Тема 3. Обзор операций и базовых инструкций языка Си

3.1 Общая структура программы на языке Си
3.2 Функция main
3.3 Стандартные математические функции
3.4 Операторы и выражения
3.5 Арифметические операторы
3.6 Операции присваивания
3.7 Понятие R-value и L-value
3.8 Операторы инкремента и декремента
3.9 Приведение типов в выражениях
3.10 Приведение типа при выполнении присваивания
3.11 Операции отношения
3.12 Логические операции
3.13 Операция запятая

3.14 Ввод-вывод данных
3.14.1 Функции вывода
3.14.2 Функции ввода
3.14.3 Пример, реализующий линейный алгоритм

- **Программа, написанная на языке Си, состоит из одной или нескольких функций, причем одна функция обязательна имеет идентификатор (имя) main()**

- **Назначение функции main() – управление работой всей программы.**

- **Функции main(), как правило, не имеет параметров и не возвращает результат (наличие круглых скобок обязательно).**

3.1 Общая структура программы на языке Си имеет вид:

<директивы препроцессора>

<определение типов пользователя - typedef>

<описание прототипов функций>

<определение глобальных переменных>

< функции>

Функции имеют следующую структуру:

```
<класс памяти> <тип> < ID функции>
(<объявление параметров>)
  { //начало функции

        <код функции>

    } //конец функции
```

3.2 Функция main

```
void main ()
{
Тело программы
}
```

1. **В языке С нет стандартных инструкций (операторов) для вывода сообщений на консоль (окно пользователя).**

2. **В языке С предусматриваются специальные библиотечные файлы, в которых имеются функции для этих целей.**

- **Заголовочный файл с именем `stdio.h` (стандартный ввод–вывод), который должен быть включен в начало программы.**
- **Для вывода сообщения на консоль используется функция `printf()`.**
- **Для работы с консолью включен заголовочный файл `conio.h`., который поддерживает функцию `_getch()`, извлекающую символ из потока ввода, т. е. она предназначенная для приема сообщения о нажатии какой-либо (почти любой) клавиши на клавиатуре.**
- **С другими компиляторами, возможно, потребуется `getch()`, т.е. без префиксного нижнего подчеркивания.**

Структура простой программы

```
#include <stdio.h> //содержит стандартные
   //функции файлового ввода-вывода
void main()
{  //Начало функции main
  printf ("Это работает функция main! ");
  }  //Окончание функции main
```

return 0;

- указывает на то, что выполнение функции main() закончено и что в систему возвращается значение 0 (целое число).

- Нуль используется в соответствии с соглашением об индикации успешного завершения программы

- **Препроцессором** называется программа, которая вызывается компилятором и обрабатывает исходную программу перед ее компиляцией.
- **Препроцессор** различает специальные команды – директивы.
- **Препроцессор** обеспечивает подключение текстов исходных модулей, формирование макроопределений, планирование условной компиляции.

Препроцессорные директивы начинаются с символа #, за которым следует наименование директивы, указывающее текущую операцию препроцессора.

#include < ID_файла.h>

где h – расширение заголовочных файлов.

stdio.h – содержит стандартные функции файлового ввода-вывода;

conio.h – функции для работы с консолью (клавиатура, экран монитора);

math.h – математические функции.

#include <stdio.h>

Вставляет в программу текст из файла stdio.h, который находится в стандартных каталогах (указанных компилятору, т.е. заданных в среде разработки).

#include "ID_файла.h"

где ID_файла символьная строка, определяющая путь к файлу.

#include "My_file.h"

Вставляет в программу текст из файла My_file.h, который находится в текущем каталоге проекта.

#include "C:\ My_file.h"

Подключает в программу текст из файла My_file.h, который находится в заданном каталоге.

