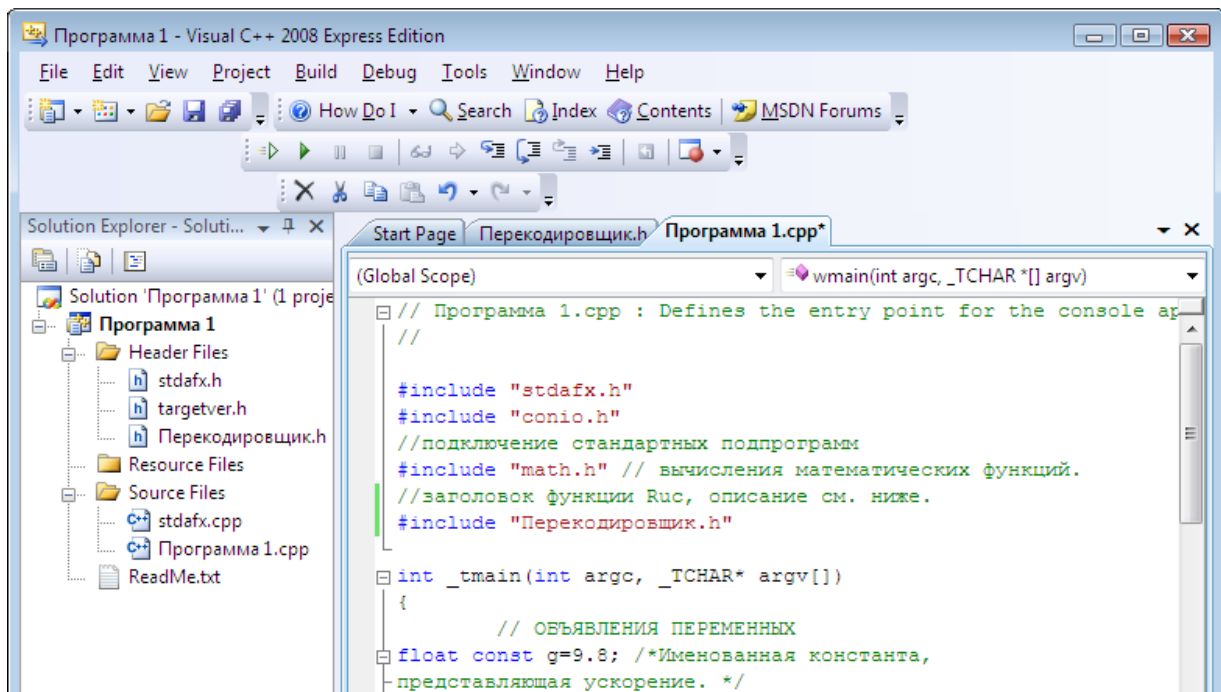
Протокол ввода-вывода программы будет иметь вид

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The window contains the following text:

```
Программа вычисляет время, за которое тело при ускорении 9.800000  
пройдёт заданный путь.  
Введите длину пути в метрах: 30  
Путь будет пройден за T=2.474358 секунд.
```

В исходном тексте программы использована, помимо главной функции `_tmain`, дополнительная функция `Ruc`. Дополнительные функции можно размещать как перед главной функцией, так и после неё. Но если функция размещена после главной или любой другой, из которой она вызывается, то необходимо давать опережающее объявление её заголовка (как и сделано в приведённом примере, так как функция `Ruc` размещена после главной).

Используемые в программе дополнительные, помимо главной, функции можно также размещать в одном или нескольких отдельных файлах. Чтобы разместить дополнительные функции при их создании в среде Visual C++ в отдельном файле следует ввести команду `Project/Add New Item...`, в открывшемся окне выделить элемент списка `Header File.h`, ввести имя, которое хотите дать файлу (например, `Перекодировщик`), и щелкнуть на кнопке `Add`. В результате в папке проекта появится новый файл, в списке `Solution Explorer` (открываемом командой `View/Solution Explorer`) будет размещен соответствующий ему элемент, а также в окне Visual C++ появится новая вкладка для ввода описаний функций, озаглавленная именем созданного файла (`Перекодировщик.h`). Затем перед заголовком главной функции следует записать директиву `#include` с именем файла (`#include "Перекодировщик.h"`). Пример представляют следующие рисунки.



Если файл с добавляемыми функциями был создан ранее (уже существует), то в файл главной функции нужно просто добавить директиву `#include` со ссылкой на него (например, `#include "D:\temp\Перекодировщик.h"`).

Одним из важных этапов разработки программ является их отладка – локализация и устранение ошибок, препятствующих созданию исполняемого файла (синтаксических ошибок), и ошибок, вносящих несоответствие программы заданию на её разработку (смысловых ошибок).

Синтаксические ошибки выявляет система программирования при выполнении команды `Build/Build Solution` или `Build/Rebuild Solution`. Сообщения об ошибках выводятся в окно `Output` построчно. Они содержат краткие пояснения о причинах ошибок, номера строк исходного текста программы, в которых ошибки обнаружены, а двойным щелчком на строке сообщения курсор устанавливается на соответствующей строке программы.

В `Visual C++` многие библиотечные функции языка `C` имеют усовершенствованный аналог, который предлагается использовать в сообщениях об ошибках типа *warning*. Однако такие ошибки не препятствуют созданию исполняемого файла. Другой тип ошибок отмечен в сообщениях словом *error*. В большинстве случаев эти ошибки связаны с нарушением правил языка `C` (например, не дано опережающее объявление заголовка функции, размещенной после главной, не поставлена знак «точка с запятой» после объявления переменной, нет соответствия между открывающими и закрывающими скобками) и вызовом не существующих функций. Такие ошибки препятствуют созданию исполняемого файла и их необходимо устранять.

Поиск и устранение смысловых ошибок более сложен. Существующие методики в большинстве случаев не могут дать гарантий на отсутствие таких ошибок, однако без их применения разработку программы нельзя считать законченной. К числу таких методик относится *тестирование* всей программы и её частей. Тестирование всей программы состоит в подборе тестовых исходных данных для работы программы, для которых известны, в соответствии с заданием, требуемые результаты работы программы, и сравнение с ними реальных результатов работы программы. Если реальные результаты не совпадают с требуемыми, значит программа не соответствует заданию и необходимо выявить и устранить причину этого несоответствия. С этой целью может использоваться как отладочная печать, так и имеющиеся в `Visual C++` средства отладки.



Обычно программу можно разделить на имеющие законченный смысл алгоритмы (подзадачи), выполняемые последовательно или являющиеся частями более сложных алгоритмов (подзадач). В простейшем случае, как в представленной ранее программе вычисления времени прохождения тела в вакууме заданного пути при заданном ускорении, в программе можно выделить части: ввод исходных данных, их обработка и вывод результатов.

Части программы можно отлаживать в любом порядке. Однако желательно начать с той части, которая представляет суть задания. В большинстве случаев это часть обработки данных, и именно она наиболее сложна в отладке. Приступить к ней, минуя ввод исходных данных, позволяют следующие средства системы программирования:

- \* задание начальных значений переменных,
- \* подготовка, как рабочего, так и отладочного вариантов программы и быстрой замены одного варианта другим,
- \* слежение в среде Visual C++ за значениями переменных и изменение их при выполнении программы в отладочном режиме,
- \* использование различных способов приостановки программы и последующего продолжения её выполнения.

Рассмотрим эти приёмы и средства более подробно.

Отладочный вариант программы, помимо начальных значений исходных данных, может содержать коды задания значений промежуточным данным и отладочный вывод, которые должны отсутствовать в рабочем варианте программы. Реализуется возможность быстрого перехода от рабочего кода программы к отладочному (и обратно) с помощью директив условной компиляции.

Например, вставив в код программы директиву `#define u1` (директива определения имени, в данном случае имени `u1`), везде ниже по тексту можем размещать конструкции

```
#ifdef u1
... // код
... // используемый для
... // отладки
#endif
```

заставляя компилятор обрабатывать код между директивами `#ifdef u1` (если `u1` определено) и `#endif` (конец блока `#ifdef`), то есть рассматривать его как часть кода про-

граммы. Чтобы исключить использование в программе отладочного кода, достаточно удалить директиву `#define ul` или превратить строку с ней в комментарий.

Подобный результат можно получить для отдельной отладочной вставки с помощью конструкции

```
///  
... // код  
... // используемый для  
... // отладки  
//*/
```

позволяющей добавить отладочный код в программу. Чтобы превратить всю конструкцию в комментарий достаточно в первой строке переместить первые два знака `//` в её конец (первая строка примет вид `/*//` и станет началом многострочного комментария).

Возможно использование рассмотренных конструкций и для временного удаления части кода рабочего варианта программы, если это ускорит процесс отладки других частей. Например, если программа требует ввода значительных объёмов исходных данных, то при её отладке целесообразно временно заменить ручной ввод заданием и редактированием начальных значений переменных.

Директив `#define` в программе может быть несколько. Чтобы определяемые директивами `#define` имена указывали на отладочный вариант программы в сочетании с директивой `#ifdef`, они должны быть информативными, например, `#define debug1`, `#define debug2` и тому подобными. Тогда временно удаляемые при отладке части кодов рабочей программы должны начинаться с директивы `#ifndef` (если имя не определено), а отладочные вставки кодов – с директивы `#ifdef`. Если в программе отладочный код должен замещать часть кода рабочей программы, то, например, при наличии директивы `#define debug1` следует использовать конструкцию

```
#ifdef debug1 //если имя debug1 определено, добавить  
... // код, используемый для  
... // отладки  
#else //иначе, т.е. если имя debug1 не определено,  
    //использовать  
... // часть кода  
... // рабочей  
... // программы  
#endif
```

Дополнительную информацию по директивам условной компиляции можно найти в справочной системе Visual C++.

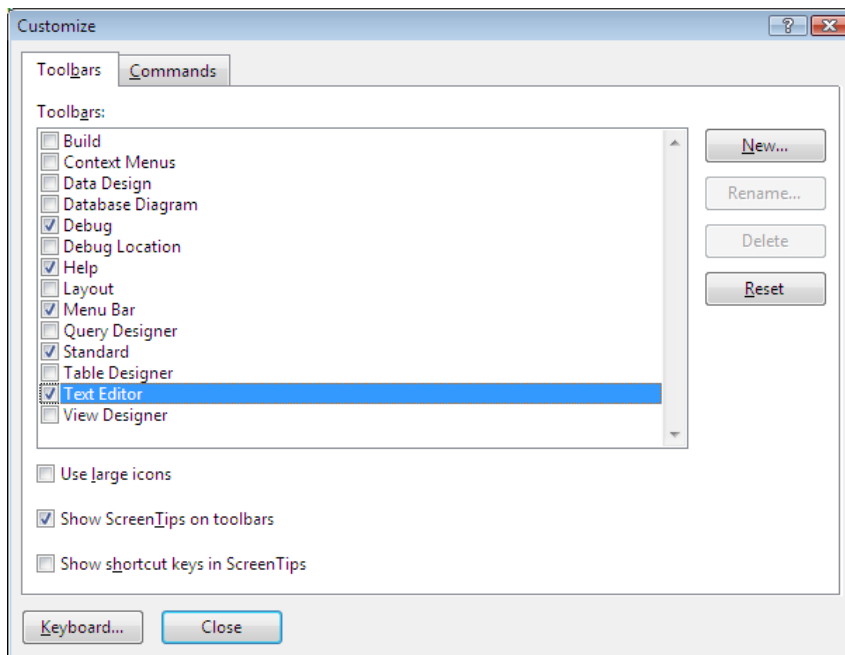
Чтобы ускорить поиск и исправление ошибок в программе желательно подготовить в окне Visual C++ минимальные наборы инструментов панели отладки (Debug)



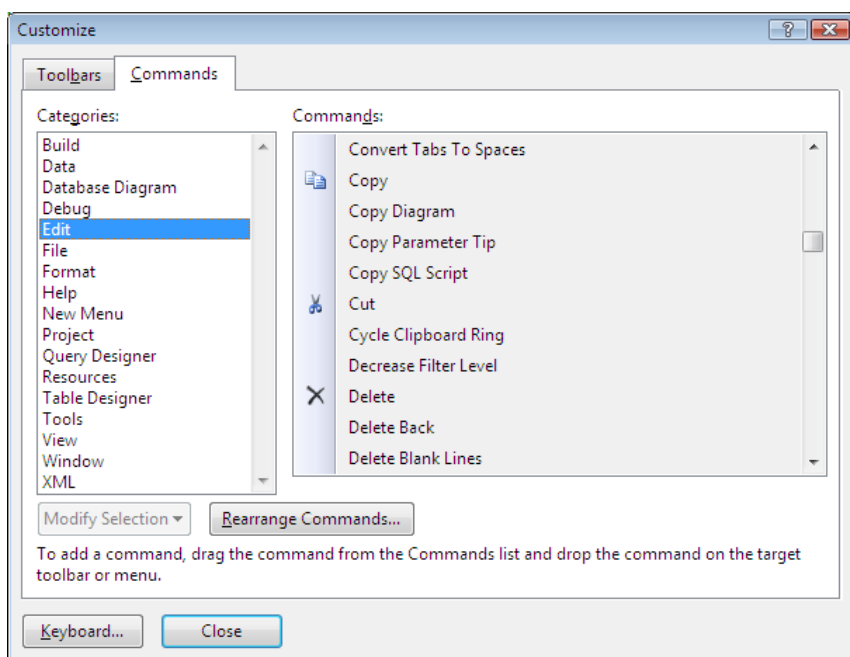
и панели редактирования текста (Edit)



Для этого в меню Tools следует выбрать пункт Customize... и в открывшемся окне





установить флажки на строках Debug и Text Editor. Затем на вкладке Commands,





выбрав в левом списке строку Edit, отбуксировать из правого списка на панель инструментов редактирования Edit окна Visual C++ недостающие значки. Аналогично выполнить настройку панели инструментов отладки, выбрав в левом списке окна Customize строку Debug. Чтобы удалить ненужные значки с панелей инструментов, их просто следует отбуксировать с панелей в любое место при открытом окне Customize.


Инструменты панели Debug имеют следующее назначение:


 - запустить программу в обычном режиме, без использования отладочных средств среды Visual C++.


 - запустить программу в режиме отладки. В этом режиме в программу будут автоматически добавлены коды, обеспечивающие использование отладочных средств (см. далее).

 - завершить работу программы, запущенной в режиме отладки, и вернуться к редактированию её кода.

 - добавить в окно наблюдения Watch строку слежения за текущим значением переменной или выражения: выражение следует выделить, а переменную либо выделить, либо поставить перед ней курсор и щелкнуть на этом значке. Можно также получить и всплывающую подсказку по значению, подведя курсор к переменной или выделенному выражению. (В VS C++ 2008 есть окно Autos, где автоматически отображаются локальные переменные текущего блока или оператора)

 - выполнить шаг трассировки без захода в подпрограммы (трассировка без захода в подпрограммы – это выполнение программы строка за строкой при каждом щелчке на кнопке, без захода в вызываемые в строке функции).

 - трассировка с заходом в подпрограммы (если в текущей строке программы есть вызовы функций, то после щелчка на кнопке появится код функции для выполнения его трассировки).

 - завершение выполнения кода функции и возврат управления в вызвавшую её программу.

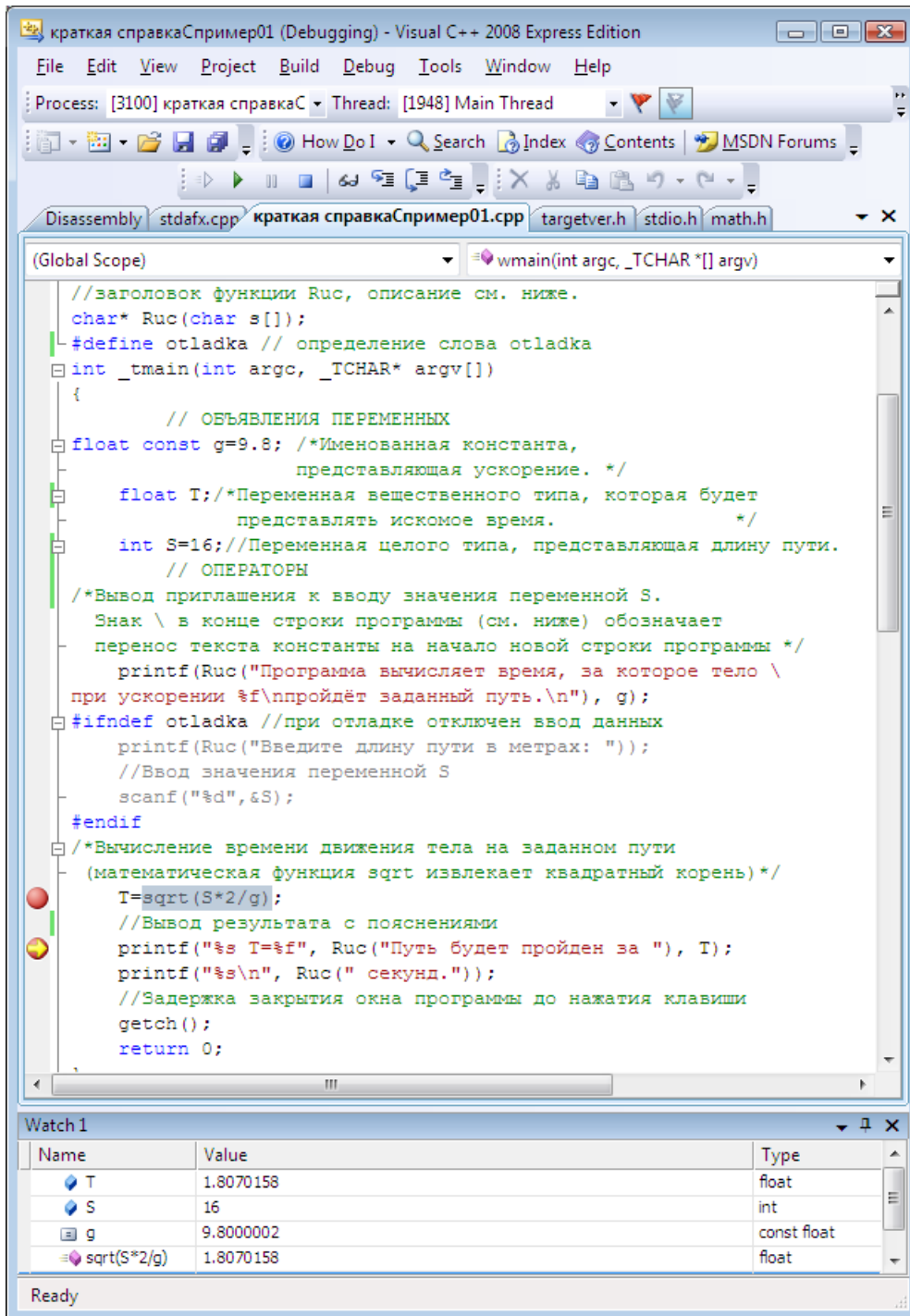
При отладке обычно используют точки безусловного и условного останова (последние рассматривать не будем). Строка программы является точкой безусловного останова, если при работе программы в отладочном режиме выполнение программы будет остановлено непосредственно перед этой строкой. Таких точек в программе может быть несколько. Сделать строку точкой безусловного останова можно выбором в контекстном меню строки пункта Breakpoint/insert Breakpoint. В результате на серой вертикальной по-



лосе перед строкой появится красный кружок – признак точки безусловного останова. Такую установку можно выполнить просто щелчком на указанной полосе, а отменить – щелкнув ещё раз.

Использование в отладочном варианте программы

- \* начального значения переменной S и отключение её ввода директивами условной компиляции,
- \* точек безусловного останова и
- \* окна наблюдения за текущими значениями переменных и выражения  $\text{sqrt}(S^2/g)$

поясняет следующий рисунок.



После запуска программы в отладочном режиме (щелчком на кнопке ) её выполнение будет приостановлено в первой точке безусловного останова, отмеченной красным кружком (перед выполнением строки  $T = \sqrt{S \cdot 2 / g}$ ); отмеченной красным кружком), а при повторном щелчке на кнопке  - перед строкой

```
printf("%s T=%f", Ruc("Путь будет пройден за "), T);
```

## Вопросы по самопроверке

1. Когда целесообразно использовать консольные приложения?
2. Какой интерфейс пользователя имеют консольные приложения?
3. Что включает шаблон консольного приложения?
4. В чем назначение операторов программы?
5. Какую конструкцию программы называют блоком?
6. В чем назначение переменных программы?
7. Что такое именованная константа?
8. В каких случаях целесообразно объявлять переменные с начальными значениями?
9. Для чего предназначены директивы `#include`?
10. Какие виды ошибок бывают в программе?
11. Такие ошибки препятствуют созданию исполняемого файла?
12. Какими способами можно запустить консольное приложение?
13. Что понимают под отладкой программы?
14. Что такое тестирование?
15. Какие средства есть в Visual C++ для поиска смысловых ошибок в программе?
16. Что называют точкой безусловного останова?
17. Как приостановить на заданной строке программы её выполнение в отладочном режиме?
18. Какие бывают виды трассировки?
19. Какими способами можно наблюдать за значениями переменных в процессе отладки программы?
20. Как создать панель инструментов отладки или изменить набор её инструментов?
21. Для чего предназначены комментарии в программе?
22. Какие есть виды комментариев?
23. Как можно быстро превратить часть строк программы в комментарий?
24. Как можно создать отладочный вариант программы, используя директивы условной компиляции, чтобы после отладки можно было быстро восстановить рабочий вариант программы?