3.3 Стандартные математические функции

Математическая функция	*ID* функции в языке Си
$\sqrt{x}$	sqrt(x)
$\lvert x \rvert$	fabs(x)
e^x	exp(x)
x^y	pow(x,y)
ln(x)	log(x)
$lg_{10}(x)$	log10(x)
sin(x)	sin(x)
cos(x)	cos(x)
tg(x)	tan(x)
arcsin(x)	asin(x)
arccos(x)	acos(x)
arctg(x)	atan(x)
arctg(x / y)	atan2(x)
sh(x)=0.5 (e^x–e^{-x})	sinh(x)
ch(x)=0.5 (e^x+e^{-x})	cosh(x)
tgh(x)	tanh(x)
остаток от деления x на y	fmod(x,y)
наименьшее целое >=x	ceil(x)
наибольшее целое <=x	floor(x)

Второе основное назначение препроцессора – обработка макроопределений.

Макроподстановка имеет общий вид:

#define < ID > <строка>

Например: #define PI 3.1415927

В ходе препроцессорной обработки программы появление в тексте идентификатора PI везде заменяется значением 3.1415927.

3.4 Операторы и выражения

Выражение – это объект, состоящий из констант, идентификаторов переменных, обращений к функциям и операций, указывающих, что надо делать над объектами в этом выражении.

По количеству участвующих в них операндов различают операции:

- **унарные** (с одним операндом);
- **бинарные** (с двумя операндами);
- **тернарные** (три операнда – условная операция).

Существуют следующие виды операторов (операций):

- **арифметические операции;**
- **логические операции;**
- **операции отношений;**
- **операции присваивания;**
- **побитовые операции;**
- **операции адресации/разадресации.**

3.5 Арифметические операторы (операции)

Знак операции	Наименование	Пример
+	сложение	a=b+c; если b=6, c=5, то a=11
-	вычитание	a=b-c; если b=6, c=5, то a=1
-	арифметическое отрицание	a=-b; если b=6, то a=-6
*	умножение	a=b*c; если b=6, c=5, то a=30
/	деление	a=b/c; если b=10, c=5, то a=2
%	остаток от деления	a=b%c; если b=6, c=5, то a=1

3.6 Операции присваивания

v=e;

v – только L-value,
т.е. слева переменная, под которую компилятор обязательно выделил область памяти (именованная либо косвенно адресуемая указателем переменная).

3.7 Понятие R-value и L-value

Данные в Си могут представляться литералами и переменными (и те и другие хранят значения данных в областях памяти). Это хранимое значение называется R-value (read value), т.е. значение, которое можно прочитать.

Значение адреса области памяти для хранения значения переменной называется L-value (location value), т.е. значение, которое определяет местоположение.

Фундаментальное различие между литералами и переменными заключается в том, что переменная именует область памяти, в которой хранится R-value, а литерал – нет.

Переменная определяет как R-value так и L-value, а литерал только R-value.

Присваивание может включать несколько операций присваивания

```
int i, j, k;
float x, y, z;

 y=10;      //y=10
 i=j=k=0; //k=0, j=k, i=j
 --------
 x=i+(y=3)-(z=0); //z=0, y=3, x=i+y-z;
```

Примеры недопустимых выражений:

а) присваивание константе: `2 = x+y;`

б) присваивание функции: `getch() = i;`

в) присваивание результату операции: `(i+1) = 2+y;`

В языке Си выражение вида v=v#e;

можно записать в другой форме:

v#=e; где # – арифметическая операция

Например:

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

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

3.8 Операторы инкремента и декремента

Унарные операции инкремента (++) и декремента (--) предназначены соответственно для увеличения и уменьшения значения операнда на единицу.

x=x#1; эквивалентно записи ##x;
(префиксная)
x##;
(постфиксная),

x=x+1; ++x; x++;
x=x-1; --x; x--;

Если эти операции используются в чистом виде, то различий между префиксной и постфиксной формами нет.

- **Если унарные операции инкремента и декремента используются в выражении, то в префиксной форме (##x) сначала значение x изменится на единицу, а затем будет использовано в выражении.**

- **В постфиксной (x##) – сначала значение используется в выражении, а затем изменяется на единицу.**

Примеры использования сокращений

```
int i, j, k;
float x,y;
```

```
x*= y;          ↔    x = x*y;
i+= 2;          ↔    i = i+2;
x/= y+15;       ↔    x=x/(y+15);
k--;            ↔  k = k-1;
--k;            ↔  k = k-1;
j = i++;        ↔  j = i;    i = i+1;
j = ++i;        ↔  i = i+1;    j = i;
```

Примеры использования сокращений

```
int n, a, b, c, d;
n = 2;    a = b = c = 0;

    a = ++n;      //  n=3, a=3
    a+ = 2;       //  a=5
    b = n++;      //  b=3, n=4
    b- = 2;       //  b=1
    c = --n;      //  n=3, c=3
    c* = 2;       //  c=6
    d = n--;      //  d=3, n=2
    d% = 2;       //  d=1
```

3.9 Приведение типов в выражениях

Правила преобразования арифметических операндов:

В выражении у операндов могут быть разные типы, тогда компилятор приводит все операнды в один тип в сторону увеличения байтов:

- **операнды char и short преобразуются к int;**
- **операнды float преобразуются к double;**
- **если один из операндов double, то второй операнд преобразуется к double, и результат будет double;**
- **если один из операндов long, то второй операнд преобразуется к long, и результат будет long;**
- **если один из операндов имеет тип unsigned то и другие преобразуются к unsigned;**

- **Результат деления целочисленных операндов:**
  1/3 =0

- **Чтобы избежать ошибок необходимо явно изменять тип хотя бы одного операнда, т.е. записывать 1./3 ~ 0.333**

- **Явное преобразование типов осуществляется, если перед выражением поставить в скобках идентификатор соответствующего типа. Вид записи операции: (тип) выражение;**
  (float)1/3 ~ 0.333

3.10 Приведение типа при выполнении присваивания

При присваивании значение правой части преобразуется к типу левой, который и является типом результата.

Рассмотрим преобразование int в char:

char s; int j=327; s=j; //s=‘G’ или 71

$327_{10} = 0x147_{16} = 0001\ 0100\ 0111_2$

$0100\ 0111_2 = 0x47 = 71_{10}$

Старший байт

```
float x; int i;
x=i; // тип результата float
i=x; // тип результата int
```

Тип float преобразуется к int с отбрасыванием дробной части.

Тип double преобразуется к float с округлением.

3.11 Операции отношения

<выражение1><знак операции><выражение2>

Знак операции	Наименование	Пример
==	Сравнение на равенство	a == b; Вырабатывает 1,если a равно b, и 0 - в противном случае
>	Больше	a > b; Вырабатывает 1, если a больше b, и 0 - в противном случае
>=	Больше или равно	a >= b; Вырабатывает 1, если a больше или равно b, и 0 - в противном случае
<	Меньше	a < b; Вырабатывает 1,если a меньше b, и 0 - в противном случае
<=	Меньше или равно	a <= b, Вырабатывает 1, если a меньше или равно b, и 0 - в противном случае
!=	Не равно	a != b; Вырабатывает 1, если a не равно b, и 0 - в противном случае

3.12 Логические операции

Знак операции	Наименование	Пример
&&	Логическое "И" (конъюнкция)	a= b && c; если b и c не равны нулю, то a=1, в противном случае a=0
||	Логическое "ИЛИ" (дизъюнкция)	a= b || c; если b и c равны нулю, то a=0, в противном случае a=1
!	Логическое отрицание (инверсия)	a= ! b; если b равно нулю, то a=1, если b не равно нулю, то a=0

!0↔1 !1 ↔0 !5 ↔0

Выражение $1 \le x \le 2$ в СИ имеет вид:

((x>=1)&&(x<=2))

Выражение принимает значение истина, если 1-е и 2-е выражения истинны.

Особенность операций конъюнкции и дизъюнкции – экономное последовательное вычисление выражений-операндов: <выражение1> <операция><выражение2>

- **если выражение1 операции конъюнкция ложно, то результат операции – ноль и выражение2 не вычисляется;**

- **если выражение1 операции дизъюнкция истинно, то результат операции – единица и выражение2 не вычисляется.**

3.13 Операция , (запятая)

Данная операция используется при организации строго гарантированной последовательности вычисления выражений.

Форма записи:

выражение1, …, выражениеN;

выражения1,…,N вычисляются гарантированно последовательно и результатом операции становится значение выражения N.