## 2. Программы линейной структуры

*Программой линейной структуры* называется такая программа, каждый оператор которой выполняется один и только один раз.

Она может строиться только из простых операторов, не меняющих естественный порядок вычислений, а именно, из операторов присваивания и вызова функций. Из числа последних в этом разделе нас будут интересовать только вызовы функций ввода и вывода для стандартных устройств – клавиатуры и монитора.

### 2.1. Средства разработки программ линейной структуры

Рассмотрение вопросов алгоритмизации задач и приёмов программирования удобнее всего проводить на примерах обработки числовых данных. Рассмотрим в первую очередь некоторые *стандартные типы* (имеющиеся в С и не требующие объявления в программе) числовых данных.

#### Целый тип данных

К числу стандартных целых типов относятся:

`int` – тип, представляющий целые со знаком от  $-2^{31}$  до  $+2^{31}-1$  (от -2147483648 до 2147483647), занимает 4 байта.

`unsigned int` – тип, представляющий целые без знака (на что указывает модификатор `unsigned`) от 0 до  $+2^{32}-1$  (от 0 до 4294967295), занимает 4 байта.

```
// объявление целых переменных I и K как целых со знаком
int I, K=-5;
// объявление целых переменных i и k как целых без знака
unsigned int i, k=5;
//при выполнении программы значения переменных можно изменять
// объявление именованной целой константы со знаком
const int Nmax=10;
//при выполнении программы значения констант нельзя изменять
```

Для данных целого типа определены следующие арифметические операции (операторы, используемые только в выражениях, то есть как составная часть других операторов). Результат их выполнения также будет иметь тип целый:

изменение знака (унарный минус -),  
сложение (знак +),



вычитание (знак -),  
 умножение (знак \*),  
 целочисленное деление (знак /),  
 взятие по модулю (знак %).

Результатом выполнения операции / является целая часть частного, а операции % – остаток от целочисленного деления (знак остатка всегда совпадает со знаком делимого). Например,

```
int I, K;
I = -5 / -2; //I получит значение 2
K = -5 % -2; //K получит значение -1
```

Частью выражений целого типа могут быть также вызовы функций, возвращающих целые значения, и операторы присваивания целых значений (см. ниже). Если же хотя бы одна составляющая выражения имеет вещественный тип, то вычисленное значение всего выражения будет иметь вещественный тип.

## Вещественные типы данных

К числу *стандартных вещественных (действительных) типов* относятся:

`float` - тип, представляющий вещественные числа со знаком с абсолютными значениями от  $1.175494351 \cdot 10^{-38}$  до  $3.402823466 \cdot 10^{+38}$ , занимает 4 байта.

`double` - тип, представляющий вещественные числа со знаком удвоенной точности с абсолютными значениями от  $2.2250738585072014 \cdot 10^{+308}$  до  $1.7976931348623158 \cdot 10^{+308}$ , занимает 8 байтов.

Например, чтобы объявить переменные с именами `r` как `float` и `R` как `double`, в программе следует записать

```
float r;
double R;
```

Константы вещественного типа `double` записываются либо в *естественной форме*, например, `-12.345`, либо в *экспоненциальной форме*, в которой то же самое число можно записать по-разному, например, `-0.12345E+2`, или `-0.12345E2`, или `-0.12345e+2`, или `-1.2345E+1`, или `-1.2345E1`, или `-12.345E0`, или `-1234.5E-2`, или `-12345E-3` и т.д. При представлении числа в такой форме безразлично, используется строчная или прописная латинская буква E. Чтобы получить значение числа, представленного в экспоненциальной форме, нужно умножить *мантиссу*, то есть то, что сто-

ит перед символом E, на 10 в степени, значением которой является *порядок*, то есть целое число, записанное после E.

Так, константу  $-0.12345E+2$  следует читать как  $-0,12345 \cdot 10^{+2}$ , а константу  $-1234.5E-2$  – как  $-1234,5 \cdot 10^{-2}$ . Константы вещественного типа `float` также записываются либо в естественной форме, либо в экспоненциальной форме, но с суффиксом f или F, например,  $-12.345f$ , или  $-0.12345E2f$ , или  $12.345F$ , или  $-0.12345E2F$  и т.д. В большинстве случаев суффиксы не используются, так как компилятор распознаёт тип константы по месту её использования в программе.

Следующий фрагмент программы представляет объявления вещественных переменных X и Y типа `float`, Z – типа `double` и именованной константы H со значением 0,00000025:

```
float X , Y;
double Z;
const float H=2.5E-7;
```

Для данных вещественного типа определены следующие арифметические операции, результат выполнения которых также будет иметь вещественный тип, представляющий минимальный диапазон, включающий вычисленное значение:

- изменение знака (унарный минус -),
- сложение (знак +),
- вычитание (знак -),
- умножение (знак \*),
- деление (знак /).

В отличие от языков программирования BASIC и Fortran, в языке C нет операции возведения в степень.

## Стандартные функции для обработки числовых данных

Основные элементарные математические функции реализованы в C с помощью стандартных *библиотечных подпрограмм - математических функций*, подключаемых к программе директивой `#include "math.h"`.

Библиотечные математические функции возвращают результаты вещественного типа либо `double`, либо `float`, а их вещественные аргументы (то есть фактические параметры, которыми могут быть константы, переменные, выражения) должны быть соответствующего типа.

- `acos(X)` – возвращает значение арккосинуса аргумента,  
`asin(X)` – возвращает значение арксинуса аргумента,  
`atan(X)` – возвращает значение арктангенса аргумента  
в диапазоне от  $-\pi/2$  до  $+\pi/2$ ,  
`ceil(X)` – возвращает в вещественной форме наименьшее целое значение,  
большее или равное аргументу,  
`cos(X)` – возвращает значение косинуса аргумента,  
`exp(X)` – возвращает значение  $e^x$ ,  
`fabs(X)` – возвращает абсолютное значение аргумента,  
`floor(X)` – возвращает в вещественной форме наибольшее целое значение,  
меньшее или равное аргументу,  
`log(X)` – возвращает значение натурального логарифма аргумента,  
`log10(X)` – возвращает значение логарифма аргумента X по основанию 10,  
`sqrt(X)` – возвращает значение квадратного корня аргумента,  
`pow(X, Y)` – возвращает значение X, возведенное в степень Y,  
`sin(X)` – возвращает значение синуса аргумента,  
`tan(X)` – возвращает значение тангенса аргумента,

Имена представленных стандартных функций соответствуют типу `double`. Для стандартных функций, возвращающих значения типа `float`, используются те же имена, но с суффиксом `f`, например `acosf(X)`, `fabsf(X)`, `powf(X, Y)`. Если не указывать суффикс `f`, то компилятор будет распознавать тип результата, вычисляемого функцией, по типу аргументов. Для функций, имеющих несколько аргументов, допустимы лишь определённые сочетания типов параметров.

Получить быструю подсказку по допустимым типам аргументов и результата, вычисляемого функцией, можно во всплывающем окне, если подвести курсор мыши к оператору вызова функции (см. рисунок – пример подсказки для функции `pow`). Из справки видно, что вызов оформлен правильно (компилятор не выдаст сообщение об ошибке), если X имеет любой вещественный тип и второй параметр целого типа, а результат будет иметь тот же тип, что и X.

`pow(Y, 3)`

```
double pow(double _X, double _Y)
double pow(double _X, int _Y)
float pow(float _X, float _Y)
float pow(float _X, int _Y)
long double pow(long double _X, long double _Y)
(+1 overloads)
```

Дополнительные сведения по стандартным функциям C можно найти в справочном разделе Floating-Point Support при поиске по этому имени на вкладке Search, или выбрав math routines при поиске по индексу.

## Оператор присваивания и его сокращенные формы

Некоторые примеры операторов присваивания уже приводились ранее в этом разделе и в разделе «Краткая справка по языку C и разработке консольных приложений в среде Visual C++ 2008». Однако этими примерами не исчерпывается всё многообразие форм записи и правил использования операторов присваивания языка C. Ниже в данном и следующих разделах приведено более детальное рассмотрение этих вопросов.

Одним из знаков оператора присваивания, как указывалось ранее, является знак =. Справа от него записывается выражение (в частном случае константа, переменная или вызов функции), а слева – переменная, которая получит вычисленное значение выражения (правила записи и вычисления выражений представлены в следующем разделе). Например,

```
int I=5, K;
float X, Y=2.3;
double Z = 1.0e-2;
const float pi = 3.1415926535;
X = tan(pi/3); //X получит значение 1.732
```

Типы переменной и выражения могут не совпадать. В этом случае перед присваиванием значение выражения автоматически преобразуется к типу переменной. Если переменная имеет тип целый, а выражение – вещественный, то перед преобразованием типа дробная часть отбрасывается, например,

```
K = -tan(pi/3); //K получит значение -1
```

Есть и *сокращенные формы операторов присваивания*. Они бывают двух видов: одноместные и двуместные.

Знаками *одноместных операторов присваивания* являются составные знаки ++ (рядом два знака +) и -- (рядом два знака -), обозначающие соответственно увеличение на 1 (инкрементацию) и уменьшение на 1 (декрементацию) переменной, рядом с которой они расположены. Эти знаки могут располагаться либо перед, либо после переменной, которую требуется увеличить или уменьшить на 1, и это не имеет значения, если такой оператор не является частью выражения, где следует учитывать приоритеты операций. Например,

`I--;` // то же, что и оператор `I = I-1;`, и оператор `--I;`  
`++I;` // то же, что и оператор `I = I+1;`, и оператор `I++;`

Знаками *двуместных операторов присваивания* являются составные знаки, состоящие из знака операции (для числовых данных это знаки + - \* / %) и знака =. Так, если X - переменная, которой присваивается новое значение, а W выражение, то оператор

`X += W;` // эквивалентен оператору `X = X + W,`  
`X -= W;` // эквивалентен оператору `X = X - W,`  
`X *= W;` // эквивалентен оператору `X = X * W,`  
`X /= W;` // эквивалентен оператору `X = X / W,`  
`X %= W;` // эквивалентен оператору `X = X % W.`

## Арифметические выражения

*Арифметическое выражение* записывается в виде последовательности целых и/или вещественных констант, переменных и обращений к функциям, разделенных знаками арифметических операций и круглыми скобками. Обращения к функциям и вычисления выражений в скобках выполняются до операций над их результатами.

В выражения могут также входить операторы присваивания в различных формах, так что вычисление выражения будет приводить и к изменению значений входящих в эти операторы переменных.

Результат вычисления выражения будет целого типа, если на завершающих шагах операторы будут выполняться над целыми константами, переменными и функциями целого типа, иначе результат будет вещественного типа. Например, результатом вычисления выражения

$$6 + \tan(3 * 3.14 / 4)$$

будет вещественное число 4.9976, так как в последней операции (+) второй операнд - вещественное число -1.0024 (значение тангенса угла, примерно равного  $45^0$ ),

а результатом вычисления выражения

$$6 + (i = \tan(3 * 3.14 / 4))$$

будет целое число 5, так как в последней операции (+) вторым операндом будет переменная i целого типа, имеющая значение -1, полученное из вещественного -1.0024 (значения  $\tan(3 * 3.14 / 4)$ ) преобразованием к целому при присваивании переменной i.

В выражениях преобразование операнда целого типа к вещественному выполняется автоматически (см. предыдущие примеры), если другой операнд вещественный, но если требуется преобразовать вещественное в целое не используя присваивания, необхо-

димо применить *явное преобразование типа*: непосредственно перед операндом в круглых скобках записать имя целого типа. Например,

```
6+(int)tan(3*3.14/4)
```

Явное преобразование типа иногда приходится применять и для перехода от целого к вещественному. Например, для получения вещественного значения типа `float` натурального основания (e) в степени, вычисляемой выражением целого типа, например, `i%3`, следует использовать вызов функции `exp((float)i%3)`.

Последовательность выполнения операторов в выражении определяется их приоритетами и порядком их следования, а при необходимости изменения порядка вычислений, как уже упоминалось, используют круглые скобки. Сейчас рассмотрим использование в выражениях только арифметических операций.

В выражениях в первую очередь вычисляются обращения к функциям и содержимое круглых скобок, затем – унарные операции изменения знака (-), затем - операции типа умножения (\*, /, %) в порядке слева направо, затем – операции типа сложения (+ и -) в порядке слева направо. Если двуместная операция выполняется над вещественными данными разных типов (`float` и `double`), то результат будет иметь тип, представляющий больший диапазон значений (`double`).

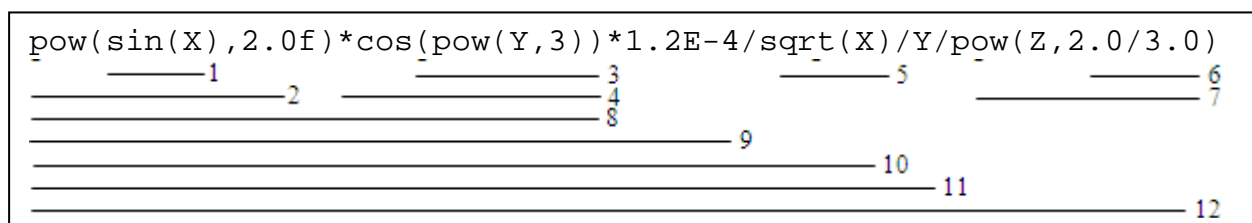
Например, для вычисления выражения

$$\frac{\sin^2(X) \cdot \cos(Y^3) \cdot 12 \cdot 10^{-5}}{\sqrt{X} \cdot Y \cdot Z^{2/3}}$$

в программе можно записать

```
pow(sin(X), 2.0f) * cos(pow(Y, 3)) * 1.2E-4 / sqrt(X) / Y / pow(Z, 2.0/3.0)
```

Порядок вычислений поясняет следующий рисунок



где линиями подчеркнуты части выражения, соответствующие отдельным операциям, а номера справа от линий указывают последовательность соответствующих действий.

Результатом вычисления выражения будет значение типа `double`, так как вещественные константы без суффикса, а также ранее объявленная переменная `Z` (см. выше) имеют тип `double`.

В C выражения могут включать операторы присваивания в различных формах.

Как и унарный минус (-), унарные операторы *инкрементации* (++) и *декрементации* (--) имеют наивысший приоритет. Однако если операторы ++ и -- меняют значение переменной, для которой они используются, то унарный минус не меняет значения переменной, а просто использует её значение со знаком минус. Операторы ++ и -- могут быть как *префиксными* (знак операции записывается перед переменной), так и *постфиксными* (знак операции записывается после переменной). От этого зависит, будет ли соответственно или новое, или исходное значение переменной использоваться для вычисления выражения. Например, если  $j$  равно 2 и целое  $k$  равно -3, то после выполнения вычислений

$$i = 2 * k -- ++ j$$

значением переменной  $i$  будет -9,  $k$  станет равным -4, а  $j$  – равным 3. Действия будут выполнены в следующем порядке: вначале будет выполнено ++  $j$  и  $j$  получит значение 3, затем, так как унарные операции выполняются в порядке справа налево, унарный минус сохранит значение  $-j$ , то есть -3, в некоторой дополнительной переменной, затем по части выражения  $2 * k$  при исходном значении  $k$  будет вычислено значение -6 и прибавлено к -3 (значению  $-j$ ), и только потом  $k$  будет уменьшено на 1 и станет равным -4. Эти же результаты можно получить, выполнив следующие три оператора присваивания:

$$j = j + 1; \quad i = 2 * k - j; \quad k = k - 1;$$

Последний способ описания процесса вычисления значений переменных  $i$ ,  $k$  и  $j$  обладает большей наглядностью и его следует использовать, если это возможно, чтобы уменьшить вероятность появления ошибки.

Если в рассмотренном выражении сделать декрементацию  $k$  префиксной:

$$i = 2 * --k ++ j;$$

то  $i$  получит значение -11, так как до умножения на 2 переменная  $k$  будет уже иметь значение -4.

Двуместные формы оператора присваивания имеют одинаковый низший приоритет и выполняются в порядке справа налево. После выполнения каждого из них переменная получает новое значение, которое и используется для продолжения вычисления выражения. Так, одним выражением можно задать последовательность вычислений значений нескольких переменных, причем значением всего выражения будет значение, присвоенное последней, самой левой переменной.

Например, площадь  $S$  основания и объём  $V$  цилиндра радиуса  $R$  и высотой  $L$  можно вычислить одним выражением

$$V = L * (S = 3.1415926535 * R * R)$$

Одним выражением можно присвоить одно и то же значение нескольким переменным, например, после вычисления выражения

```
X=Y=Z=3.5
```

переменные `X`, `Y` и `Z` будут иметь значение 3,5. Такая форма уменьшает текст программы и делает её более наглядной.

## Вывод десятичных чисел в окно программы

В C для форматированного вывода данных в окно программы используется стандартная функция `printf`. Эта функция позволяет выводить данные различных типов (числовых, символьных, строковых), однако сейчас ограничимся рассмотрением вывода только целых и вещественных десятичных чисел, а также текстов, размещаемых непосредственно в функции `printf`, и некоторых символов, управляющих размещением данных в строках окна программы.

Первым в списке параметров функции `printf` является *управляющая строка*, а далее – список вывода, в котором через запятую записываются выражения (в частности переменные, константы), значения которых должны быть выведены. Список вывода может отсутствовать.

Основными элементами управляющей строки являются форматы вывода, которые состоят из одной или пары букв после знака `%`. Для десятичных данных есть следующие форматы вывода:

`%d` – для целых со знаком,

`%u` – для целых без знака,

`%f` – для вещественных типа `float` в естественной форме,

`%lf` – для вещественных типа `double` в естественной форме,

`%e` – для вещественных в экспоненциальной форме,

`%g` – для вещественных в наиболее компактной форме: экспоненциальной или в естественной форме.

Форматы, записанные в управляющей строке, ставятся в соответствие элементам списка вывода порядком их следования: первый по порядку формат соответствует первому элементу списка вывода, второй – второму элементу и т.д.

Например, если в программе есть переменные

```
int k=-25;
```

```
unsigned int j=7;
```

```
float x=-4.55;
```



```
double y=3.125E-2;
```

то оператором

```
printf( "%d%u%f%lf" ,k, j, x, y) ;
```

будет выведена следующая строка чисел

```
-257-4.5500000.031250
```

В использованной управляющей строке `"%d%u%f%lf"` форматы расположены слитно, поэтому числа в строке окна программы также расположены слитно, и прочитать правильно данные не всегда будет возможно (в примере число 7, представляющее значение j, слилось с числом -25, представляющим значение k, и не понять, то ли это -2 и 57, то ли -25 и 7; вещественные числа также оказались выведенными слитно).

Для разделения чисел в окне программы в управляющую строку между форматами следует вставлять либо тексты, поясняющие выводимые данные, либо простые печатные символы (пробелы, подчеркивания и другие) в качестве разделителей, либо управляющие символы (табуляции, перехода на новую строку), либо их сочетания. Далее представлены примеры их использования.

#### 1. Использование поясняющих текстов:

оператор

```
printf( "k=%d j=%u x=%f y=%lf" ,k, j, x, y) ;
```

выведет следующую строку в окно программы

```
k=-25 j=7 x=-4.550000 y=0.031250
```

#### 2. Использование пробелов:

оператор

```
printf( "k=%d %u %f %lf" ,k, j, x, y) ;
```

выведет следующую строку в окно программы

```
-25 7 -4.550000 0.031250
```

#### 3. Использование управляющих знаков табуляции (\t) :

оператор

```
printf( "\n\t%d\t%u\t%f\t%lf\n" ,k, j, x, y) ;
```

выведет следующую строку в окно программы

```
-25      7      -4.550000      0.031250
```

#### 4. Использование управляющих знаков (\n) перехода на новую строку:

оператор

```
printf( "%d\n%u\n%f\n%lf\n" ,k, j, x, y) ;
```

выведет следующие строки в окно программы, в том числе пустую строку после чисел



```

scanf("%d",&Fi);
//Вывод значения угла в градусах и пропуск строки
printf("Fi = %d\n",Fi);
//Перевод угла в радианы и присвоение переменной R
R=Fi*atan(1.0)/45;
//Вывод значения R (угла в радианах) и пропуск строки
printf("\nR = %f rad\n",R);//
//Вывод tg(R) с поясняющим текстом
printf("\ntg(%lf) = %lf\n",R, tan(R));
return 0;
}

```

Протокол ввода-вывода в окне программы для угла  $30^\circ$  будет иметь вид:

```
input Fi: 30
```

```
Fi = 30
```

```
R = 0.523599 rad
```

```
tg(0.523599) = 0.577350
```

В этом примере для всех типов данных использовались принятые по умолчанию (что не всегда удобно) минимальные поля вывода (количество позиций в строке окна программы, отводимых для изображения выводимого числа), а дробные части вещественных (последовательность цифр, расположенных после точки) округляются до шести цифр. Целые и строковые занимают минимально необходимое число позиций, причем целые положительные изображаются без знака +.