Пример:

m=(i=1, j=i++, k=6, n=i+j+k);

получим последовательность вычислений: i=1, j=i=1, i=2, k=6, n=2+1+6, в результате m=n=9.

Рассмотрим пример:

```
int x=1, y=2, z;
z=(x++, x-y); //z=0
z=x++, x-y;   /* z=1 т.к. оператор "," имеет
наименьший приоритет из всех
операторов */
```

3.14 Ввод-вывод данных
3.14.1 Функции вывода
Функция форматного вывода на экран printf

Функция форматного вывода на экран printf описывается:

printf ("управляющая строка", a_1, a_2, …, a_n);

Управляющая строка содержит объекты трех типов:

- **поясняющий текст;**
- **список модификаторов форматов;**
- **управляющие символы.**

Количество и порядок следования форматов должен совпадать с количеством и порядком следования печатаемых объектов.

Основные модификаторы формата:

%d – десятичное целое число;
%c – один символ;
%s – строка символов;
%f – число с плавающей точкой, десятичная запись;
%lf – число с плавающей точкой, удвоенной точности;
%e – число с плавающей точкой в экспоненциальной форме;
%p – указатель (адрес);
%o – восьмеричное целое число;
%x – шестнадцатеричное целое число.

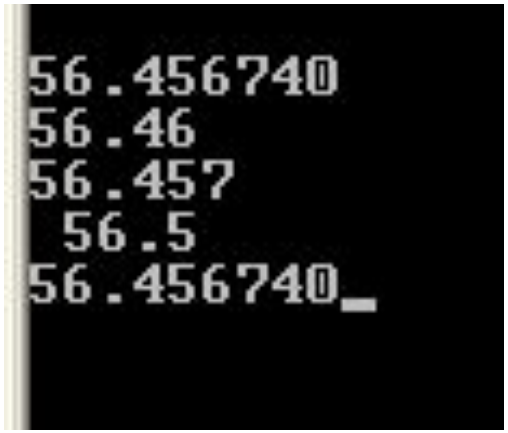
```
printf("INT – %d; DOUBLE – %lf; CHAR – %c", 5,
    4.35, 'a' );
```

Результат:

```
INT – 5; DOUBLE – 4.350000; CHAR – a
```

В модификаторах формата функции printf() после символа % можно указывать строку цифр, задающую минимальную ширину поля вывода, например: %5d (для целых), %4.2f (две цифры после запятой для поля, шириной 4 символа).

```
double u=56.45674;

printf("\n%lf", u);
printf("\n%4.2lf", u);
printf("\n%4.3lf", u);
printf("\n%5.1lf", u);
printf("\n%6lf", u);
```



Аргументами функции printf() могут быть переменные, константы, выражения, вызовы функции. Главное, чтобы их значения соответствовали заданной спецификации.

Управляют выводом специальные последовательности.

Например,
\n – новая строка
\t – горизонтальная табуляция

Если нужно напечатать сам символ %, то его нужно указать 2 раза.

```
printf("Только %d%% предприятий не работало. \n",5);
```

Получим: Только 5% предприятий не работало.

Можно использовать функцию printf() для нахождения кода ASCII некоторого символа.

```
printf("%c – %d\n", 'a', 'a');
```

Получим изображение и десятичный ASCII код символа 'a': a – 97

```
#include<stdio.h>
#include<conio.h>
void main() {
float f=125.0;
//Надежда на интеллект компилятора
printf("\n??? %f  %d  %x  %f", f, f, f, f);
//На компилятор не надейся.
//Приводи тип в соответствие
printf("\n*** %f  %d  %x  %f", f, (int)f,
(int)f, f);
getch();
                }
```

```
??? 125.000000  0  405f4000  125.000000
*** 125.000000  125  7d  125.000000_
```

Для вывода данных могут использоваться также функцию **puts**, которая выводит на экран дисплея строку символов, автоматически добавляя к ней символ перехода на начало новой строки.

```
puts("Печатаемая строка.");
```

Функция **putchar** – выводит на экран дисплея один символ без добавления символа '\n '.

```
 putchar('A');
 putchar(65);
```

3.14.2 Функции ввода

Функция форматированного ввода scanf описывается:

scanf ("управляющая строка", a_1, a_2, …, a_n);

Управляющая строка содержит только модификаторы форматов.