Программисту, чтобы определить для вещественных чисел свои поля вывода и количества цифр в дробной части (после точки), следует перед символом формата записать эти значения через точку. Если поле вывода будет иметь длину, большую чем требуется для представления выводимого значения, то это значение будет выведено в правой части поля (с выравниванием по правой стороне поля). Чтобы выравнивание было по левой стороне поля вывода, перед значением длины поля следует поставить знак минус (-).

Например, если в программе вместо оператора

```
printf("\nR = %f rad\n",R);
```

длина поля вывода по умолчанию

использовать

```
printf( "\nR = %30.18f rad \n", R );
```

то будет выведено в поле из 30 знакомест в экспоненциальной форме число с 18-ю цифрами после точки в мантиссе с выравниванием по правой стороне поля вывод

```
R = 0.523598790168762210 rad
```

а если использовать выравнивание влево (знак минус в формате вывода)

```
printf( "\nR = % -30.18f rad \n", R );
```

то будет выведено

```
R = 0.523598790168762210 rad
```

Аналогично, если в программе вместо оператора

```
printf( "\nR = %e rad \n", R ); //длина поля вывода по умолчанию
```

выводящего

```
R = 5.235988e-001 rad
```

использовать

```
printf( "\nR = %25.18e rad \n", R );
```

то будет выведено в поле из 25 знакомест в естественной форме число с 18-ю цифрами в дробной части с выравниванием по правой стороне поля вывод

```
R = 5.235987901687622100e-001 rad
```

а если использовать выравнивание влево (знак минус в формате вывода)

```
printf( "\nR = % -25.18e rad \n", R );
```

то будет выведено

```
R = 5.235987901687622100e-001 rad
```

## Ввод десятичных чисел с клавиатуры

В C для форматированного ввода данных с клавиатуры используется стандартная функция `scanf`. Эта функция позволяет вводить данные различных типов (числовых, символьных, строковых), однако сейчас ограничимся рассмотрением ввода только целых и вещественных десятичных чисел.

Первым параметром функции `scanf` является управляющая строка, а следующими - *указатели* на вводимые переменные, то есть имена переменных с предшествующим знаком `&`. В управляющей строке указываются форматы ввода одной или парой букв после знака `%`:

`%d` – для целых со знаком,

`%u` – для целых без знака,

`%f` – для вещественных типа `float`,

`%lf` – для вещественных типа `double`.

Например, если в программе есть переменные

```
int k;
unsigned int j;
float x;
double y;
```

и требуется ввести их значения, то можно использовать оператор

```
scanf( "%d %u %f %lf" , &k , &j , &x , &y ) ;
```

Вводимые числа можно разделять пробелами или знаками табуляции, вводить одной строкой или несколькими строками (нажимая для перехода на очередную строку клавишу Enter), например, так

```
-25  77  5.71  1e-5
```

или так

```
-25 77
```

```
5.71
```

```
1e-5
```

В любом случае будет ожидание ввода значений всех переменных, представленных параметрами функции `scanf`, а если набрано больше чисел, чем переменных в списке параметров, то оставшиеся не прочитанными будут сохранены для очередного ввода. Так подготовленные для ввода одной строкой числа

```
-25  77  5.71  1e-5
```

можно ввести несколькими, например двумя вызовами функции `scanf`, даже если они разделены другими операторами

```
scanf( "%d %u" , &k , &j ) ;
```

.....

```
scanf( "%f %lf" , &x , &y ) ;
```

В случае ошибки в наборе вводимых данных (появлении во входном потоке при вводе по числовым форматам не чисел и недопустимых разделителей между числами) работа программы не будет прервана. Это может оказаться причиной неверного результата работы программы, так как переменные из списка ввода функции `scanf`, не получившие значений из входного потока, сохранят свои старые значения. Для обработки подобных ошибок можно использовать различные приёмы, например,

1. вывод введённых чисел для визуального контроля:

```
scanf ("%d %u %f %lf" , &k , &j , &x , &y ) ;
printf ("%d \t%u \t%f \t%g" , k , j , x , y ) ;
```

2. сравнение в программе с помощью операторов `if` или `while` (см. далее раздел «Программы разветвляющейся структуры» и «Программы циклической структуры») количества переменных в списке ввода функции `scanf`, с количеством фактически прочитанных из входного потока чисел, возвращаемым функцией (если оно 0, значит ошибка возникла до ввода первого числа, если 1, значит было прочитано только одно число и т.д.) и при несовпадении указанных количеств организовать либо повторный ввод, либо вывод соответствующего сообщения и выход из программы, либо какую-то другую обработку.

## 2.2 Приемы, используемые для минимизации вычислений

Одним из критериев, характеризующих качество составленной программы, является объем выполняемых ею вычислений для достижения требуемого результата. Чем он меньше, тем, как правило, быстрее будет работать программа. Существуют разные приемы, применение которых позволяет сократить объем вычислений за счет уменьшения в первую очередь количества вызовов функций, затем – количества операций типа умножения, и, наконец, – количества операций типа сложения. Вот некоторые из них, рассмотренные отдельно, хотя, как правило, они используются в сочетании.

Вынесение общих множителей за скобки. Например, вместо оператора

```
Z=sin(X)*Y-sin(X)*sqrt(Y)*Y+sqrt(X)*X+X;
```

лучше использовать оператор

```
Z=sin(X)*Y*(1-sqrt(Y))+X*(sqrt(X)+1);
```

Использование схемы Горнера для полиномов. Например, полином

$$2X^5 - 5X^4 + 2X^3 + 7X^2 - 4X + 6$$

преобразованный по схеме Горнера, примет вид

$$(((2X-5)X+2)X+7)X-4)X+6$$

где явно меньше число операций умножения, если считать, что возведение в степень выполняется через умножение. В программе по второму варианту записи полинома будем иметь выражение

$$(((2 * X - 5) * X + 2) * X + 7) * X - 4) * X + 6 ,$$

а по первому –

$$2 * \text{pow}(X, 5) - 5 * \text{pow}(X, 4) + 2 * \text{pow}(X, 3) + 7 * \text{pow}(X, 2) - 4 * X + 6$$

где помимо такого же числа операций умножения требуется выполнить четыре обращения к функции возведения в степень.

Другой пример, где также применима схема Горнера:

$$1+2!+3!+4!+5! = 1+2(1+3(1+4(1+5)))$$

Использование дополнительных переменных. Например, при вычислении значения функции  $Z = (X - Y) \frac{1 + (X - Y)^3}{1 + (X - Y)^2}$  целесообразно предварительно вычислить

$A = X - Y$  и  $B = A^2$ , а исходную формулу преобразовать к виду  $Z = A \frac{1 + AB}{1 + B}$ . В этом

случае в программе будут использованы три оператора присваивания:

```
A=X-Y;
```

```
B=A*A;
```

```
Z=A*(1+A*B)/(1+B);
```

## 2.3 Примеры выполнения задания

### Пример 2.1 выполнения задания

Найти коэффициенты  $k_0, k_1, k_2, k_3$  представления числа  $X$  ( $0 \leq X \leq 80$ ) в троичной системе счисления:

$$X = k_3 \cdot 3^3 + k_2 \cdot 3^2 + k_1 \cdot 3 + k_0 \quad (2.1)$$

используя операции получения частного целочисленного деления (/) и остатка (%). Для контроля результатов выполнить вычисление  $X$  непосредственно по формуле (2.1) для найденных коэффициентов, а также после преобразования выражения в формуле (2.1) по схеме Горнера. Вывести все результаты вычислений в наглядной форме с поясняющими текстами. Проверить работу программы на значениях

$X=(0; 1; 2; 10; 27; 48; 80)$ .

Программа

```
#include <stdafx.h>
int _tmain(int argc, _TCHAR* argv[])
{
    unsigned int X, k0, k1, k2, k3, X1, X2;
    //Ввод исходных данных
    printf("input X: ");
    scanf("%u", &X);
```

```

//Вычисление коэффициентов разложения
    k0=X%3; X=X/3; //k0:=остаток от деления X на 3;
    k1=X%3; X=X/3; //k1:=остаток от деления X/3 на 3;
    k2=X%3; //k2:=остаток от деления X/9 на 3;
    k3=X/3; //k3:=X/27;

//Вывод вычисленных коэффициентов разложения
//числа X по степеням 3 с поясняющими текстами
    printf("\nk3=%u    k2=%u    k1=%u
k0=%u", k3, k2, k1, k0);
//    Проверки результатов вычислений
// 1.
//Вычисление непосредственно по формуле (2.1)
    X1=k3*27+k2*9+k1*3+k0;
//и вывод результата
    printf("\nX1 = %u", X1);
// 2.
//Вычисление по формуле (2.1),
//преобразованной по схеме Горнера
    X2=((k3*3+k2)*3+k1)*3+k0;
//и вывод результата
    printf("\nX2 = %u\n", X2);
    return 0;
}

```

Протокол ввода-вывода этой программы для числа 77 будет следующим:

```

input X: 77
k3=2    k2=2    k1=1    k0=2
X1 = 77
X2 = 77

```

## Пример 2.2 выполнения задания

Найти значение функции

$$Y(X) = \frac{\left(\frac{a}{2}\right)^x - \lg\left(\frac{a}{2} + 1\right)}{\left(\frac{a}{2}\right)^3 - \left(\frac{a}{2}\right)^2} \quad (2.2)$$



упростив вычисления за счет использования скобочных форм и/или дополнительных переменных (в этом предложении и в дальнейшем конструкция «А и/или Б» обозначает «или А, или Б, или А и Б одновременно»). Для контроля правильности результата выполнить вычисление по формуле (2.2) без использования скобочных форм и дополнительных переменных.

Протестировать программу на значениях  $X=(0,5; 2)$  и  $A=(1; -1; 2; -2; 4; -4)$ . Обратить внимание на вывод значений выражений при значениях аргументов, не принадлежащих областям определения входящих в выражения функций.

Программа

```
#include "stdafx.h"
#include "math.h"
int _tmain(int argc, _TCHAR* argv[])
{
    double A=2.0, X=0.5, B, C, Y1, Y2;
    //Ввод исходных данных
    printf("input X: ");
    scanf("%lf",&X);
    printf("input A: ");
    scanf("%lf",&A);
    //Вычисление выражения
    // - с использованием дополнительных переменных
    B=A/2;
    C=B*B;
    // - непосредственно по формуле (2.2)
    Y2=(pow(A/2,X)-log10(A/2+1)) / (pow(A/2,3)
        -pow(A/2,2));
    //Вывод вычисленных значений с надписями
    printf("\n\t\tY1\t\tY2");
    printf("\n\t\t%e\t\t%e\n",Y1,Y2);
    return 0;
}
```

Исходные данные для тестирования программы подобраны так, чтобы проверить:

1) правильность вычислений на допустимых значениях функций вычисления десятичного логарифма (выполняемого в программе библиотечной функцией  $\log_{10}$ ) и воз-

ведения вещественного числа в степень (выполняемого в программе библиотечной функцией `pow`), и

2) реакцию программы на недопустимые значения аргументов функций (вычисление логарифма при отрицательном аргументе и возведение отрицательного вещественного числа в дробную степень).

Ниже представлены соответствующие примеры результатов работы программ

1)

input X: 2

input A: 1

Y1	Y2
-5.912699e-001	-5.912699e-001

2)

input X: 0.5

input A: -1

Y1	Y2
-1.#IND00e+000	-1.#IND00e+000

В примере 2) выведенные значения `-1.#IND00e+000` не являются числами. Такие и подобные им тексты указывают на ситуации, когда численное решение получить невозможно. Это не только использование недопустимых значений аргументов функций, но и невозможность представления результата арифметической операции, например, при делении на ноль. Для повышения информативности программы желательно не допускать возникновения подобных ситуаций. Простейшим способом является проверка значений операндов и в случае их недопустимости, вывод соответствующего сообщения и завершение работы программы. В представленных далее заданиях обработка подобных ситуаций не предполагается, но при их возникновении необходимо найти и объяснить причину.

## **2.4. Задания А для самостоятельной работы**

В заданиях 1 – 28:

\* вычислить, упростив за счет использования скобочных форм и/или дополнительных переменных, значения по заданным формулам,

\* для контроля правильности результатов выполнить вычисления по формулам без использования скобочных форм и дополнительных переменных,

\* проверить результаты на комбинациях заданных значений.

1.  $Z = X^2 \cdot Y^2 + 3 \cdot X \cdot Y^2 - 5 \cdot X^2 \cdot Y + X^2 - 2 \cdot Y^2 + 4 \cdot X \cdot Y - X + Y$ ,  $X=(2; -2)$ ,  $Y=(4; -3)$ .

2.  $B=A+2$ ;  $C=(A+3)/(A+2)$ ;  $D=(A+4)/(A+3)$ ;  $E=(A+5)/(A+4)$ ;  $A=(1; 2; -2; 3; 4)$ .

3.  $Z = (X + 2) \frac{(X + 2)^2 + 3}{(X + 2)^4 + (X + 2)^2 + 3}$ ;  $X=(0; 1; 2; -2; 4)$ .

4.  $B=\sin A$ ;  $C=\lg A$ ;  $D=e^A$ ;  $E=|A|$ ;  
 $S=(A+B) \cdot (A+B+C) \cdot (A+B+C+D) \cdot (A+B+C+D+E)$ ;  $A=(8; -2; 4; -5)$ .

5.  $B=A+5$ ;  $C=A-2$ ;  $D=B+C$ ;  $E=A-C$ ;  
 $P1 = \frac{A}{B}$ ;  $P2 = \frac{A \cdot C}{B}$ ;  $P3 = \frac{A \cdot C}{B \cdot D}$ ;  $P4 = \frac{A \cdot C \cdot E}{B \cdot D}$ ;  $A=(-15; -5; 0; 7; 14)$ .

6.  $B=A-2$ ;  $C=A+3$ ;  $D=B+C$ ;  $E=A-2$ ;  
 $P1=A \cdot B$ ;  $P2=A \cdot B \cdot C$ ;  $P3=A \cdot B \cdot C \cdot D$ ;  $P4=A \cdot B \cdot C \cdot D \cdot E$ ;  $A=(-4; 0; 4; 7)$ .

7.  $Y = \frac{3}{1+3+X} - \frac{3 \cdot 5}{1+3+5+2 \cdot X} + \frac{3 \cdot 5 \cdot 7}{1+3+5+7+3 \cdot X} - \frac{3 \cdot 5 \cdot 7 \cdot 9}{1+3+5+7+9+4 \cdot X}$ ,  
 $X=(-9; -4; 0; 3; 9)$ .

8.  $Y = -X^6 + A \cdot X^5 - A^2 \cdot X^4 + A^3 \cdot X^3 - A^4 \cdot X^2 + A^5 \cdot X - A^6$ ,  $X=(-3; 5)$ ,  $A=(-3; 5)$ .

9.  $Y = A^X - \frac{A^{2X}}{3} + \frac{A^{3X} \cdot 4}{3 \cdot 5} - \frac{A^{4X} \cdot 6}{3 \cdot 5 \cdot 7}$ ,  $X=(-3; 0; 3)$ ,  $A=4$ .

10.  $Y = X + \frac{2 \cdot 4 \cdot X^2}{2+4} + \frac{2 \cdot 4 \cdot 6 \cdot X^3}{2+4+6} + \frac{2 \cdot 4 \cdot 6 \cdot 8 \cdot X^4}{2+4+6+8} + \frac{2 \cdot 4 \cdot 6 \cdot 8 \cdot 10 \cdot X^5}{2+4+6+8+10}$ ,  
 $X=(-7; -2; 0; 2; 7)$ .

11.  $Y = \left( \frac{\sqrt[3]{A-B} \cdot (A+B)^{2/3} \cdot X^2}{A^4 + B^4 - 2A^2B^2 - X^4} \right)^{\frac{1}{X}}$ ,  $X=(0,5; 1; 2)$ ,  $A=4$ ,  $B=3$ .

12.  $Y = \left(\frac{A}{B}\right)^X + \left(\frac{A}{B}\right)^{2X} + 2\left(\frac{A}{B}\right)^{-2/3} \cdot \log_2\left(\frac{A}{B}\right)$ ,  $X=(2,5; 5; 7; 10)$ ,  $A=4$ ,  $B=3$ .

13.  $Y = \frac{-3,3 \cdot 10^{-4} \operatorname{tg} X \cdot \lg(X^2 - 5) \sqrt{|\operatorname{tg} X|}}{\sqrt[3]{X^2 - 5} \cdot X \cdot e^{-2X}}$ ,  $X=(1; 2,5; 5; 7; 10)$ .

14.  $Y = \frac{\sqrt[3]{(X+1)^2} \cdot (X+2) \cdot \ln(X+1)}{(X+2)^3 - (X+2)^2 + (X+1)^2}$ ,  $X=(-0,5; 5; 10; 25)$ .

15.  $Y = \frac{\sqrt[3]{X^4 + X^3 - X^2 - X + 1}}{X^4 + 2X^3 - 2X - 1}$ ,  $X=(-15; -5; -2; 2; 5)$ .

16.  $Y = \frac{X + X^3 - 3}{2X^2 - 4X + 2 + 6X^3 - 4X^5}$ ,  $X=(-5; -2; 2; 5)$ .

$$17. Y = \frac{\cos^4 X}{4} - \frac{\cos^2 X}{2} - \ln(|\cos X|), \quad X=(0; 30^\circ; 45^\circ; 60^\circ; 90^\circ).$$

$$18. Y = \left(\frac{1}{2} + (X-1) - \frac{(X-1)^2}{2} + \frac{(X-1)^4}{3}\right) / (X^2 - 1), \quad X=(0; 1; 2; 3).$$

$$19. Y = \frac{\left(\frac{1}{2} + \sin^2 X\right) \ln |\sin X|}{\frac{\pi}{3} - \arcsin X}, \quad X=(0,001; 0,1; 0,3; 0,5; 0,9; 1,8).$$

$$20. Y = \frac{e^X - e^{2X} + e^{3X} \arccos^2 X}{e^X - e^{2X} + e^{3X} \arcsin^{0,5} X}, \quad X=(0,001; 0,02; 0,1; 0,9).$$

$$21. Y = \frac{\left|\frac{1}{4} - A^2 \operatorname{tg} X\right|}{(\sin X + \cos X)(\operatorname{tg}^2 X + 1)}, \quad X=(0,001^\circ; 15^\circ; 30^\circ; 60^\circ; 135^\circ).$$

$$22. Y = \frac{|\sin X \cdot \cos X| + \operatorname{tg} X}{\sin X + \sin X \cdot \cos X + \cos X}, \quad X=(0,001^\circ; 15^\circ; 30^\circ; 60^\circ; 270^\circ).$$

$$23. Y = \frac{2X^3 + 6X^2 - 8X + 4}{-4X^3 + 8X^2 - X^5 + 2X^4}, \quad X=(-5; -2; 2; 5).$$

$$24. Y = \frac{1 - \sqrt{|\log_2 X| + 25 \cdot 10^{-5} \cdot \log_{10} X}}{\log_2 X + 0,00025 \cdot \log_{10} X}, \quad X=(0,001; 0,1; -1; 1; 4).$$

$$25. Y = \frac{X \lg(X+1) + \lg(X+1) + X \ln A + \ln A + A^{X+1} \lg(X+1) + A^{X+1} \ln A}{\ln A + \ln(X+1)}, \quad X=(0,001; 0,1; -1; 1; 4), A=3.$$

$$26. Y = \frac{A^X X^A + 2A^{2X} X^A - 2A^X X^{2A} - 4A^{2X} X^{2A}}{\lg A + \lg X}, \quad X=(0,001; 0,1; 1; 4), A=1,5.$$

$$27. Y = \frac{A^{X^A} X^{A^{A \cdot X}}}{A^{A \cdot X} + X^{A \cdot X}}, \quad X=(0,001; 0,1; 1; 4) \text{ и } A=2.$$

$$28. Y = 1 + \frac{X^2}{3^2} + \frac{X^2}{7^2} + \frac{X^2}{11^2} + \frac{X^4}{3^2 7^2} + \frac{X^4}{3^2 11^2} + \frac{X^4}{7^2 11^2} + \frac{X^6}{3^2 7^2 11^2}, \quad X=(-4; 0; 4; 11).$$

В заданиях 29 и 30 найти коэффициенты  $k_0, k_1, k_2, \dots$  представления числа  $X$  ( $0 \leq X < P^N$ ) в позиционной системе счисления с основанием  $P$ , используя операции  $/$  и  $\%$ . Для контроля результатов выполнить вычисление  $X$  непосредственно по заданной формуле разложения  $X$  по степеням  $P$  для найденных коэффициентов, а также после преобразования выражения в формуле по схеме Горнера. Вывести все результаты вычислений в наглядной форме с поясняющими текстами. Проверить работу программы при вводимых значениях  $X$  из набора  $M$ .

**29.**  $P=8, N=4, X = k_4 \cdot 8^4 + k_3 \cdot 8^3 + k_2 \cdot 8^2 + k_1 \cdot 8 + k_0, M=\{0; 1; 2; 4; 7, 8; 65; 1023; 4095\}$ .

**30.**  $P=16, N=3, X = k_3 \cdot 16^3 + k_2 \cdot 16^2 + k_1 \cdot 16 + k_0, M=\{0; 1; 15; 64; 127; 255; 2047; 4095\}$ .