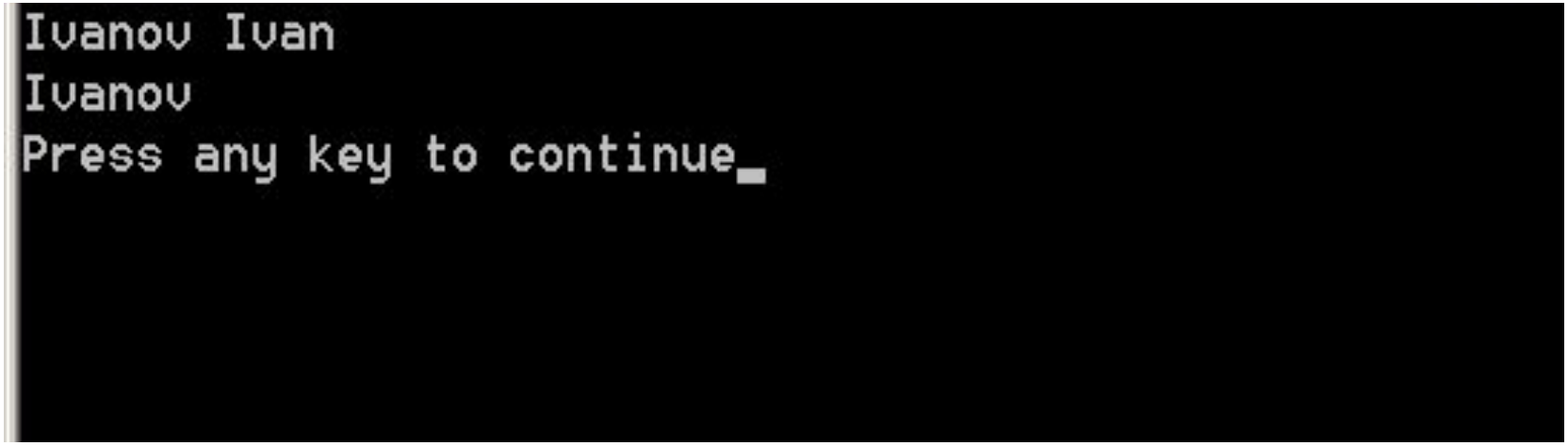
Аргументы (a_1, a_2, …, a_n) – адреса переменных, разделенные запятыми (перед именем переменной записывается символ &).

Исключение – строки символов.

```
int course;
float grant;
char name[20]; //строка символов
printf("Укажите курс, стипендию, имя \n");
scanf("%d%f%s", &course, &grant,  name);
```

Функция scanf() ввод строки по формату %s производит только до первого пробела.

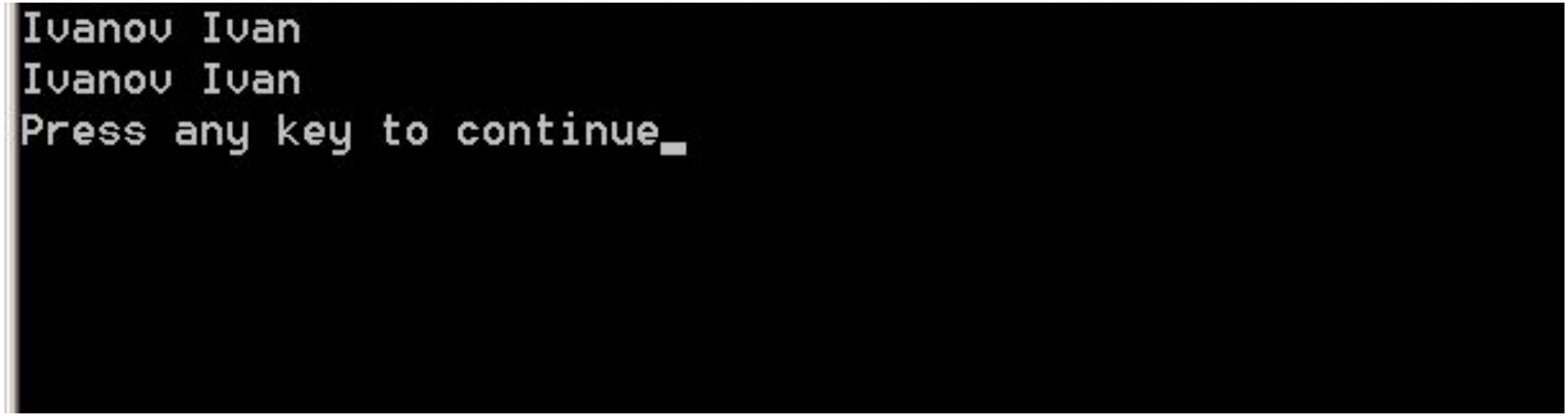
```
char name[20];
.....................
scanf("%s",  name);
puts(name);
```