**31.** Найти среднее геометрическое абсолютных значений частных от целочисленного деления  $X, X_2, X_3$  на  $Y$  и среднее арифметическое остатков от целочисленного деления  $X, X_2, X_3$  на  $Y$ . Для контроля результатов целочисленного деления выводить на экран с поясняющими надписями делимое, делитель, частное, абсолютное значение частного, остаток. Также с поясняющими текстами вывести найденные средние геометрические и средние арифметические. Проверить работу программы при вводе значений  $X=(-5; 5)$  и  $Y=(-3; 3)$ .

**32.** Координаты вершин параллелепипеда заданы положительными значениями  $X_1, X_2, Y_1, Y_2, Z_1, Z_2$  ( $X_1 < X_2, Y_1 < Y_2, Z_1 < Z_2$ ), имеющими ненулевые дробные части. Требуется найти целочисленные координаты  $I_1, I_2, J_1, J_2, K_1, K_2$  вершин такого параллелепипеда, который находится внутри заданного и имеет наибольший объем. Найти также объемы этих параллелепипедов и отношение объемов. Все значения  $X_1, X_2, Y_1, Y_2, Z_1, Z_2$  и  $I_1, I_2, J_1, J_2, K_1, K_2$  вывести на экран с поясняющими надписями, а найденные объемы и их отношения вывести с предшествующими поясняющими текстами. Проверить работу программы на вводимых  $X_1=(2,7; 5,2), X_2=2 \cdot X_1, Y_1=X_1-1, Y_2=2 \cdot Y_1, Z_1=\frac{X_1}{2}, Z_2=3 \cdot Z_1$ .

## **2.5. Задания Б для самостоятельной работы**

Во всех заданиях:

- \* вводимые и выводимые данные сопровождать краткими поясняющими текстами,
- \* для проверки численных значений результатов предусмотреть в программе соответствующие вычисления,

**1.** Решить систему из двух линейных уравнений и проверить найденное решение подстановкой результатов в уравнения.

**2.** Вычислить площадь  $S$  остроугольного треугольника, заданного координатами вершин на плоскости, по формуле Герона, а затем – величины углов, используя со-

отношение  $S = \frac{La \cdot Lb \cdot \sin C}{2}$ , где  $C$  – угол между сторонами с длинами  $La$  и  $Lb$ , а также,

для проверки результатов, вычислить сумму углов.

**3.** Вычислить координаты точек на плоскости, делящих отрезок прямой, заданный координатами концов, в отношении  $m:n:k$ . Выполнить поверку работы программы вычислением  $m$ ,  $n$  и  $k$  на основе полученных координат точек деления отрезка прямой.

**4.** Решить квадратное уравнение считая, что оно имеет только вещественные корни. Проверить результаты подстановкой корней в уравнение.

**5.** Вычислить коэффициенты уравнения прямой  $Y=K \cdot X+B$ , проходящей через точки с координатами  $(X1, Y1)$  и  $(X2, Y2)$ , и найти точку пересечения этой прямой с осью абсцисс. Проверить результаты подстановкой в уравнение для заданных координат точек.

**6.** Вычислить координаты точки пересечения двух прямых, заданных уравнениями:  $A \cdot X+B \cdot Y=C$  и  $D \cdot X+E \cdot Y=F$ . Проверить результаты подстановкой в уравнения.

**7.** Точка имеет координаты  $X0, Y0$ . Вычислить координаты точки после поворота осей координат относительно начала на угол  $A$  против часовой стрелки. Проверить работу программы для:

а)  $A=\arctg(Y0/X0)$ , б)  $A=\pi$ , в)  $A=\arctg(Y0/X0)-\pi/2$ .

**8.** Вычислить координаты вершин треугольника, находящихся на пересечении прямых  $Y=k1 \cdot X+b1$  и  $Y=k2 \cdot X+b2$  между собой и с осью  $X$ . Найти площадь  $S$  этого треугольника, а также длины сторон. Проверить работу программы вводом данных для уравнений  $Y=X+1$  и  $Y=-X+1$ .

**9.** Вычислить координаты точек пересечения прямой и окружности на плоскости.

$A \cdot X+B \cdot Y=C$ ,  $X^2+Y^2=R^2$ . Проверить результаты подстановкой в уравнения.

**10.** Вычислить площадь  $S$  равнобедренного треугольника, вписанного в окружность радиуса  $R$ , если известна длина  $La$  его стороны, не равная длинам других сторон. Найти также длины других сторон треугольника и угол  $A$  между ними. Проверить работу программы на равностороннем треугольнике по его площади, которую следует вычислить заранее.

**11.** Вычислить координаты точки пересечения эллипса  $A \cdot X^2 + B \cdot Y^2 = R^2$  и гиперболы  $Y = C/X$ . Проверить результаты: при  $A=B=C=1$  и  $R^2=2$  должно быть  $X_1=X_3=1, Y_1=Y_3=1$   $X_2=X_4=-1, Y_2=Y_4=-1$ .

**12.** Найти числа  $X$  и  $Y$ , произведение которых равно  $A$ , а разность равна  $B$ . Вывести найденные значения, а также, для контроля, – их произведение и разность. Проверить работу программы также при  $A=1$  и  $B=0$ , где решение очевидно.

**13.** Вычислить площадь треугольника, заданного координатами вершин в пространстве, по формуле Герона. Подобрать два варианта исходных данных для проверки работы программы.

**14.** Найти числа  $X$  и  $Y$ , сумма которых равно  $A$ , а сумма квадратов равна  $B$ . Вывести найденные значения, а также, для контроля, – их сумму и сумму квадратов. Проверить работу программы также при вводе  $A=1$  и  $B=1$ , где решение очевидно.

**15.** Для треугольника, заданного длинами сторон  $L_a, L_b, L_c$ , найти угол, противоположный стороне длины  $L_a$ , используя соотношение  $\sin\left(\frac{\alpha}{2}\right) = \sqrt{\frac{(P - L_b) \cdot (P - L_c)}{L_b \cdot L_c}}$ , где  $P$  – полупериметр треугольника. Найти также другие углы. Проверить результаты для различных исходных данных по сумме углов.

**16.** Вычислить координаты точек пересечения кривых, заданных уравнениями  $Y = X + C$  и  $\left(\frac{X}{2}\right)^2 + Y^2 = 1$ . Проверить результаты подстановкой в исходные уравнения.

**17.** Вычислить площадь правильного  $N$ -угольника, вписанного в окружность радиуса  $R$ . Найти относительные ошибки замены площади круга площадью такого  $N$ -угольника при значениях  $N$ , равных 6, 60 и 360. Проверить правильность решения: при  $N=4$  и любом  $R$  относительная ошибка должна быть равна 0,363.

**18.** Вычислить площадь правильного  $N$ -угольника, в который вписана окружность диаметра  $D$ . Найти относительные ошибки замены площади такого  $N$ -угольника площадью круга при значениях  $N$ , равных 12, 120, 720. Проверить правильность решения: при  $N=4$  и любом  $D$  относительная ошибка должна быть равна 0,274.

**19.** Вычислить площадь равнобедренного треугольника, вписанного в окружность радиуса  $R$ , если известен угол  $A$  между его сторонами равной длины. Вычислить также отношение площади круга радиуса  $R$  к площади треугольника. Проверить работу программы при вычислении отношения площади круга радиуса  $R$  к площади треуголь-

ника при вводе следующих значений угла А:

а)  $\pi/3$ , когда площадь треугольника равна  $\frac{3\sqrt{3}R^2}{4}$ ,

б)  $\pi/2$ , когда площадь треугольника равна  $R^2$

**20.** Найти числа X и Y, сумма которых равно A, а разность равна B. Вывести найденные значения, а также, для контроля, – их сумму и разность. Проверить работу программы также при вводе A=1 и B=1, где решение очевидно.

**21.** Для треугольника, заданного длинами сторон  $L_a, L_b, L_c$ , найти угол  $\alpha$ , противоположный стороне длины  $L_a$ , используя соотношение  $\cos\left(\frac{\alpha}{2}\right) = \sqrt{\frac{P \cdot (P - L_a)}{L_b \cdot L_c}}$ , где P – полупериметр треугольника. Найти также другие углы. Проверить результаты для различных исходных данных по сумме углов.

**22.** На плоскости найти угол A между двумя сторонами (1, 2) и (1, 3) остроугольного треугольника, заданного координатами вершин  $X_1, Y_1, X_2, Y_2, X_3, Y_3$  ( $X_1 < X_2 < X_3$ ), длины  $L_{12}, L_{13}$  этих сторон и затем – площадь треугольника S по формуле  $S = L_{12} \cdot L_{13} \cdot \sin(A) / 2$ . Значение угла A вывести в градусах. Проверить работу программы при вводе  $X_1=0, Y_1=0, X_2=1, Y_2=0, X_3=2$  и  $Y_3=(2, -2)$ .

**23.** Вычислить площадь S равнобедренного треугольника, в который вписана окружность радиуса R, если известна длина  $L_a$  его стороны, не равная длинам других сторон. Найти также длину L других сторон треугольника и его углы. Проверить работу программы на равностороннем треугольнике по его площади, которую следует вычислить заранее.

**24.** Найти площадь прямоугольного треугольника, в который вписана окружность радиуса R, а также значения его углов, если известна длина  $L_a$  его катета  $K_a$ . Для проверки работы программы предусмотреть вычисление  $L_a$  по найденной длине  $L_b$  другого катета. Проверить работу программы также при  $R=1$  и  $L_a=2 + \sqrt{2}$ , когда прямоугольник будет равнобедренным.

**25.** Найти координаты центра тяжести треугольника на плоскости, то есть координаты точки, лежащей на медиане и отстоящей на  $2/3$  ее длины от вершины, из которой медиана проведена. Для проверки результата выполнить вычисления для всех трех медиан. Проверить работу программы также для равнобедренного прямоугольного треугольника с координатами вершин  $(0; 0), (3; 0), (0; 3)$ , где решение очевидно.



**26.** Вычислить  $S$  – площадь остроугольного треугольника по формуле  $S = \frac{La \cdot Lb \cdot \sin C}{2}$ , где  $La$  и  $Lb$  – длины сторон, а  $C$  – угол между ними. Затем вычислить длину третьей стороны  $Lc$ , используя соотношение  $Lc^2 = La^2 + Lb^2 - 2 \cdot La \cdot Lb \cdot \cos C$  и остальные углы, используя соотношение  $\sin A / \sin C = La / Lc$ . Проверить результаты для различных исходных данных по сумме углов.

**27.** Найти:

а) уравнение прямой  $Y = k_2 \cdot X + b_2$ , проходящей через точку  $(X_0, Y_0)$  и перпендикулярную заданной прямой  $Y = k_1 \cdot X + b_1$

б) точку  $(X_1, Y_1)$  пересечения этих прямых,

в) площадь и длины сторон треугольника, вершинами которого являются точки  $(X_1, Y_1)$ ,  $(X_0, Y_0)$  и точка  $(X_2, Y_2)$  пересечения оси  $Y$  с заданной прямой.

Проверить результаты, предварительно вычислив площадь треугольника с вершинами в этих точках при вводе  $k_1=1$ ,  $b_1=1$ ,  $X_0=0$ ,  $Y_0=2$ .

**28.** Дано уравнение

$$A \cdot X^2 + B \cdot Y^2 + C \cdot X + D \cdot Y + E = 0.$$

Вычислить коэффициенты уравнения

$$A_1 \cdot X^2 + B_1 \cdot Y^2 + C_1 \cdot X + D_1 \cdot Y + E_1 = 0,$$

получающиеся после переноса начала координат в точку  $X_1, Y_1$  и проверить результаты.

Выполнить также проверку решения обратным преобразованием координат.

## Вопросы по самопроверке

1. Какие операторы используются в программах линейной структуры?
2. Какие числовые типы данных рассматривались в разделе?
3. Какие есть способы записи вещественных констант?
4. Как обеспечить использование в программе библиотечных подпрограмм математических функций?
5. Как в окне редактора кода программы получить справку по допустимым типам параметров библиотечных подпрограмм и типам возвращаемых ими результатов?
6. Какие есть конструкции операторов присваивания?
7. Что может быть частью арифметического выражения (целого, вещественного)?
8. Каков порядок вычисления частей выражения?
9. Можно ли в выражении изменять значения переменных?
10. В чём назначение первого и остальных параметров оператора `printf`?
11. Какие есть форматы вывода десятичных чисел?
12. Из каких частей строится формат вывода десятичных чисел?
13. Что, кроме форматов вывода, может быть частью управляющей строки оператора `printf`?
14. Что может быть элементом списка вывода оператора `printf`?
15. В чём назначение первого и остальных параметров оператора `scanf`?
16. Что может быть элементом списка ввода оператора `scanf`?
17. Какие приёмы могут использоваться для минимизации объёма вычислений?

### 3. Программы разветвляющейся структуры

#### 3.1 Средства разработки программ разветвляющейся структуры

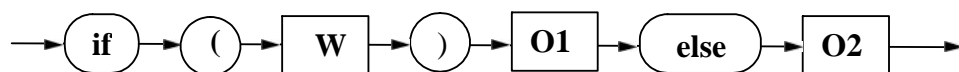
*Программой разветвляющейся структуры* называют такую программу, в которой, в зависимости от исходных данных, возможны различные последовательности выполнения операторов, причем на любой из них каждый оператор выполняется только один раз.

Для реализации программ или фрагментов программ с разветвляющейся структурой используются сложные операторы<sup>1</sup>: условные операторы `if`, операторы переключатели `switch` и тернарный оператор (условное выражение) `?`. В этом разделе ограничимся рассмотрением полной формы условного оператора - оператора `if else` и его сокращенной формы – оператора `if` без `else`.

При использовании условных операторов `if` ветвление алгоритма обусловлено проверками выражений различных типов, результаты вычисления которых рассматриваются либо как «истина» (числовое значение не ноль), либо как «ложь» (числовое значение ноль). В условных операторах логические выражения могут быть как простейшие, основанные на сравнении значений выражений, так и сложные, использующие логические операции.

#### Условные операторы

Оператор `if else` имеет следующую синтаксическую диаграмму



где

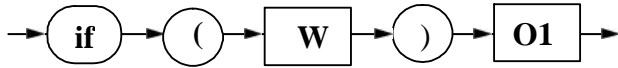
`W` – *выражение*, значением которого может быть либо «истина» (`true` или «не ноль»), либо «ложь» (`false` или 0).

`O1` и `O2` – операторы, заканчивающиеся знаком `;` (точка с запятой), или блоки. Каждый из операторов может быть пустым (просто знак `;`).

<sup>1</sup> Сложные операторы включают в себя другие операторы

При выполнении оператора `if else` вначале вычисляется выражение `W` и если результат – «истина», то выполняется оператор `O1`, иначе, то есть если результат имеет значение «ложь», выполняется оператор `O2`.

Оператор `if` без `else` имеет синтаксическую диаграмму



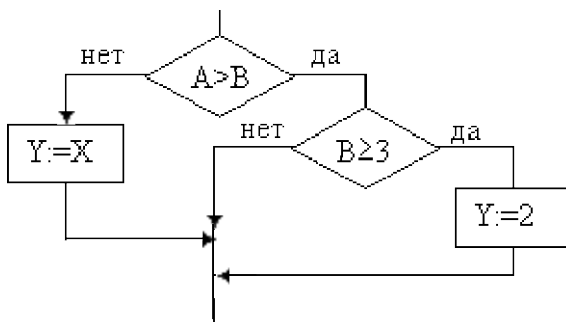
где `W` – выражение, `O1` – оператор, заканчивающийся знаком `;` (точка с запятой), или блок. При выполнении оператора `if` вначале вычисляется выражение `W` и если результат – «истина», то выполняется `O1`, иначе управление сразу передается следующему по порядку оператору программы.

Простейшими логическими выражениями являются отношения. Знаки отношений записываются следующим образом: `>` и `<` – так же, как в математике, знаки `=`, `≤`, `≥`, `≠` записываются парами символов `==`, `<=`, `>=`, `!=` соответственно. Более сложные логические выражения рассмотрены в следующем разделе.

Пример. Требуется записать условный оператор, вычисляющий новое значение `Y` по заданным значениям `A`, `B`, `X`, `Y` по формуле

$$Y = \begin{cases} 2, & \text{если } (A > B) \ \& \ (B \geq 3), \\ Y, & \text{если } (A > B) \ \& \ (B < 3), \\ X, & \text{в остальных случаях} \end{cases}$$

то есть в соответствии с алгоритмом



Вот этот оператор:

```
if (A > B)
    if (B >= 3)
        Y=2;
    else
```

```

;
else
    Y=X;

```

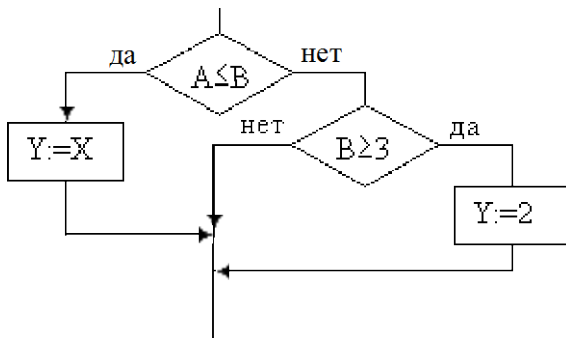
Этому оператору `if else` подчинен оператор присваивания `Y=X;` и еще один оператор `if else`, который, в свою очередь, содержит *пустой оператор* (после первого `else`) и оператор присваивания `Y=2;`. Необходимость использования `else` во вложенном условном операторе вытекает из следующего правила: `else` относится к ближайшему предшествующему `if`, у которого нет части `else`. Можно было бы не использовать `else` во вложенном условном операторе, но тогда пришлось бы заключить его в операторные скобки, то есть заменить его блоком `{if (B >= 3) Y=2 }`:

```

if (A > B)
{
    if (B >= 3)
        Y=2;
}
else
    Y=X;

```

Пример. Для задачи предыдущего примера можно составить другой алгоритм:



Тогда соответствующим ему оператором `if else` будет

```

if (A<=B)
    Y=X;
else
    if (B>=3)
        Y=2;

```

и вложенный в него условный оператор естественно использовать в сокращенной форме.

## Сложные логические выражения

Операциями над логическими данными в выражениях Си являются:

- *отрицание* (другое название – *не*), знак операции **!** (восклицательный знак),
- *конъюнкция* (другие названия – *логическое произведение* или просто *И*), знак операции **&&**,
- *дизъюнкция* (другие названия – *логическая сумма* или просто *ИЛИ*), знак операции **||**,

Отрицание является унарной операцией, остальные – бинарными.

В математике операции **!** (отрицания) соответствует знак  $\neg$  перед аргументом ( $\neg X$ ) или черта над аргументом ( $\bar{X}$ ); конъюнкции соответствует знак **&** или  $\cdot$  или  $\wedge$  ( $X \& Y$  или  $X \cdot Y$  или  $X \wedge Y$ ); дизъюнкции соответствует знак **v** ( $X v Y$ ).

Ниже представлены их таблицы перечисленных логических функций.

X	! X	X	Y	X && Y	X	Y	X    Y
Ложь	Истина	Ложь	Ложь	Ложь	Ложь	Ложь	Ложь
Истина	Ложь	Истина	Ложь	Ложь	Истина	Ложь	Истина
		Ложь	Истина	Ложь	Ложь	Истина	Истина
		Истина	Истина	Истина	Истина	Истина	Истина

*Логические выражения* могут содержать отношения выражений (в частности констант, переменных, обращений к функциям) различных типов, разделенные знаками логических операций и круглыми скобками. В первую очередь вычисляются выражения в скобках и обращения к функциям, затем – отношения и уже после них – логические операции в соответствии с их приоритетами. Операция *отрицание* имеет наивысший приоритет и выполняется в порядке справа налево. Приоритет операции *конъюнкция* выше, чем у операции *дизъюнкция*, а последовательности этих операций выполняются в порядке слева направо.

Пример. Составить условный оператор для вычисления нового значения Y по формуле

$$Y = \begin{cases} A, & \text{если } (A \cdot B > 1) \& (A > 0), \\ B, & \text{если } (A + B > 1) \& (A < 0), \\ Y - \text{в остальных случаях} \end{cases}$$

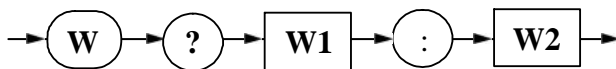
Как видно из задания, при истинности одного из условий другие условия будут иметь значение «ложь», поэтому для вычисления лучше использовать не два, а один условный оператор `if else` (с вложенным в него другим оператором `if`), что приведет к сокращению вычислений:

```
if (A*B>1 && A>0)
    Y=A;
else
    if (A+B>1 && A<0)
        Y=B;
```

Скобки можно использовать не только для изменения порядка вычислений в логических выражениях, но и для повышения их наглядности (например, отношения, входящие в сложные логические выражения, можно заключить в скобки).

## Условное выражение (тернарный оператор)

Конструкцию условного выражения описывает синтаксическая диаграмма



где

W – выражение,

W1 – выражение, вычисляемое в случае, если значение W не равно нулю (`true`),

W2– выражение, вычисляемое в случае, если значение W равно нулю (`false`).

Значением условного выражения будет значение выражения W1, если значением W не равно нулю, иначе – значение выражения W2.

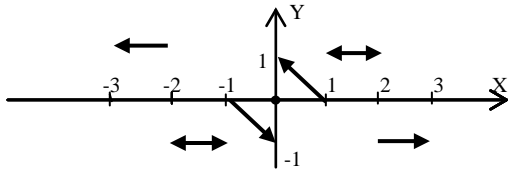
Условное выражение отличается от оператора `if else` следующим.

- оно может быть частью другого выражения или частью оператора, представляющей выражение, например, оператор `printf("%f", x>0 ? sin(x) : 0);` при `x>0` выведет значение выражения `sin(x)`, а при `x<=0` - значение `0`.

- выражения W1 и W2 могут быть отдельными операторами (присваивания, вызова функций), но не могут быть блоками.

## 3.2. Примеры выполнения задания

1. Составить программу вывода значений функции Y(X), заданной графиком (функция не определена при  $|X|>3$ )



или, что то же самое, – формулой

$$Y(X) = \begin{cases} Z \cdot (1 - |X|), & \text{если } 0 \leq |X| \leq 1, \\ Z, & \text{если } 1 < |X| < 2, \\ -Z, & \text{если } 2 \leq |X| < 3, \\ \text{в остальных случаях – не определена,} \end{cases}$$

где  $Z = -1$ , если  $X < 0$ ;  $Z = 0$ , если  $X = 0$ ;  $Z = 1$ , если  $X > 0$

двумя способами:

с помощью минимального числа операторов `if else`, без применения логических операций (`!`, `&&`, `||`),

с помощью минимального числа операторов `if` без `else` с применением логических операций.

Вывести с поясняющими текстами значение  $X$  и вычисленные значения функции.

```
#include "stdafx.h"
#include "math.h"
char* Ruc(char s[])
{ //Функция перекодирования русских букв.
  //Используется в операторах вывода.
  int i;
  static char ss[256];
  for (i=0; s[i] != '\0'; i++)
  {
    if (s[i] >= -64 && s[i] <= -17)
      ss[i] = (-64 + s[i]); //А..п
    else if (s[i] >= -16 && s[i] <= 0)
      ss[i] = (char)(-16 + s[i]); //р..я
    else if (s[i] == -72)
      ss[i] = (char)(-15); //ё
    else if (s[i] == -88)
      ss[i] = (char)(-16); //Ё
```



```

        else
            ss[i]=s[i];
    }
    ss[i]='\0';
    return ss;
}
int _tmain(int argc, _TCHAR* argv[])
{
    float x=-2;
    float X=-0.3,A,Z,Y;//21.0001
    printf(Ruc("Введите значение аргумента: "));
    scanf("%f",&X);
    //Вычисление абсолютного значения X
    A=fabs(X);
    //Вычисление Z
    if (X>0)            Z=1.0;
    else if (X==0)     Z=0.0;
    else                Z=-1.0;
    // Вычисление Y без применения логических операций
    if (A>=3)
        printf(Ruc("Функция не определена. \n"));
    else
    {
        if (A<=1.0)
            Y=Z*(1-A);
        else if (A<2.0)
            Y=Z;
        else
            Y=-Z;
        printf("Y = %f\n",Y);
    }
    // Вычисление Y с применением логических операций
    if (A>=3)
        printf(Ruc("Функция не определена. \n"));
    if (A<=1)
        printf("Y = %f\n",Z*(1-A));
}

```

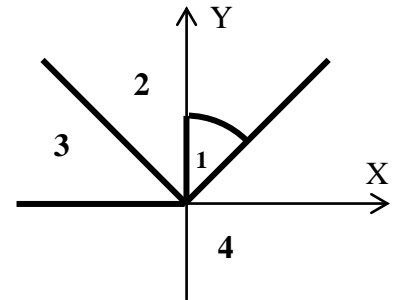
```

    if (A>1 && A<2)
        printf("Y = %f\n", Z);
    if (A>=2 && A<3)
        printf("Y = %f\n", -Z);
    return 0;
}

```

2. Составить программу вычисления  $Z$  – номера области (см. рисунок), в которую попадает точка с координатами  $(X, Y)$ , двумя способами:

- с помощью операторов `if else` без применения логических операций и сохранением результата в переменной  $Z1$ ,
- с помощью операторов `if else` с применением логических операций и сохранением результата в переменной  $Z2$ .



Все области, кроме области с номером 1 с границей в виде дуги окружности радиуса  $R=5$ , бесконечны. Точку, лежащую на границе областей, можно считать принадлежащей любой из них.

Вывести с поясняющими текстами значения  $X$ ,  $Y$  и вычисленные значения  $Z1$  и  $Z2$ .

```
#include "stdafx.h"
```

```

int _tmain(int argc, _TCHAR* argv[])
{
    const float R=5.0;
    float X,Y;
    int Z1,Z2;
    //Ввод координат точки
    printf("X, Y: ");
    scanf("%f %f",&X, &Y);
    //Определение Z1 - номера области, которой
    //принадлежит точка, без применения логических операций.
    //Если точка лежит в нижней полуплоскости
    //или на оси X,
    if (Y<=0.0) //то
        Z1=4; //переменной Z1 присвоить значение 4,

```

```

else        //иначе (то есть точка лежит в верхней
            //полуплоскости),
if (Y<=X)   //если Y не больше X, то
    Z1=4; //переменной Z1 присвоить значение 4,
else       //иначе (то есть точка вне области 4),
    if (Y<-X) //если точка лежит ниже прямой,

                //разделяющей области 2 и 3, то
        Z1=3; //переменной Z1 присвоить значение 3,
else       //иначе (то есть точка лежит выше или
            //на прямой, разделяющей области 2 и 3),
    if (X<0) //если X<0, то переменной
        Z1=2; //Z1 присвоить значение 2,
else      //иначе (то есть точка лежит в первой
            //четверти),
    if (X*X+Y*Y <= R*R) //если расстояние до точки от
        //начала координат при X>0 не превосходит R, то
        Z1=1; //точка лежит в области 1,
    else //иначе (то есть расстояние до точки
        //от начала координат превосходит R),
        //значит
        Z1=2; //точка лежит в области 2.
printf("Z1 = %d\n", Z1);

```

//Определение номера Z2 - области,

//которой принадлежит точка,

//с применением логических операций.

```

if (Y<=0 || Y<=X) //Если точка принадлежит области 4, то
    Z2=4; //переменной Z2 присвоить значение 4,
else //иначе (то есть точка вне области 4),
    if (Y<-X) //если точка принадлежит области 3, то
        Z2=3; //переменной Z2 присвоить значение 3,
    else //иначе (то есть точка вне областей 3 и 4),
        if (X*X+Y*Y <= R*R && X>0) // если точка
            //принадлежит области 1, то

```

```

        Z2=1; //переменной Z2 присвоить значение 1,
else //иначе (то есть точка вне
        //областей 1, 3 и 4),
        Z2=2; //переменной Z2 присвоить значение 2.
printf("Z2 = %d\n", Z2);
return 0;
}

```

### 3.3. Задания для самостоятельной работы

В заданиях с номерами от 1 до 25 требуется для зависимости  $Y(X)$ , заданной аналитически или графиком, составить программу вычисления для вводимого  $X$ :

$Y1 = Y(X)$  – с помощью минимального числа операторов `if else`, без применения логических операций,

$Y2 = Y(X)$  – с помощью минимального числа операторов `if` без `else`, с применением логических операций,

и вывода с поясняющими текстами вычисленных значений  $Y1$  и  $Y2$ .

Для значений аргумента, при которых функция не определена, выводить соответствующие сообщения.

В заданиях с графиками функций (см. первый пример выполнения задания в разделе 3.2):

- стрелка на линии графика указывает открытую границу интервала, в котором функция имеет заданное положением линии значение,
- в точках отсутствия линии графика функция не определена.

$$29. Y(X) = \begin{cases} 1 & \text{при } X < -2, \\ X/2 & \text{при } -2 \leq X < 0, \\ \text{не определена} & \text{при } X = 0, \\ 2 & \text{– в остальных случаях.} \end{cases}$$

$$30. Y(X) = \begin{cases} 0 & \text{при } -1 > X, \\ 1 & \text{при } -1 \leq X < 0, \\ -1 & \text{при } 0 \leq X < 2, \\ 1 & \text{при } 2 \leq X. \end{cases}$$

$$31. Y(X) = \begin{cases} X & \text{при } 0 > X + X^2 > -0,2 \\ X^2 & \text{при } 0 < X + X^2, \\ \text{иначе} - \text{не определена.} \end{cases}$$

$$32. Y(X) = \begin{cases} 1 & \text{при } -1 > X, \\ 0 & \text{при } -1 \leq X < 0, \\ -1 & \text{при } 0 \leq X < 1, \\ 0 & \text{при } 1 \leq X. \end{cases}$$

$$33. Y(X) = \begin{cases} -1 & \text{при } -2 > X, \\ X+1 & \text{при } -2 \leq X < 0, \\ 1 & \text{при } 0 \leq X < 1, \\ 0 & \text{при } 1 \leq X. \end{cases}$$

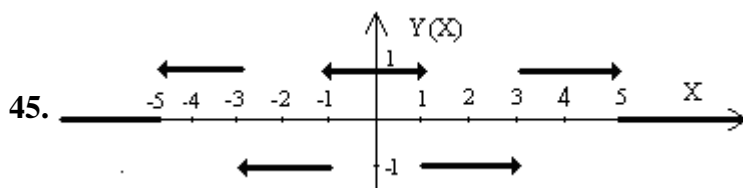
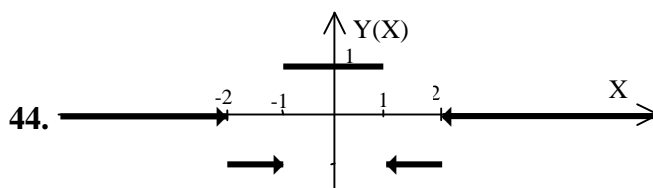
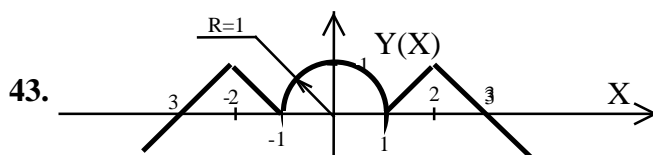
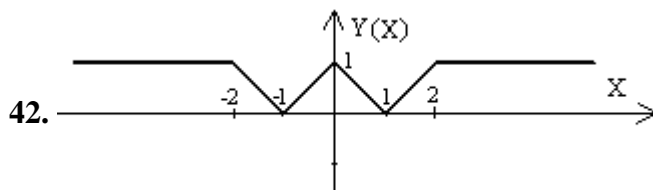
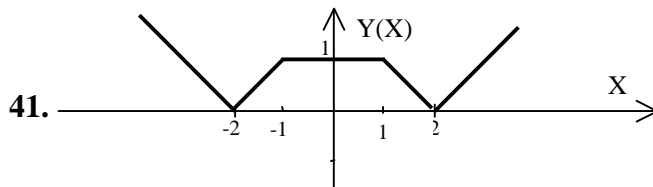
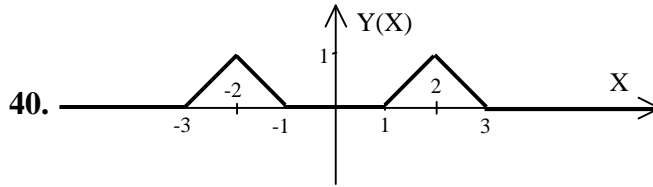
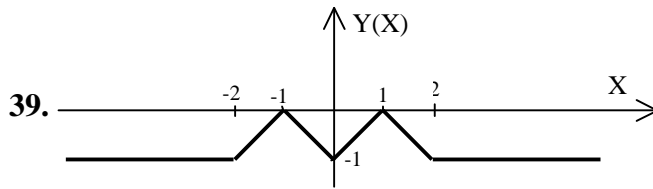
$$34. Y(X) = \begin{cases} 0 & \text{при } X < -2, \\ 1 & \text{при } -2 \leq X < -1, \\ 0 & \text{при } -1 \leq X < 0, \\ 1 & \text{при } 0 \leq X. \end{cases}$$

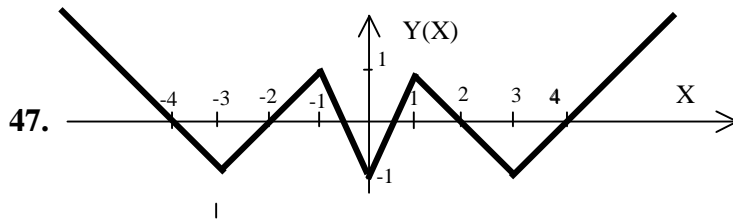
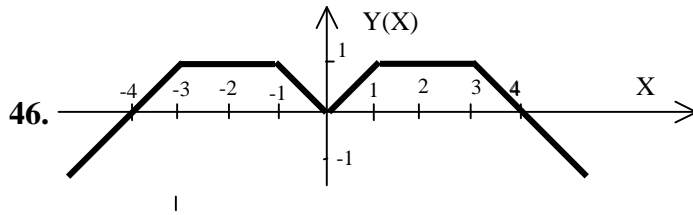
$$35. Y(X) = \begin{cases} 0 & \text{при } X < -1, \\ X+1 & \text{при } -1 \leq X < 0, \\ X & \text{при } 0 \leq X < 1, \\ 0 & \text{при } 1 \leq X. \end{cases}$$

$$36. Y(X) = \begin{cases} -1 & \text{при } X < -1, \\ X & \text{при } -1 \leq X < 1, \\ -X+2 & \text{при } 1 \leq X < 2, \\ 0 & \text{при } 2 \leq X. \end{cases}$$

$$37. Y(X) = \begin{cases} -1/X & \text{при } X < -3, \\ \sqrt{-X} & \text{при } -3 \leq X < 0, \\ X^2 & \text{при } 0 \leq X < 1, \\ \sqrt{X} & \text{при } 1 \leq X. \end{cases}$$

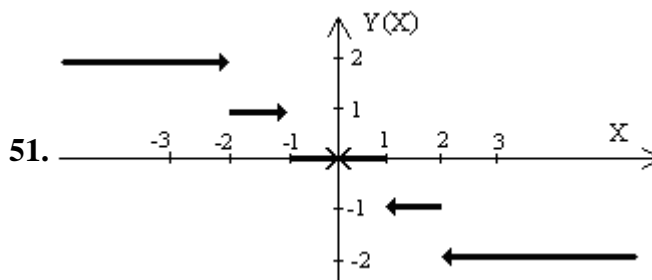
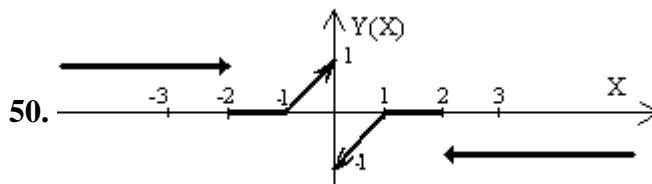
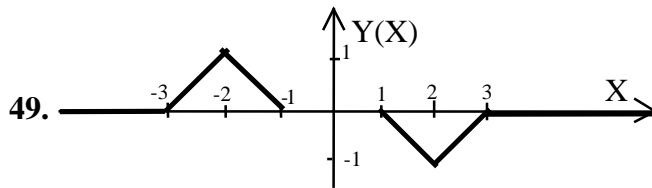
$$38. Y(X) = \begin{cases} -2-X & \text{при } X \leq 0, \\ 0 & \text{при } 0 < X < 1, \\ X & \text{при } 1 \leq X < 3, \\ 1-X & \text{при } 3 \leq X. \end{cases}$$

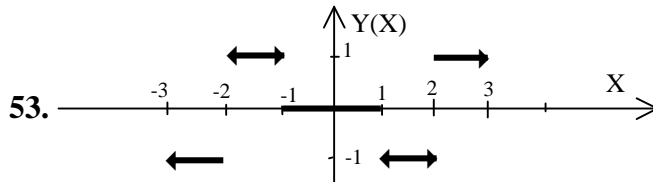
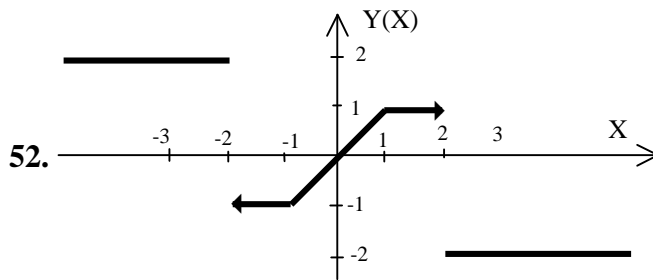




48. 
$$Y(X) = \begin{cases} 0, & \text{если } |X| < 3, & \text{иначе} \\ 1, & \text{если } \lfloor |X| \rfloor \oplus 2 \text{ четное, иначе} \\ -1, & \text{если } \lfloor |X| \rfloor \oplus 2 \text{ нечетное,} \end{cases}$$

где скобки  $\lfloor \ \rfloor$  обозначают целую часть числа,  
а знак  $\oplus$  – остаток от деления целого числа на 2



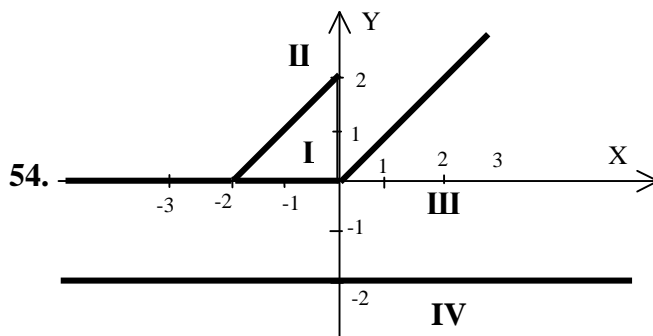


В заданиях с номерами от 26 до 30 требуется для рисунков, на которых области обозначены римскими цифрами, составить программу вычисления для вводимых  $X$  и  $Y$ :  $Z1$  – номера области с помощью минимального числа операторов `if else`, без применения логических операций,

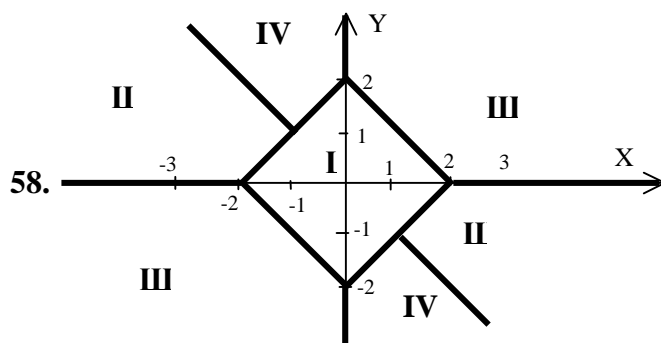
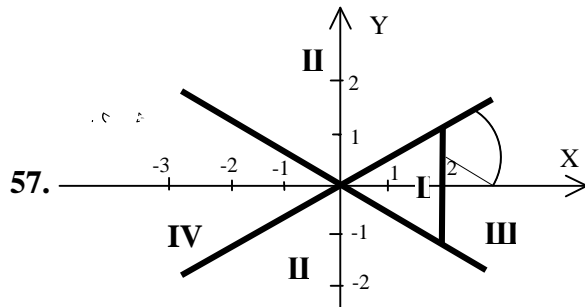
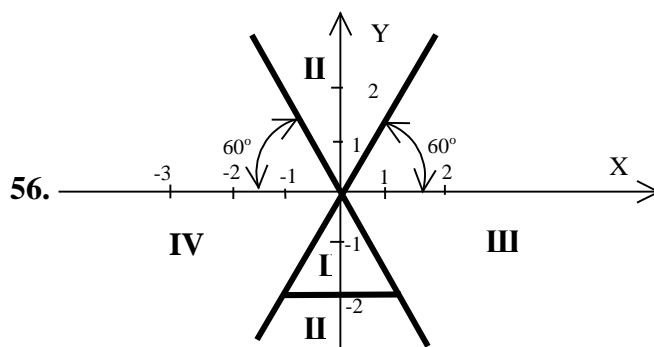
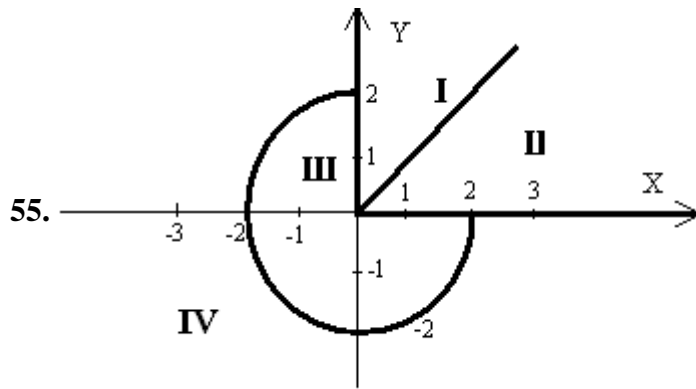
$Z2$  – номера области с помощью минимального числа операторов `if` без `else`, с применением логических операций.

и вывода с поясняющими текстами вычисленных значений  $Z1$  и  $Z2$ .

Точку, лежащую на границе областей, можно считать принадлежащей любой из них.