Для ввода фраз, состоящих из слов используется функция:

gets(<ID строковой переменной>);

```
char name[20];
.....................
gets(name); puts(name);
```



Стандартные потоки

Каждой программе предоставляются три стандартных потока, которые по умолчанию соединены с консолью.

- **stdout – предназначен для вывода данных на консоль (с этим потоком используются функции printf, puts, putchar);**
- **stdin – предназначен для ввода данных с консоли (scanf, gets, getchar);**
- **stderr – предназначен для вывода сообщений об ошибках на консоль. Для работы с потоком используется функция void perror(const char* str), параметр str указывает на строку, содержащую сообщение об ошибке.**

- **Функция scanf("%d", &n); при вводе числа обращается к буферу ввода stdin, создаваемому ОС.**
- **Если при выполнении функции scanf("%d", &n); ввести не цифру, а букву (строку), то scanf не извлечет ее из буфера. Программа не выдаст сообщений, но в дальнейшем это может привести к трудно диагностируемым ошибкам в работе.**

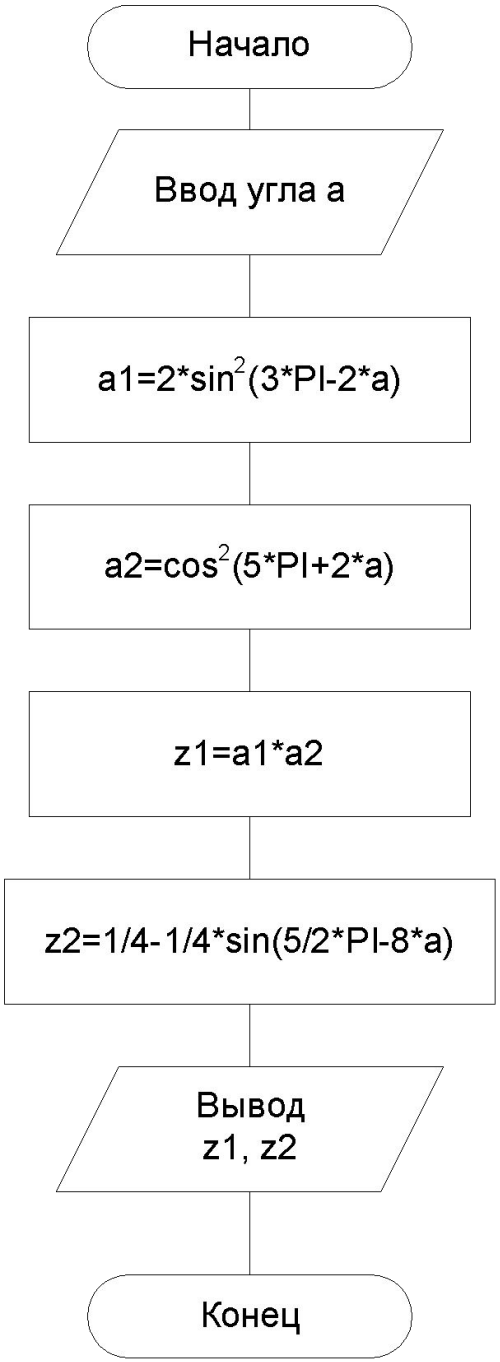
- **Явная очистка буфера ОС: fflush(stdin);**