## Вопросы по самопроверке

1. Какие операторы используются в программах разветвляющейся структуры?
2. Может ли в подчинении части `if` и\или `else` быть другой оператор `if else` или `if` без `else`?
3. Как в сложном операторе `if else`, содержащем другие операторы `if else`, найти для части `else` соответствующую часть `if`?
4. Что может быть частью логического выражения?
5. Какими знаками обозначают отношения и логические операции?
6. Каков порядок вычисления частей логического выражения?
7. Может ли тернарный оператор быть частью логического выражения?

## 4. Программы циклической структуры

### 4.1. Средства разработки программ циклической структуры

Программой циклической структуры называют такую программу, в которой операторы могут повторно, при изменяющихся значениях переменных выполняться несколько раз, образуя *цикл*. Различают следующие виды циклов (для их организации используются специальные сложные операторы - *операторы циклов*):

*цикл с заданным числом повторений* или *цикл с параметром* (операторы цикла `for`),

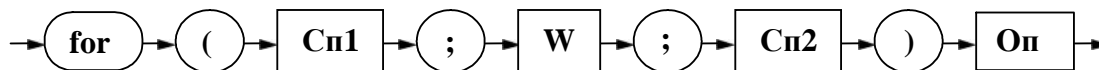
*цикл с предусловием* (оператор цикла `while`),

*цикл с постусловием* (оператор цикла `do while`).

В циклах можно выделить управляющие части, определяющие начало и условия выполнения цикла, и *тело цикла* - части из одного оператора или блока, выполняющие необходимые преобразования данных. Цикл называют *простым*, если в его теле нет других циклов.

#### Цикл с параметром (for)

Структура оператора цикла `for` описывается синтаксической диаграммой



где используются следующие обозначения:

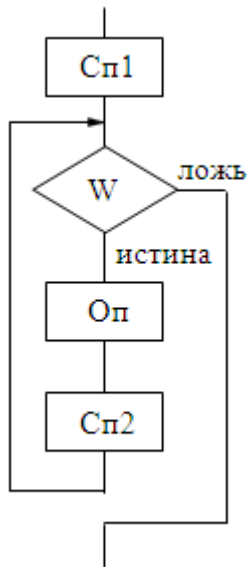
Сп1 – список операторов присваивания, задающих начальные значения одной или несколькими переменными - *параметрам цикла*, которые могут использоваться в выражении W и других частях цикла. Список может отсутствовать, если параметры получили требуемые значения до входа в цикл.

W – выражение, вычисляемое перед передачей управления в тело цикла, если его значение «истина» (не ноль), или следующему за циклом оператору, если его значение «ложь» (ноль). Выражение может отсутствовать, если выход из цикла организован в его теле, например, с помощью оператора `break` (см. ниже).

Сп2 – список операторов, задающих новые значения одному или нескольким параметрам цикла. Он может отсутствовать, если значения параметров цикла изменяются, при необходимости, в теле цикла.

Оп - тело цикла, в качестве которого должен использоваться или блок, или один оператор, возможно, пустой.

Алгоритма выполнения цикла представляет следующая схема



Например, в программе

```

#include "stdafx.h"
#include "math.h"

int _tmain(int argc, _TCHAR* argv[])
{
    int i;
    for (i=0; i<=6; i++)
        printf("%10d%8.2f\n", i*10, sin(3.14/18*i));
    return 0;
}
  
```

оператор `printf` будет выполняться 7 раз при значениях параметра целого типа `i`, изменяющемся от 0 до 6 с шагом 1.

На экран будет выведена таблица, в первом столбце которой будут целые числа 0, 10, 20, ..., 60, представляющие величины углов в градусах, а во втором – соответствующие им значения синуса:

0	0.00
10	0.17
20	0.34
30	0.50
40	0.64

```

50    0.77
60    0.87

```

Параметром цикла `for` может быть и вещественная переменная, а при наличии в списках `Sp1` и `Sp2` нескольких операторов они будут выполняться в порядке слева направо, как в следующем примере

```

int i;
float r;
. . .
for (i=1, r=-1.0; i<=6; ++i, r/=-i)
    printf("%10d%8.4f\n", i, r);

```

Результатом выполнения этого фрагмента программы будет таблица

```

1 -1.0000
2  0.5000
3 -0.1667
4  0.0417
5 -0.0083
6  0.0014

```

В приведенном примере переменная `i` является основным параметром цикла, определяющим условие выхода из него, а переменная `r` – дополнительным параметром цикла, изменялись синхронно с `i`, но по закону геометрической прогрессии.

При наличии в первом и последнем разделах заголовка цикла `for` нескольких операторов (как в рассмотренном примере) они должны разделяться знаком «запятая», и выполняться будут в порядке слева направо. Это позволяет сделать компактное описание цикла, возможно, в ущерб пониманию алгоритма. Для улучшения читаемости алгоритма лучше вынести все работы, связанные с собственно обработкой данных, а не организацией цикла, за пределы его заголовка, разместив соответствующие операторы в теле цикла и, если требуется, до входа в цикл. Для рассмотренного примера это выглядело бы так

```

int i;
float r;
. . .
r=-1.0;
for (i=1; i<=6; ++i)
{

```

```

printf ("%10d%8.4f\n", i, r);
r/=-i-1;
}

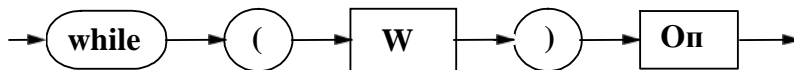
```

В общем случае условие выхода из цикла может строиться на основе переменных, изменяющих свои значения в теле цикла, а не в заголовке цикла `for`. Например, в последнем фрагменте программы заголовок цикла можно заменить следующим

```
for (i=1; fabs(r)>0.0013; ++i)
```

### Цикл с предусловием (while)

Структура оператора цикла `while` описывается синтаксической диаграммой



где используются следующие обозначения:

W – выражение,

Оп – тело цикла - оператор или блок, выполняемый внутри цикла.

Заголовок цикла – конструкция, предшествующая части Оп, управляет выполнением цикла следующим образом: тело цикла будет последовательно выполняться, пока выражение W имеет отличное от нуля значение («истина»), или не будет выполняться вообще, если при входе в цикл значение выражения окажется равным нулю («ложь»).

Например, фрагмент программы

```

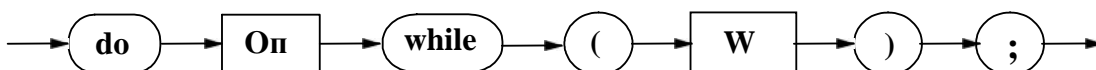
i=0;
while (i<=60)
{
    printf ("%10d%8.2f\n", i, sin(3.14/180*i));
    i=i+10;
}

```

буде выполнять ту же работу, что и в первом из предыдущих примеров на оператор `for`.

### Цикл с постусловием (do while)

Структура оператора цикла `do while` описывается синтаксической диаграммой



Внутри такого цикла может находиться либо один оператор, либо блок, который будет выполняться один или более раз до получения выражением  $W$  значения ноль («ложь»). Например, такую же таблицу, что и в первом примере с применением оператора `for`, будет выводить следующий фрагмент программы:

```

i=0;
do
{
    printf( "%10d%8.2f\n", i, sin(3.14/180*i) );
    i=i+10;
}
while ( i<=60 );

```

Существует возможность и досрочного выхода из любого цикла, организованного рассмотренными операторами, либо с помощью оператора безусловного перехода `goto` (их мы не будем использовать), либо с помощью оператора `break`.

В теле любого из рассмотренных циклов допускается использовать оператор `continue`. Его действие сводится к тому, что сразу происходит переход к очередному выполнению тела цикла (в циклах `for` с очередным значением параметра), или выход из цикла, если выполнено условие его завершения.

Следующий пример - на использование операторов `break` и `continue`

```

. . .
#include "conio.h"
. . .
int d,i;
. . .
d=0;
for ( i=1; i<=3; ++i )
{
    printf( Ruc( "\nВведите первый символ пароля: " ) );
    if ( getch() != 'a' ) //getch требует #include "conio.h"
        continue;
    printf( Ruc( "\nВведите второй символ пароля: " ) );
    if ( getch() == 'b' )
    {

```

```

        d=1;
        break;
    }
}

if (!d)
{
    printf(Ruc( "\nОшибка! Работа программы будет завершена.\n" ));
    return 0;
}
printf(Ruc( "\nВы допущены к работе с программой. \n" ));
. . . //продолжение программы

```

В этом примере программа предоставляет три попытки ввода пароля из двух символов при запуске в режиме без отладки (Debug\Start Without Debugging). Ввод символа выполняется функцией `getch()`, требующей использования директивы `#include "conio.h"`. Функция возвращает символ нажатой клавиши без его отображения в окне программы. При ошибке ввода первого символа оператором `continue` происходит переход к очередному выполнению тела цикла – повторному вводу первого символа пароля. Если первый символ введён правильно, а второй – нет, то очередная попытка ввода пароля также начинается с ввода первого символа.

Если за три попытки не удалось правильно ввести пароль (переменная `d` после выхода из цикла сохранила значение 0), то программа выводит сообщение «Ошибка! Работа программы будет завершена.» и завершает работу.

При успешном вводе пароля на любом шаге выполнения тела цикла переменная `d` получит значение 1, произойдёт выход из цикла по оператору `break` и пользователь получит сообщение «Вы допущены к работе с программой.»

## 4.2. Вычисление и вывод данных в виде таблицы

Простейшими примерами применения операторов цикла на практике являются программы вычисления значений функций при изменяющихся значениях аргумента и вывода данных в виде таблиц с заголовками. В качестве аргумента обычно выступает



переменная, изменяющаяся в цикле по требуемому закону. Она может быть как дополнительным, так и основным параметром в циклах `for`, `while` и `do while`.

При работе с вещественными данными необходимо иметь в виду, что их значения могут представляться с ошибкой, что эти ошибки могут зависеть от типа переменных, накапливаться при выполнении арифметических операций и приводить к непредусмотренному программистом выполнению программы. Рассмотрим два фрагмента программы вывода таблицы значений аргумента и функции, где аргумент - дополнительный параметр в цикле является вещественной переменной с именем X:

1)

```
float X, H, Xk; //Все переменные типа float
. . .
X=0.0;
H=1.0/3.0;
Xk=5.0/3.0; //==1.6(6)
while (X<=Xk)
{
    printf("\n%6.2f %8.2f", X, sin(X));
    X=X+H;
}
```

Результат выполнения:

0.00	0.00
0.33	0.33
0.67	0.62
1.00	0.84
1.33	0.97

2)

```
H=1.0/3.0;
X=-H;
Xk=5.0/3.0;
do
{
    X=X+H;
    printf("\n%6.2f %8.2f", X, sin(X));
}while (X!=Xk);
```

Казалось бы, оба фрагмента выведут на экран таблицы значений синуса для аргумента  $X$ , изменяющегося от 0 до  $5/3$  включительно с шагом  $1/3$ . Но в действительности первый фрагмент не выведет значения  $5/3$  и  $\sin(5/3)$ , а второй не обеспечит завершение выполнения цикла при  $X$  равном  $5/3$ , так что цикл продолжит выполняться и при больших значениях  $X$ .

Во избежание подобных ситуаций следует вместо проверок вещественных данных на равенство использовать проверки на  $<$  (меньше) и/или  $>$  (больше) с некоторым достаточно малым запасом, превышающим точность представления чисел данного типа и не нарушающим логику работы программы. Например, рассмотренные фрагменты программы можно изменить так

1)

```
float X, H, Xk; //Все переменные типа float
. . .
X=0.0;
H=1.0/3.0;
Xk=5.0/3.0+H/2.0;
while (X<Xk) //и при X=5.0/3.0 цикл будет выполнен.
{
    printf("\n%6.2f %8.2f", X, sin(X));
    X=X+H;
}
```

Результат выполнения:

0.00	0.00
0.33	0.33
0.67	0.62
1.00	0.84
1.33	0.97
1.67	1.00

2)

```
H=1.0/3.0;
X=-H;
//Запас в H/2.0 больше ошибки
```

```

//представления вещественных чисел
Xk=5.0/3.0-N/2.0;
do
{
    X=X+N;
    printf("\n%6.2f %8.2f", X,sin(X));
}while (X<Xk); //и выход из цикла будет, и будет
//только при X, равном 5.0/3.0, что больше Xk.

```

Результат выполнения будет таким же, как и для предыдущего фрагмента программы.

Рассмотрим решение этой же задачи вывода таблицы значений синуса, но при использовании оператора цикла `for` с параметром целого типа. Для этого потребуется заранее вычислить, сколько раз цикл должен выполняться, воспользовавшись формулой

$$N = \left\lceil \frac{X1 - X0}{H} \right\rceil + 1,$$

где

$N$  – искомое число повторений цикла,

$X0$  и  $X1$  – начальное и конечное значения аргумента,

$H$  – шаг изменения аргумента,

а скобки  $\lceil \rceil$  в формуле обозначают округление.

Округление вещественного числа можно получить, добавив к нему 0,5 и применив к результату библиотечную функцию `ceil` (она возвращает наименьшее целое, не меньшее аргумента), или вычлв из него 0,5 и применив к результату библиотечную функцию `floor` (она возвращает наибольшее целое, не большее аргумента).

Дополним условие задачи требованием сохранения значений аргумента и функции в массивах и вычисление очередного значения аргумента не методом накопления суммы, а умножением параметра цикла на шаг приращения аргумента. Так как в нашем случае  $X0=0$ ,  $X1=5/3$  и  $H=1/3$ , фрагмент программы с циклом `for` будет следующим

```

float X,H,Xk; //Все переменные типа float
float x[6], y[6]; //Массивы для сохранения значений
//аргумента и функции
int i,N;
. . .

```

```

H=1.0/3.0;
//Вычисление количества повторений цикла
N=ceil(5.0/3.0/H-0.5)+1;//или так,
//N=floor(5.0/3.0/H+0.5)+1;//или так
for (i=0;i<N;i++)
{
    X=i*H;
    x[i]=X;
    y[i]=sin(X);
    printf("\n%6.2f %8.2f", X,y[i]);
}

```

### 4.3. Пример выполнения задания с использованием цикла while

Упростив вычисления за счет использования дополнительных переменных и/или скобочных форм, вычислить значения функции

$$Y(X) = \frac{e^{-X^2}}{X} \sin(10X)$$

и ее производной

$$Y'(X) = \frac{X(e^{-X^2} 10 \cos(10X) - 2Xe^{-X^2} \sin(10X)) - e^{-X^2} \sin(10X)}{X^2}$$

на интервале значений X от 40° до 50° с шагом 1°.

Для проверки правильности вычислений  $Y'$  вычислить её значение по формуле без преобразований, а также по *разностной схеме*  $Y'(X) \approx \frac{Y(X + \Delta X / 2) - Y(X - \Delta X / 2)}{\Delta X}$  при  $\Delta X = 10^{-8}$ .

Результаты вычислений вывести в виде таблицы с заголовками столбцов. В начале каждой строки должен быть порядковый номер и соответствующее значение аргумента X.

После отладки программы при значениях X, изменяющихся от 40° до 50° с шагом 1°, добавить в программу ввод значений переменных A, B и H, которые должны представлять:

A - начало, B - конец диапазона значений X, H - шаг изменения X. Протестировать работу программы при разных вводимых данных.

```
#include "stdafx.h"
```

```

#include "math.h"
int _tmain(int argc, _TCHAR* argv[])
{
    const double Pi=3.14;
    double H=1;          //Шаг изменения аргумента в градусах
    double A=40;         //Начальное значение аргумента в градусах
    double B=50;         //Конечное значение аргумента в градусах
    double F1,X,Y,P,P1,C,D,E,S;

    int I;
    //ЧТОБЫ ОТЛАДИТЬ ПРОГРАММУ С НАЧАЛЬНЫМИ ЗНАЧЕНИЯМИ А, В и Н,
    //СДЕЛАЙТЕ КОММЕНТАРИЕМ СЛЕДУЮЩИЕ ОПЕРАТОРЫ printf И scanf
    printf("Enter A, B, H:");
    scanf("%lf%lf%lf",&A,&B,&H);
    //Перевод градусов в радианы
    A=Pi*A/180;
    H=Pi*H/180;
    B=Pi*B/180+H/2;
    //Вывод заголовка таблицы.
    printf("\n #      X      Y(X)      P\
          P1          F1");
    X=A;//Переменная X будет представлять
    //текущее значение аргумента
    I=0;//Переменная I будет представлять номер строки таблицы
    while (X<B)
    {
        //Увеличение значения счетчика строк таблицы
        I=I+1;
        //Вычисление значений дополнительных переменных
        C=X*X; D=10*X; E=exp(-C); S=sin(D);
        //Вычисление: Y - значения функции и
        //                P - ее производной
        //с использованием дополнительных переменных.
        Y=E*S/X;
        P= E*(D*cos(D)-S*(1+2*C))/C;
    }
}

```

```

//Вычисление P1 - контрольного значения
//производной без использования
//дополнительных переменных.
P1=( (exp(-X*X)*10.0*cos(10*X)
      -2*X*exp(-X*X)*sin(10*X))*X
      -exp(-X*X)*sin(10*X))/X/X;
//Вычисление F1 - контрольного значения
//производной по разностной схеме.
F1=( exp(-(X+5E-9)*(X+5E-9))*sin(10*(X+5E-9))/(X+5E-9)
      -exp(-(X-5E-9)*(X-5E-9))*sin(10*(X-5E-9))/(X-5E-9)
      )/1E-8;
//Вывод в строку таблицы вычисленных значений
printf("\n%3d%8.2lf%10.5lf%11.5lf%16.10lf%16.10lf"
       ,I,X*180/Pi,Y,P,P1,F1 );
X=X+N;    //Увеличение значения аргумента
} //while
printf("\n");
return 0;
}

```

При выводе заголовка таблицы управляющую строковую константу в операторе printf пришлось продолжить с начала новой строки текста программы, используя знак переноса \. Для выражений нет необходимости использования знака переноса, если продолжение текста выражения происходит на знаке операции, как, например, в операторах, вычисляющих значения переменных P1 и F1.

Результат работы программы при вводе исходных данных будет иметь вид.

Enter A, B, H:40 50 1

#	X	Y(X)	P	P1	F1
1	40.00	0.56371	5.17198	5.1719834346	5.1719833971
2	41.00	0.64022	3.60082	3.6008211921	3.6008211635
3	42.00	0.68954	2.06375	2.0637549307	2.0637549114
4	43.00	0.71268	0.60541	0.6054120938	0.6054120871
5	44.00	0.71134	-0.73589	-0.7358908651	-0.7358908483
6	45.00	0.68787	-1.92882	-1.9288233230	-1.9288233011

7	46.00	0.64505	-2.94945	-2.9494455353	-2.9494455323
8	47.00	0.58606	-3.78135	-3.7813491750	-3.7813491449
9	48.00	0.51428	-4.41555	-4.4155472441	-4.4155472168
10	49.00	0.43317	-4.85013	-4.8501348030	-4.8501347771
11	50.00	0.34620	-5.08975	-5.0897482960	-5.0897481951

Обратите внимание на достаточно близкие, с точностью до 7-го .. 8-го знака значения производной, вычисленные по аналитически полученной и по разностной формулам. Однако следует иметь в виду, что подобное сходство значений не всегда возможно и зависит как от самой функции, так и от значения аргумента и от точности представления данных в машине.

#### 4.4. Пример выполнения задания с использованием цикла `for`

Для функции 
$$F(X) = \frac{2X(X-1) - (X^2-1)\ln(X^2-1)}{(X^2-1)(X-1)^2}$$

при 9 значениях приращения аргумента  $DX=(0,5; 0,25; 0,125; \dots)$

вычислить:

1) точные значения приращений первообразной

$$DP(X) = \frac{\ln((X+DX)^2-1)}{(X+DX)-1} - \frac{\ln(X^2-1)}{X-1},$$

а также вычислить

2) по формуле  $F(X+DX/2) \cdot DX$  - приближенные значения

а) DP1, упростив вычисления за счет дополнительных переменных,

б) DP2, не используя дополнительных переменных, и

3)  $|DP-DP1|$  - абсолютные ошибки вычисленных приближенных значений.

Результаты вычислений и соответствующие значения  $DX$  вывести в виде таблицы с заголовками столбцов и номерами строк. В последней колонке таблицы вывести в экспоненциальной форме значения ошибки, чтобы был виден её порядок.

```
#include "stdafx.h"
#include "math.h"
int _tmain(int argc, _TCHAR* argv[])
{
    const double X=1.5;
    double DP, DP1, DP2, D, R, K, X1, DX;
```

```

int I,N;
//Вывод заголовка таблицы
printf( "%s%s"
        , " #      DX      DP      "
        , "DP1      DP2      |DP-DP1| " );
//Переменная DX будет представлять
//текущее приращение аргумента
DX=0.5;
for (I=1;I<10;I++)
{
    //Вычисление точного значения DP(X)
    DP=log( (X+DX)*(X+DX)-1)/(X+DX-1)-log(X*X-1)/(X-1);

    //Вычисление значений дополнительных переменных
    X1=X+DX/2; K=X1*X1; R=K-1; D=X1-1;
    //Вычисление приближенных значений DP(X)
    //с использованием дополнительных переменных
    DP1=(2 * X1 * D - R * log(R)) / R / D*D*DX;
    //без использования дополнительных переменных
    DP2=(2*(X+DX/2)*(X+DX/2-1)-((X+DX/2)*(X+DX/2)-1)
        *log((X+DX/2)*(X+DX/2)-1))
        /((X+DX/2)*(X+DX/2)-1)/(X+DX/2-1)/(X+DX/2-1)*DX;
    //Вывод в строку таблицы вычисленных значений
    printf( "\n%3d%10.5lf  %12.6lf %10.6lf %9.6lf      %e"
            , I,DX,DP,DP1,DP2,fabs(DP-DP2));
    //Изменение значения приращения аргумента
    DX=DX/2;
} //конец for
printf( "\n");
return 0;
}

```

Результат работы программы представляет следующая таблица.

#	DX	DP	DP1	DP2	DP-DP1
1	0.50000	0.652325	0.274404	0.487830	1.644955e-001



2	0.25000	0.518938	0.185754	0.475532	4.340651e-002
3	0.12500	0.345837	0.106737	0.337340	8.496082e-003
4	0.06250	0.203703	0.057105	0.202339	1.363851e-003
5	0.03125	0.111248	0.029526	0.111054	1.947223e-004
6	0.01563	0.058238	0.015011	0.058212	2.607041e-005
7	0.00781	0.029810	0.007568	0.029806	3.374592e-006
8	0.00391	0.015082	0.003800	0.015082	4.293168e-007
9	0.00195	0.007586	0.001904	0.007586	5.414117e-008

В этом примере, в отличие от предыдущего, длинная строка заголовка таблицы была разбита на две последовательно выводимые по формату %s строковые константы. Такой приём, то есть помещение строковой константы в списке вывода, приходится использовать также в случае перекодирования символов строки.

#### 4.5. Задания для самостоятельной работы

Во всех заданиях использовать только простые циклы.

1. Вычислить для первых 20 значений  $X = \frac{1}{2}, 1 - \frac{2}{3}, 1 - \frac{3}{4}, \dots$  и вывести в виде таблицы с заголовками:

- значения функции  $\ln(1+x)$ ,
- приближенные значения функции по формуле

$$X - \frac{X^2}{2} + \frac{X^3}{3} - \frac{X^4}{4} + \frac{X^5}{5},$$

используя скобочные формы и/или дополнительные переменные,

- приближенные значения функции по этой же формуле, не используя скобочные формы и дополнительные переменные,
- абсолютную и относительную ошибки приближенных значений.

Для организации цикла использовать оператор `for`. При вычислениях приближенных значений использовать только операции сложения, вычитания, умножения, деления.

2. Вычислить при  $X=(-0,5; -0,25; 0; 0,25; 0,5; 0,75; 1)$  и вывести в виде таблицы с заголовками:

- значения функции  $e^x$ ,

- приближенные значения функции по формуле

$$1 + X + \frac{X^2}{2!} + \frac{X^3}{3!} + \frac{X^4}{4!} + \frac{X^5}{5!} \quad ,$$

используя скобочные формы и/или дополнительные переменные,

- приближенные значения функции по этой же формуле, не используя скобочные формы и дополнительные переменные,
- абсолютную и относительную ошибки приближенных значений.

Для организации цикла использовать оператор `while`. При вычислениях приближенных значений использовать только операции сложения, вычитания, умножения, деления.

**3.** Вычислить при  $X$ , изменяющемся от 0,1 до  $\pi/3$  с шагом 0,05, и вывести в виде таблицы с заголовками:

- значения функции  $\sin(x)$ ,
- приближенные значения функции по формуле

$$X - \frac{X^3}{3!} + \frac{X^5}{5!} - \frac{X^7}{7!} + \frac{X^9}{9!} \quad ,$$

используя скобочные формы и/или дополнительные переменные,

- приближенные значения функции по этой же формуле, не используя скобочные формы и дополнительные переменные,
- абсолютную и относительную ошибки приближенных значений.

Для организации цикла использовать оператор `for`. При вычислениях приближенных значений использовать только операции сложения, вычитания, умножения, деления

**4.** Вычислить в цикле `do while` при  $X$ , изменяющемся от 0 до  $\pi/4$  с шагом 0,1, и вывести в виде таблицы с заголовками:

- значения функции  $\cos(x)$ ,
- приближенные значения функции по формуле

$$1 - \frac{X^2}{2!} + \frac{X^4}{4!} - \frac{X^6}{6!} + \frac{X^8}{8!} \quad ,$$

используя скобочные формы и/или дополнительные переменные,

- приближенные значения функции по этой же формуле, не используя скобочные формы и дополнительные переменные,
- абсолютную и относительную ошибки приближенных значений.

При вычислениях приближенных значений использовать только операции сложения, вычитания, умножения, деления.

**5.** Вычислить при  $X$ , изменяющемся от  $A$  до  $B$  с шагом  $H$ , и вывести в виде таблицы с заголовками:

- значения функции  $\text{tg}(x)$ ,
- приближенные значения функции по формуле

$$X + \frac{X^3}{3} + \frac{2X^5}{15} + \frac{17X^7}{315} + \frac{62X^9}{2835} ,$$

используя скобочные формы и/или дополнительные переменные,

- приближенные значения функции по этой же формуле, не используя скобочные формы и дополнительные переменные,
- абсолютную и относительную ошибки приближенных значений.

Для организации цикла использовать оператор `for`. При вычислениях приближенных значений использовать только операции сложения, вычитания, умножения, деления.

**6.** Вычислить при  $M$ , изменяющемся от 0 до 6 с шагом 0,5, и вывести в виде таблицы с заголовками:

- значения функции  $(1 + X)^M$ ,
- приближенные значения функции по формуле

$$1 + M \cdot X + \frac{M(M-1)X^2}{2!} + \frac{M(M-1)(M-2)X^3}{3!} + \frac{M(M-1)(M-2)(M-3)X^4}{4!} ,$$

используя скобочные формы и/или дополнительные переменные,

- приближенные значения функции по этой же формуле, не используя скобочные формы и дополнительные переменные,
- абсолютную и относительную ошибки приближенных значений.

Для организации цикла использовать оператор `while`. При вычислениях приближенных значений использовать только операции сложения, вычитания, умножения, деления.

**7.** Вычислить при  $X=(1; 0,5; 0,25; 0,125; 0,0625; 0,03125; 0,015625)$  и вывести в виде таблицы с заголовками:

- значения функции  $\sqrt{1 + X}$ ,

- приближенные значения функции по формуле

$$1 + \frac{X}{2} - \frac{X^2}{2 \cdot 4} + \frac{3X^3}{2 \cdot 4 \cdot 6} - \frac{3 \cdot 5X^4}{2 \cdot 4 \cdot 6 \cdot 8} \quad ,$$

используя скобочные формы и/или дополнительные переменные,

- приближенные значения функции по этой же формуле, не используя скобочные формы и дополнительные переменные,
- абсолютную и относительную ошибки приближенных значений.

Для организации цикла использовать оператор `for`. При вычислениях приближенных значений использовать только операции сложения, вычитания, умножения, деления.

**8.** Вычислить при  $X = \sin(5^\circ), \sin(10^\circ), \dots, \sin(60^\circ)$  и вывести в виде таблицы с заголовками:

- значения функции  $\arcsin(x)$
- приближенные значения функции по формуле

$$X + \frac{X^3}{2 \cdot 3} - \frac{3X^5}{2 \cdot 4 \cdot 5} + \frac{3 \cdot 5X^7}{2 \cdot 4 \cdot 6 \cdot 7} - \frac{3 \cdot 5 \cdot 7X^9}{2 \cdot 4 \cdot 6 \cdot 8 \cdot 9} \quad ,$$

используя скобочные формы и/или дополнительные переменные,

- приближенные значения функции по этой же формуле, не используя скобочные формы и дополнительные переменные,
- абсолютную и относительную ошибки приближенных значений.

Для организации цикла использовать оператор `for`. При вычислениях приближенных значений использовать только операции сложения, вычитания, умножения, деления/

**9.** Вычислить в цикле `do while` при первых 15 значениях

$X = \operatorname{tg}\left(\frac{1}{2}45^\circ\right), \operatorname{tg}\left(\frac{1}{3}45^\circ\right), \operatorname{tg}\left(\frac{1}{4}45^\circ\right), \dots$  и вывести в виде таблицы с заголовками:

значения функции  $\operatorname{arctg}(x)$

- приближенные значения функции по формуле

$$X - \frac{X^3}{3} + \frac{X^5}{5} - \frac{X^7}{7} + \frac{X^9}{9} \quad ,$$

используя скобочные формы и/или дополнительные переменные,

- приближенные значения функции по этой же формуле, не используя скобочные формы и дополнительные переменные,
- абсолютную и относительную ошибки приближенных значений.

- При вычислениях приближенных значений использовать только операции сложения, вычитания, умножения, деления.

**10.** Вычислить при  $X$ , изменяющемся от  $X_0$  до  $X_1$  с шагом  $H$ , и вывести в виде таблицы с заголовками:

- значения функции  $\frac{e^x - e^{-x}}{2}$

- приближенные значения функции по формуле

$$X + \frac{X^3}{3!} + \frac{X^5}{5!} + \frac{X^7}{7!} + \frac{X^9}{9!},$$

используя скобочные формы и/или дополнительные переменные,

- приближенные значения функции по этой же формуле, не используя скобочные формы и дополнительные переменные,

- абсолютную и относительную ошибки приближенных значений.

Для организации цикла использовать оператор `for`. При вычислениях приближенных значений использовать только операции сложения, вычитания, умножения, деления.

**11.** Для функции  $Y = \frac{Xe^{-x^2}}{1+X}$  и вводимого значения  $X$  вычислить:

- точное значение производной  $Y' = \frac{e^{-x^2} \cdot ((1-2X^2) \cdot (1+X) - X)}{(1+X)^2}$

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных,

- а также вычислить для 8-ми значений  $DX=(0,2; 0,04; 0,008; \dots)$ :

- приближенные значения приращений функции  $DY=Y(X+DX)-Y(X)$ ,

- приближенные значения производной по отношению  $DY/DX$ ,

- абсолютные ошибки приближенных значений производной.

Для организации цикла использовать оператор `while`. Результаты вычислений и соответствующие значения  $DX$  вывести в виде таблицы с заголовками столбцов.

**12.** Для функции  $Y = \frac{1+2X}{X^2-1}$  и вводимого значения  $X$  вычислить:

- точное значение производной  $Y' = \frac{2(X^2-1) - 2X(1+2X)}{(X^2-1)^2}$

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных,

- а также вычислить для значений  $DX=(0,0001; 0,001; 0,01; 0,1)$ :
- приближенные значения приращений функции  $DY=Y(X+DX/2)-Y(X-DX/2)$ ,
- приближенные значения производной по отношению  $DY/DX$ ,
- абсолютные ошибки приближенных значений производной.

Для организации цикла использовать оператор `for`. Результаты вычислений и соответствующие значения  $DX$  вывести в виде таблицы с заголовками столбцов.

**13.** Для функции  $Y = \frac{\ln(1 + \cos X)}{2^X + 1}$  и вводимого значения  $X$  вычислить:

- точное значение производной

$$Y' = \frac{-\sin X}{(1 + \cos X) \cdot (2^X + 1)} - \frac{\ln(1 + \cos X) \cdot 2^X \cdot \ln 2}{(2^X + 1)^2}$$

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных,

- а также вычислить для значений  $DX=(10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6})$ :
- приближенные значения приращений функции  $DY=Y(X+DX)-Y(X)$ ,
- приближенные значения производной по отношению  $DY/DX$ ,
- абсолютные ошибки приближенных значений производной.

Для организации цикла использовать оператор `for`. Результаты вычислений и соответствующие значения  $DX$  вывести в виде таблицы с заголовками столбцов.

**14.** Для функции  $Y = \frac{2^X}{1 + \ln(2 + \cos(X))}$  и вводимого значения  $X$  вычислить

- точное значение производной

$$Y' = \frac{2^X \ln 2 \cdot (1 + \ln(2 + \cos X)) + \frac{2^X \sin X}{2 + \cos X}}{(1 + \ln(2 + \cos X))^2}$$

а) упростив вычисления за счет дополнительных переменных,  
б) не используя дополнительных переменных,

- а также вычислить в цикле `do while` для значений  $DX=(0,00001; 0,0001; 0,001; 0,01; 0,1)$ :
- приближенные значения приращений функции  $DY=Y(X+DX/2)-Y(X-DX/2)$ ,
- приближенные значения производной по отношению  $DY/DX$ ,
- абсолютные ошибки приближенных значений производной

- и вывести полученные значения и соответствующие значения DX в виде таблицы с заголовками столбцов.

**15.** Для функции  $Y = \ln(1 + X^2) \operatorname{tg}(X^2)$  в точке  $X=0,3$  вычислить:

- точное значение производной

$$Y' = \frac{2X \cdot \operatorname{tg} X^2}{1 + X^2} + \frac{2X \ln(1 + X^2)}{\cos^2 X^2}$$

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных,

- а также вычислить в цикле `for` для значений  $DX=(0,00000025; 0,000005; 0,0001; 0,002; 0,04; 0,8)$ :

- приближенные значения приращений функции  $DY=Y(X+DX)-Y(X)$ ,
- приближенные значения производной по отношению  $DY/DX$ ,
- абсолютные ошибки приближенных значений производной.
- и вывести полученные значения и соответствующие значения DX в

виде таблицы с заголовками столбцов.

**16.** Для функции  $Y = \frac{\sin X}{\ln(2 + \sin^2 X)}$  и вводимого значения X вычислить:

- точное значение производной

$$Y' = \frac{\cos X \cdot \ln(2 + \sin^2 X) - \frac{2 \sin^2 X \cdot \cos X}{2 + \sin^2 X}}{\ln^2(2 + \sin^2 X)}$$

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных,

- а также вычислить в цикле `while` для значений  $DX=(0,0005; 0,001; 0,002; 0,004; 0,008; 0,016)$ :

- приближенные значения приращений функции  $DY=Y(X+DX/2)-Y(X-DX/2)$ ,
- приближенные значения производной по отношению  $DY/DX$ ,
- абсолютные ошибки приближенных значений производной.
- и вывести полученные значения и соответствующие значения DX в

виде таблицы с заголовками столбцов.

**17.** Для функции  $Y = \operatorname{arctg}\left(\sqrt{\frac{X+1}{1-X}}\right)$  и вводимого значения X вычислить:

$$- \quad \text{точное значение производной } Y' = \frac{1}{(1-X)^2 \left(1 + \frac{1+X}{1-X}\right) \sqrt{\frac{1+X}{1-X}}}$$

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных,

- а также вычислить в цикле `for` для семи значений  $DX=(0,000001; 0,000004; 0,000016; 0,000064; \dots)$ :

- приближенные значения приращений функции  $DY=Y(X+DX)-Y(X)$ ,

- приближенные значения производной по отношению  $DY/DX$ ,

- абсолютные ошибки приближенных значений производной.

- и вывести полученные значения и соответствующие значения  $DX$  в виде таблицы с заголовками столбцов.

**18.** Для функции  $Y = \frac{X+1}{(X+2)(X+3)}$  и вводимого значения  $X$  вычислить:

$$- \quad \text{точное значение производной } Y' = \frac{1-(X+2)X}{(X+2)^2(X+3)^2}$$

а) упростив вычисления за счет дополнительных переменных, и

б) не используя дополнительных переменных,

- а также вычислить в цикле `for` для 12-ти значений  $DX=(1/3, 1/9, 1/27, 1/81, \dots)$ :

- приближенные значения приращений функции  $DY=Y(X+DX)-Y(X)$ ,

- приближенные значения производной по отношению  $DY/DX$ ,

- абсолютные ошибки приближенных значений производной.

- и вывести полученные значения и соответствующие значения  $DX$  в виде таблицы с заголовками столбцов.

**19.** Упростив вычисления за счет использования дополнительных переменных и/или скобочных форм, вычислить в цикле `do while` значения функции

$$Y(X) = \frac{\operatorname{tg}^4 X}{4} - \frac{\operatorname{tg}^2 X}{2} - \ln(\cos^2 X)$$

и ее производной

$$Y'(X) = \frac{\operatorname{tg}^3 X}{\cos^2 X} - \frac{\operatorname{tg} X}{\cos^2 X} - 2\operatorname{tg} X$$

на интервале от  $-7,5^\circ$  до  $7,5^\circ$  с шагом  $0,75^\circ$ .



Для проверки правильности результата вычислить также значение производной по заданной формуле без преобразований.

Найденные значения вывести в виде таблицы с предшествующими порядковым номером и соответствующим значением аргумента X.

**20.** Упростив вычисления за счет использования дополнительных переменных и/или скобочных форм, вычислить значения функции

$$Y(X) = \left( \frac{(X-1)}{2} - \frac{(X-1)^2}{2} + \frac{(X-1)^4}{3} \right) (X^2 - 1)$$

и ее производной

$$Y'(X) = \left( \frac{1}{2} - (X-1) + \frac{4(X-1)^3}{3} \right) (X^2 - 1) + \left( \frac{(X-1)}{2} - \frac{(1-X)^2}{2} + \frac{(1-X)^4}{3} \right) 2X$$

на интервале от -1,1 до 1,0 с шагом 0,1.

Для проверки правильности результата вычислить также значение производной по заданной формуле без преобразований.

Найденные значения вывести в виде таблицы с предшествующими порядковым номером и соответствующим значением аргумента X. Для организации цикла использовать оператор `for`.

**21.** Для функции  $F(X) = \frac{2X(1-X) + (1+X^2)}{2(1-X)^2} \sqrt{\frac{1-X}{1+X^2}}$  при X= 0,5 и K приращений аргумента DX=(0,0005; 0,001; 0,002; 0,004; 0,008;...) вычислить:

- точное значение приращения первообразной

$$DP = \sqrt{\frac{1+(X+DX)^2}{1-(X+DX)}} - \sqrt{\frac{1+X^2}{1-X}}$$

- а также по формуле  $(F(X+DX/2) \cdot DX)$  - приближенные значения приращения первообразной:

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных,

- абсолютные и относительные ошибки в процентах для вычисленных приближенных значений.

Результаты вычислений и соответствующие значения DX вывести в виде таблицы с заголовками столбцов. Для организации цикла использовать оператор `while`.

22. Для функции  $F(X) = \frac{e^X - e^{-X}}{e^X + e^{-X}}$  и вводимого значения X при N приращени-

ях аргумента

$Dx = (-0,1; -0,1/4; -0,1/16; \dots)$  вычислить:

- точное значение приращения первообразной

$$DP = \ln(e^{X+DX} + e^{-(X+DX)}) - \ln(e^X + e^{-X}),$$

- а также вычислить

- по формуле  $F(X) \cdot Dx$  - приближенные значения приращения первообразной

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных,

- абсолютные ошибки и относительные ошибки в процентах для вычисленных приближенных значений.

Результаты вычислений и соответствующие значения  $Dx$  вывести в виде таблицы с заголовками столбцов. Для организации цикла использовать оператор `for`.

23. Для функции  $F(X) = \frac{\sin^3 X + 1}{\sin^2 X}$  и вводимого значения X при приращениях

аргумента

$Dx = (-0,0005; +0,001; -0,002; +0,004; -0,008; +0,016)$  вычислить:

- точное значение приращения первообразной

$$DP = -\cos(X + DX) - \operatorname{ctg}(X + DX) + \cos X + \operatorname{ctg} X,$$

- а также вычислить

- по формуле  $\frac{F(X + DX) + F(X)}{2} Dx$  - приближенные значения при-

ращения первообразной

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных,

- абсолютные ошибки и относительные ошибки в процентах для вычисленных приближенных значений.

Результаты вычислений и соответствующие значения  $Dx$  вывести в виде таблицы с заголовками столбцов. Для организации цикла использовать оператор `for`.

24. Для функции  $F(X) = \frac{-2Xe^{-X^2}}{1+e^{-X^2}}$  и вводимого значения X при K прираще-

ниях аргумента

$DX=(-0,0005; -0,001; -0,002; -0,004; \dots)$  вычислить в цикле `do while`:

- точное значение приращения первообразной

$$DP = \ln(1 + e^{-(X+DX)^2}) - \ln(1 + e^{-X^2}),$$

- а также вычислить
- по формуле  $F(X+DX/2) \cdot DX$  - приближенные значения приращения

первообразной

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных,

- абсолютные ошибки и относительные ошибки в процентах для вычисленных приближенных значений.

Результаты вычислений и соответствующие значения DX вывести в виде таблицы с заголовками столбцов.

25. Для функции  $F(X) = \frac{2}{(2X-1)^2 \sqrt{1 - (\frac{1}{2X-1})^2}}$  при  $X=1,5$  и K приращениях

аргумента

$DX=(5 \cdot 10^{-1}; 5 \cdot 10^{-2}; 5 \cdot 10^{-3}; 5 \cdot 10^{-4}; \dots)$  вычислить:

- точное значение приращения первообразной

$$DP = \arccos\left(\frac{1}{2(X+DX)-1}\right) - \arccos\left(\frac{1}{2X-1}\right),$$

- а также вычислить
- по формуле  $F(X) \cdot DX$  - приближенные значения приращения перво-

образной

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных,

- абсолютные ошибки и относительные ошибки в процентах для вычисленных приближенных значений.

Результаты вычислений и соответствующие значения DX вывести в виде таблицы с заголовками столбцов. Для организации цикла использовать оператор `for`.

26. Для функции  $F(X) = -\frac{2X}{(X^2 - 1)^2 + 1}$  и вводимого значения X при K приращении аргумента  $DX=(0,1; -0,05; 0,025; -0,0125; \dots)$  вычислить:

- точное значение приращения первообразной

$$DP = \operatorname{arctg}\left(\frac{1}{(X + DX)^2 - 1}\right) - \operatorname{arctg}\left(\frac{1}{X^2 - 1}\right),$$

- а также вычислить

- по формуле  $\frac{F(X + DX) + F(X)}{2}DX$  - приближенные значения приращения первообразной

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных,

- абсолютные ошибки и относительные ошибки в процентах для вычисленных приближенных значений.

Результаты вычислений и соответствующие значения DX вывести в виде таблицы с заголовками столбцов. Для организации цикла использовать оператор [while](#).

27. Для функции  $F(X) = \cos X - \frac{1}{\cos^2 X}$  и вводимого значения X при K приращениях аргумента  $DX=(0,08; 0,04; 0,02; \dots)$  вычислить:

- точное значение приращения первообразной

$$DP = (\sin(X + DX) - \operatorname{tg}(X + DX)) - (\sin X - \operatorname{tg}X),$$

- а также вычислить

- по формуле  $F(X+DX/2) \cdot DX$  - приближенные значения приращения первообразной

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных,

- абсолютные ошибки и относительные ошибки в процентах для вычисленных приближенных значений.

Результаты вычислений и соответствующие значения DX вывести в виде таблицы с заголовками столбцов. Для организации цикла использовать оператор [for](#).

28. Для функции  $F(X) = \frac{-1}{(X-1)^2 \sqrt{1 - \left(\frac{1}{X-1}\right)^2}}$  при  $X=10$  и 12 приращениях

аргумента

$Dx=(1/4, 1/6, 1/8, \dots)$  вычислить:

- точное значение приращения первообразной

$$DP = \arcsin(1/(X+Dx-1)) - \arcsin(1/(X-1)),$$

- а также вычислить

- по формуле  $F(X) \cdot Dx$  - приближенные значения приращения первообразной

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных,

- абсолютные ошибки и относительные ошибки в процентах для вычисленных приближенных значений.

Результаты вычислений и соответствующие значения  $Dx$  вывести в виде таблицы с заголовками столбцов. Для организации цикла использовать оператор `for`.

29. Для функции  $F(X) = \frac{X}{\sqrt{X^2 + 1}}$  при  $X=0,95$  и приращениях аргумента

$Dx=(0,0005; 0,001; 0,002; 0,04; 0,08; 0,016; 0,032)$  вычислить в цикле `do while`:

- точное значение приращения первообразной

$$DP = \sqrt{(X + Dx)^2 + 1} - \sqrt{X^2 + 1},$$

- а также вычислить

- по формуле  $\frac{F(X + Dx) + F(X)}{2} Dx$  - приближенные значения приращения первообразной

ращения первообразной

а) упростив вычисления за счет дополнительных переменных,

б) не используя дополнительных переменных,

- абсолютные ошибки и относительные ошибки в процентах для вычисленных приближенных значений.

Результаты вычислений и соответствующие значения  $Dx$  вывести в виде таблицы с заголовками столбцов.

30. Упростив вычисления за счет использования дополнительных переменных и/или скобочных форм, вычислить значения функции

$$Y(X) = \frac{a^{X^2-1} + a^{X-1}}{X-1} \quad \text{и ее производной}$$

$$Y'(X) = \frac{(2Xa^{X^2-1} + a^{X-1})\ln(a)(X-1) - (a^{X^2-1} + a^{X-1})}{(X-1)^2}$$

на 20-ти значениях  $X = (\frac{1}{2}, \frac{3}{4}, \frac{7}{8}, \dots, \frac{2^{20}-1}{2^{20}})$ .

Для проверки правильности вычислений  $Y'$  вычислить также её значение по заданной формуле без преобразований.

Вычисленные значения вывести с предшествующими порядковыми номерами и соответствующими значениями аргумента  $X$  в виде таблицы с заголовками столбцов. Для организации цикла использовать оператор `for`.

## 4.6. Сохранение результатов вычислений в массиве

*Массивами* называют структурные переменные, представляющие набор однотипных элементов – *элементов массива*. Доступ к элементам массива, называемым *индексными переменными*, осуществляется по имени массива и набору индексов, однозначно определяющих положение элемента в массиве. Если элементы массива сами не являются массивами, то такой массив называют *одномерным* и для обращения к элементам массива используется только один индекс, иначе массив называют многомерным (*двумерным, трехмерным* и так далее). В этом разделе ограничимся рассмотрением одномерных массивов.

В качестве индексов могут выступать выражения (в частности, константы и переменные) целого типа, которые записываются после имени массива в квадратных скобках через запятую. Минимальным значением индекса всегда является 0, соответственно максимальный индекс должен быть на единицу меньше размера массива.

Объявления массивов отличаются от объявлений простых переменных только тем, что после имен массивов в парах квадратных скобок записываются их размеры, а имена массивов являются указателями на область памяти, выделяемой для хранения значений его элементов. Например,

```
float x[3], z[3];
```

имена X и Z массивов будут указателями на области памяти, выделенные для хранения значений индексных переменных X[0], X[1], X[2] и Z[0], Z[1], Z[2].

Массивам, как и простым переменным, можно задавать начальные значения (что удобно при отладке программ), записав их списком в фигурных скобках через запятую, причем длина списка должна быть меньше или равна размеру массива. Например, в объявлении

```
double y[10]={ 1.2, 4.2, -5.1, 4.4, -1.5};
```

первые пять из десяти элементов массива будут иметь начальные значения:

```
y[0] = 1.2, y[1] = 4.2, y[2] = -5.1, y[3] = 4.4, y[4] = -1.5.
```

При решении задач, связанных с обработкой множеств значений в массивах или получаемых в результате вычисления значений функций, может потребоваться сохранение результатов вычислений для дальнейшего использования. В таких случаях необходимо выделить память для хранения результатов, объявив массив соответствующего типа, и очередное вычисленное значение сохранять в очередной ячейке массива, присваивая его индексной переменной - элементу этого массива с очередным значением индекса.

Размеры массива нельзя изменить во время работы программы, поэтому для обеспечения массовости алгоритма программы необходимо объявлять массивы с максимально возможными размерами, исходя из условий применения программы, обычно формулируемыми в задании на её разработку.

Свойство «массовость» алгоритма предполагает его применимость к различным, заранее оговоренным, наборам данных, в частности, задаваемых при вводе. В рассмотренном выше примере объявления массива  $Y$ , если он будет представлять исходные данные для какого-либо алгоритма, свойство «массовость» может обозначать, что могут обрабатываться любые наборы от одного числа до девяти чисел с любыми значениями, допустимыми для типа `double`.

В дальнейшем в заданиях на обработку массивов будут использоваться следующие сокращенные обозначения, которые рассмотрим на примерах:

$X(20)$  – будет обозначать, что для хранения данных должен использоваться одномерный массив, в котором подлежат обработке 20 последовательно расположенных элементов.

$X(N)$ ,  $N \leq 20$  – будет обозначать, что для хранения данных должен использоваться одномерный массив, в котором подлежат обработке первые  $N$  последовательно расположенных элементов.

Иногда при постановке задач удобно использовать слово *вектор* или *последовательность*, имея в виду размещаемые в последовательных ячейках массива данные.

Пример. Составить программу вычисления и сохранения в массиве  $Y$  значений функции  $y = \sin x$  и в массиве  $X$  - соответствующих значений аргумента. Аргумент должен изменяться в градусах с шагом  $dX$  от начального значения  $X_0$ .

Программа

```
#include "stdafx.h"
#include "math.h"
int _tmain(int argc, _TCHAR* argv[])
{
    const int n=10;
    double X[n], Y[n], X0, dX, Pi;
    int i;
    printf("Enter X0 :");
    scanf("%lf",&X0);
    printf("Enter dX :");
```



```

scanf ("%lf", &dx);
Pi=atan(1.0)*4.0;
for (i=0; i<n; i++)
{
    X[i]=X0;
    Y[i]=sin(X0*Pi/180);
    X0+=dx;
}
for (i=0; i<n; i++)
    printf ("%6.2lf %lf\n", X[i], Y[i]);

return 0;
}

```

#### 4.7. Пример выполнения задания

Составить программу для подсчёта и сохранения в массиве  $M(10)$  количеств значений целочисленного массива  $X(N)$ ,  $N \leq 500$ , попадающих в интервалы с номерами от 1 до 10 шириной  $h=(X_{\max}-X_{\min}+1)/10$ , где  $X_{\max}=50$  и  $X_{\min}=1$  – максимальное и минимальное значения в массиве  $X$  соответственно. Массив  $X$  заполнить случайными числами с равномерным распределением в диапазоне от  $X_{\min}$  до  $X_{\max}$ , используя стандартную функцию `rand`. Использовать также функцию `srand`, чтобы генерировать при каждом запуске программы новый набор случайных чисел, задавая в качестве аргумента функции `srand` новое значение - календарное время, установленное в системе (см. ниже комментарий в тексте программы).

Объявить  $X_{\min}$ ,  $X_{\max}$  и  $h$  в разделе констант. Полученные результаты использовать для вывода в виде гистограммы:

```

C:\Windows\system32\cmd.exe
Enter N:500
 1 -  5 63 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 6 - 10 55 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
11 - 15 60 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
16 - 20 52 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
21 - 25 48 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
26 - 30 38 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
31 - 35 39 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
36 - 40 50 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
41 - 45 54 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
46 - 50 41 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

MX = 23.927999
DX = 218.514801
Для продолжения нажмите любую клавишу . . .

```

Вычислить также для помещенных в массив X случайных чисел среднее значение

$$MX \text{ и дисперсию } DX \text{ по формулам: } MX = \frac{\sum_{i=1}^N X_i}{N} \text{ и } DX = \frac{\sum_{i=1}^N (X_i - MX)^2}{N}$$

Программа

```
#include "stdafx.h"
#include "time.h" //для функции time
#include "stdlib.h"//для функций srand и rand
int _tmain(int argc, _TCHAR* argv[])
{
    const unsigned Nmax=500, Xmax=50, Xmin=1, h=5;
    unsigned M[10], X[Nmax], t;
    unsigned i, j, u, N;//N - количество случайных чисел
    float MX, DX;
    //Ввод количества случайных чисел
    printf("Enter N:");
    scanf("%u", &N);

    //Получение числа секунд, прошедших
    t=time( NULL ); //с 1.1.1970 до текущего момента
    //задание начального значения
    srand(t);//генератора случайных чисел

    //Получение N случайных чисел в диапазоне
    //от Xmin до Xmax и сохранение в массиве X
    //(RAND_MAX - максимальное случайное число)
    for (i=0; i<N; i++)
        X[i]= (double)rand()/(RAND_MAX + 1)
            * (Xmax - Xmin+1)+ Xmin;

    //Накопление в ячейках массива M количеств
    //попаданий значений из массива X в интервалы
    //[0..4], [5..9], ..., [45..49]
    for (i=0; i<10; i++) M[i]=0;
    for (i=0; i<N; i++)
```

```

        //Увеличить на 1 значение элемента массива M
        M[(X[i]-Xmin)/h]++; //с индексом (X[i]-Xmin)/h

//Вывод результатов в виде гистограммы
for (i=0;i<10;i++)
{
    printf("%3d - %3d%3d ",i*h+1,(i+1)*h,M[i]);
    for (j=0;j<M[i]; j++)
        printf("X");
    printf("\n");
}

//Вычисление и вывод среднего значения
MX=0;
for (i=0; i<N; i++)
    MX=MX+X[i];
MX=(float)MX/N;
printf("\nMX = %lf\n",MX);

//Вычисление и вывод дисперсии
DX=0;
for (i=0; i<N; i++)
    DX=DX+(X[i]-MX)*(X[i]-MX);
DX=(float)DX/N;
printf("DX = %lf\n",DX);

return 0;
}

```

#### 4.8. Задания для самостоятельной работы

Во всех заданиях использовать только простые циклы.

**10.** В массиве M(5) хранятся в порядке возрастания значения 1, 5, 10, 50, 100. Требуется найти для положительного целого числа N и сохранить в массиве K(5) коэффициенты разложения  $N = K_1 \cdot M_1 + K_2 \cdot M_2 + K_3 \cdot M_3 + K_4 \cdot M_4 + K_5 \cdot M_5$ , при котором сумма

$\sum_{i=1}^5 K_i$  будет минимальна (использовать операции % и /).

**11.** В целочисленном массиве  $M(N)$ ,  $N \leq 20$ , содержатся разные числа от 1 до  $k$ ,  $k < N$ , а в массиве  $S(k)$  - не повторяющиеся числа от 1 до  $k$  в произвольном порядке. Требуется зашифровать данные массива  $M$  следующим образом: новым значением элемента массива  $M$  будет значение элемента массива  $S$ , индекс которого равен значению этого элемента массива  $M$ . Затем расшифровать  $i$ -тое значение массива  $M$  и присвоить результат переменной  $P$ .

**12.** Выполнить циклический сдвиг элементов массива  $X(N)$ ,  $N \leq 10$ , в результате которого значение последнего элемента должно оказаться на месте первого, а остальные – сдвинутыми на одну позицию в сторону увеличения индекса.

**13.** На заданном отрезке, с заданным шагом изменения аргумента вычислить и поместить в массив  $F$  30 значений функции  $e^{-x} \sin(6x)$ , делённые на её последнее положительное значение.

**14.**  $S$  является последовательностью нулей и единиц длиной  $L \leq 30$ . Требуется сохранить в массиве  $Y$  информацию, представленную  $S$ , в виде:  $Y_0 = S_1$ , а далее – числа, представляющие длины локальных подпоследовательностей с одинаковыми значениями. Подсчитать количество записанных в массив  $Y$  чисел.

**15.** Восстановить последовательность  $S$  (см. предыдущий пункт задания) по данным из массива  $Y$  и количеству записанных в массив  $Y$  чисел.

**16.** Последовательность  $S$  из нулей и единиц длиной  $L < 30$  зашифровать и поместить в массив  $D$ . Шифровать по следующему правилу: положить  $D_1 = S_1$ , а далее  $D_i = 1$ , если  $S_i = S_{i-1}$ , иначе – 0. Затем по данным из  $D$  расшифровать последовательность и поместить в массив  $R$ .

**17.** В массиве  $X(4)$  хранятся в порядке возрастания значений положительные вещественные числа. Требуется найти и сохранить в целочисленном массиве  $K(4)$  коэффициенты разложения переменной  $R$ :  $R = D + K_0 \cdot X_0 + K_1 \cdot X_1 + K_2 \cdot X_2 + K_3 \cdot X_3$ , где  $D < X_0$ , при котором сумма  $\sum_{i=1}^4 k_i$  будет минимальна.

**18.** Из массива  $X(N)$ ,  $N \leq 20$ , упорядоченного по невозрастанию значений элементов, переписать в массив  $Y$  без повторов значения элементов с четными индексами, меньшие  $S$ , сохранив упорядоченность

**19.** Изменяя  $X$  от заданного начального значения с заданным шагом  $H$  вычислить и поместить в массив  $F$  20 значений разности функции  $e^{-X} \sin(6X)$  и её значением в точке первого локального минимума.

**20.** В массиве  $V(10)$ , заданном начальными значениями, содержатся разные числа от 0 до 9 в произвольном порядке. Требуется поместить в массив  $D$  зашифрованную произвольную последовательность  $S$  длины  $L \leq 30$  из целых чисел от 0 до 9. Шифрование выполнить по следующему правилу:  $D_i = i - V_{S_i}$ . Затем по данным из  $D$  расшифровать  $k$ -тую цифру и поместить в  $R$ .

**21.** Найти и сохранить в массиве  $N$  коэффициенты  $p_0, p_1, p_2, p_3, p_4, p_5$  разложения целого числа  $K$  ( $0 < K < 10^6$ ) по степеням числа 10.

**22.** Выполнить циклический сдвиг элементов массива  $X(N)$ ,  $N \leq 20$ , на  $K$  позиций, в результате которого последние  $K$  элементов займут место в начале массива, а остальные будут сдвинуты на  $K$  позиций в сторону увеличения индекса. Использовать дополнительный массив  $D$ .

**23.** На заданном отрезке, с заданным шагом изменения аргумента вычислить и поместить в массив  $X(20)$  значения аргумента функции  $e^{-x} \sin(3x) - 0,2$ , предшествующие изменению знака функции, и подсчитать их количество. Вычисления проводить либо до достижения границы интервала, либо до заполнения массива.

**24.** В массив  $X(N)$ ,  $N \leq 20$ , упорядоченный по возрастанию значений элементов, добавить новое число так, чтобы не нарушить упорядоченность.

**25.**  $S$  является последовательностью из чисел 1, 2, 3 и 4 длины  $L \leq 20$ . Требуется сохранить в массивах  $K$  и  $N$  информацию, представленную  $S$ , в виде:  $K_i$  – число из  $i$ -той подпоследовательности из одинаковых чисел в  $S$ ,  $N_i$  – длина этой подпоследовательности, а также количество записанных в массивы  $K$  и  $N$  чисел.

**26.** Из массива  $X$ , упорядоченного по невозрастанию значений элементов, переписать в массив  $Y$  числа, исключив их повторы и обеспечив упорядоченность по возрастанию.

**27.** Поместить положительные элементы массива  $X$  в начало массива  $Y$ , а следом – его отрицательные элементы.

**28.** Из целочисленного массива  $X(N)$ ,  $N \leq 20$ , удалить числа, кратные  $K$ , поместив остальные числа в его начале без пропусков, не изменив их взаимного расположения. Вывести количество оставленных в массиве чисел и эти числа.

**29.** Найти и сохранить в массиве  $K(N)$ ,  $N \leq 14$ , старшие  $N$  цифр правильной дроби  $R$  при представлении её в десятичной системе счисления, а в переменной  $D$  – часть числа  $R$ , меньшую  $10^{-N}$ . Использовать стандартные функции `floor` и `ceil`.

**30.** На заданном отрезке, с заданным шагом изменения аргумента вычислить и поместить в массив  $X(50)$  значения аргумента функции  $e^{-3x} \sin^2 20x$ , предшествующие первому локальному экстремуму функции типа максимум, а в массив  $Y$  – соответствующие значения функции. Если за 50 шагов экстремум не будет найден, то вывести соответствующее сообщение, иначе вывести помещенные в массивы  $X$  и  $Y$  значения в виде таблицы.

**31.** Из массива  $X(20)$ , упорядоченного по неубыванию значений элементов, переписать в массив  $Y$  числа, исключив их повторы и добавив новое вводимое значение  $R$  так, чтобы не нарушить упорядоченность.

**32.** На заданном отрезке, с заданным шагом изменения аргумента вычислить и поместить в массив  $X(12)$  значения аргумента функции  $e^{-x/3} \sin^2 5x$ , непосредственно предшествующие локальным максимальным приращениям функции. Если до достижения верхней границы интервала массив окажется заполненным, то вычисления прекратить и сопроводить вывод результатов соответствующим сообщением.

**33.** Поместить элементы массива  $X$  в начало массива  $Y$  в обратном порядке, исключив элементы, превосходящие по абсолютной величине вводимое значение  $R$ .

**34.** В массиве  $K$  с индексами от 0 до 9, заданном начальными значениями, содержатся разные числа от 0 до 9 в произвольном порядке. Требуется поместить в массив  $Y$  зашифрованную произвольную последовательность  $X$  длины  $L \leq 30$  из целых чисел от 0 до 9. Шифрование выполнить по следующему правилу:  $Y_i = i - K_{X_i}$ . Затем по данным из  $Y$  расшифровать последовательность и поместить в массив  $P$ . Использовать дополнительный массив  $T$  с начальными значениями, заданными следующим образом:  $T_i$  равно номеру ячейки массива  $K$  со значением  $i$ .

## Вопросы по самопроверке

1. Какие операторы используются в программах циклической структуры?
2. Как устроен (назначение составных частей) и как работает оператор цикла `for`?
3. Можно ли в качестве параметра цикла `for` использовать вещественную переменную?
4. Как устроен и как работает оператор цикла `while`?
5. Как устроен и как работает оператор цикла `do while`?
6. Как вычислить округлённое вещественное значение?
7. Как дать объявление одномерного массива?

8. Какой индекс имеет первый элемент массива?

## Список рекомендуемой литературы

1. Керниган Б. И., Ритчи Д. М. Язык программирования С, 2-е издание,: Пер. с англ. – М.:Издат. дом «Вильямс» , 2006. – 304 с.: ил.
2. Лафоре Р. Объектно-ориентированное программирование в С++. Классика Computer Science. 4-е изд. – СПб.: Питер, 2008. -928 с.: ил.
3. Павловская Т.А. С/С++. Программирование на языке высокого уровня. – СПб:Питер, 2007.-461 с.: ил.
4. Пахомов Б.И. С/С++ и MS Visual С++ 2008 для начинающих. – СПб: БХВ-Петербург, 2008.- 624 с.: ил.