3.14.3 Пример, реализующий линейный алгоритм

Составить программу для расчета значений *z*1 и *z*2 (результаты должны совпадать).

$$z_1 = 2\sin^2(3\pi - 2\alpha)\cos^2(5\pi + 2\alpha), \quad z_2 = \frac{1}{4} - \frac{1}{4}\sin\left(\frac{5}{2}\pi - 8\alpha\right)$$

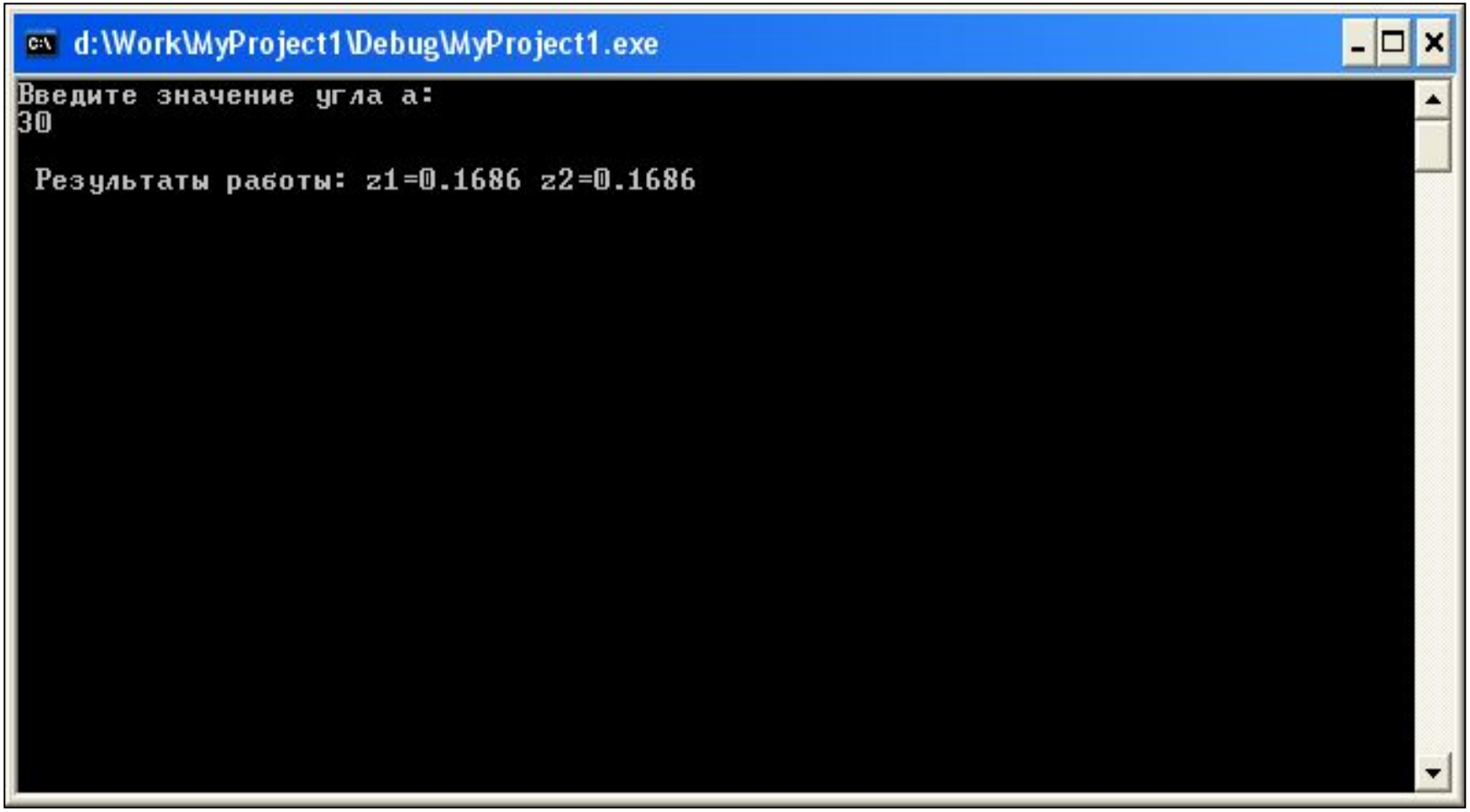
Схема программы

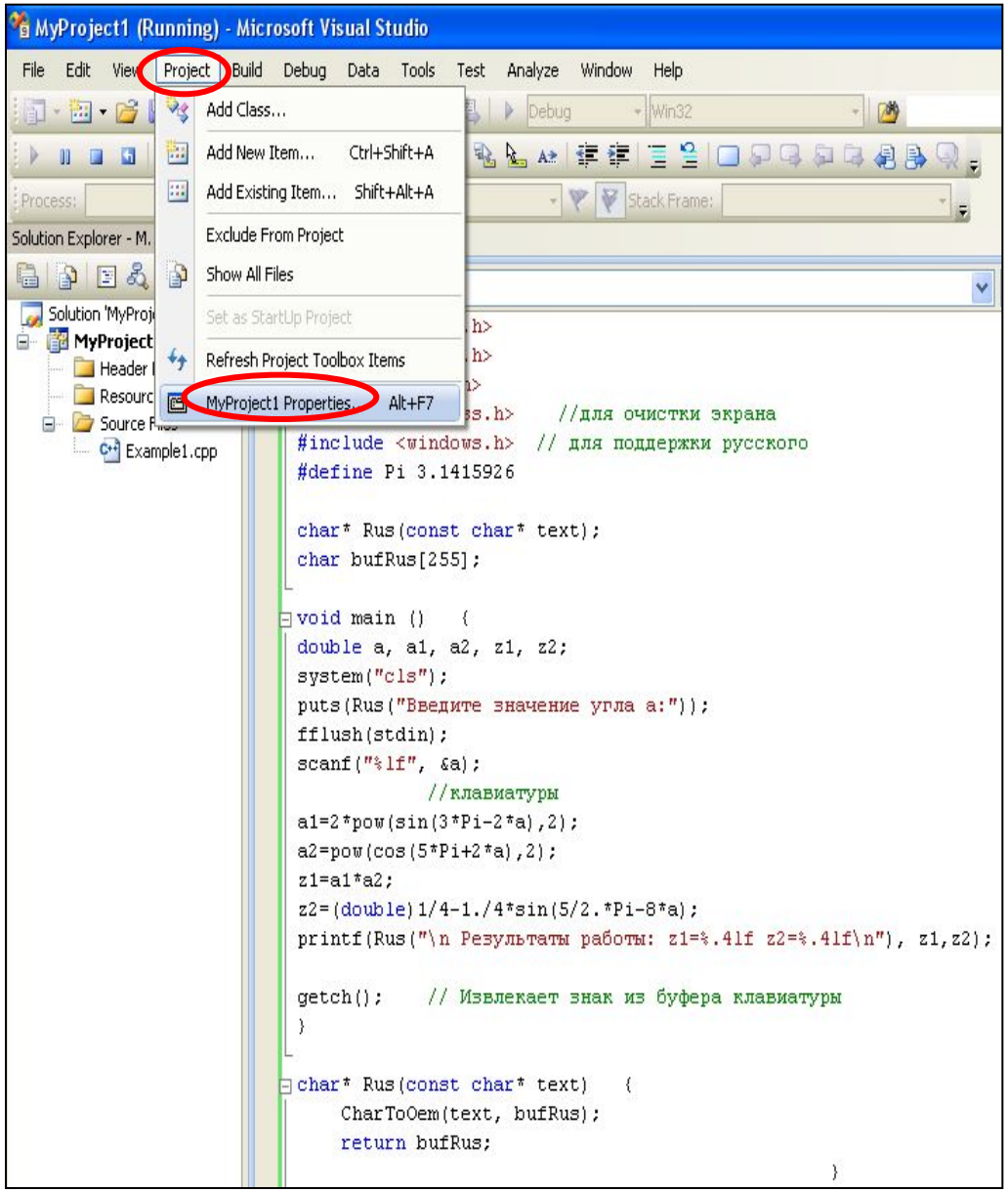


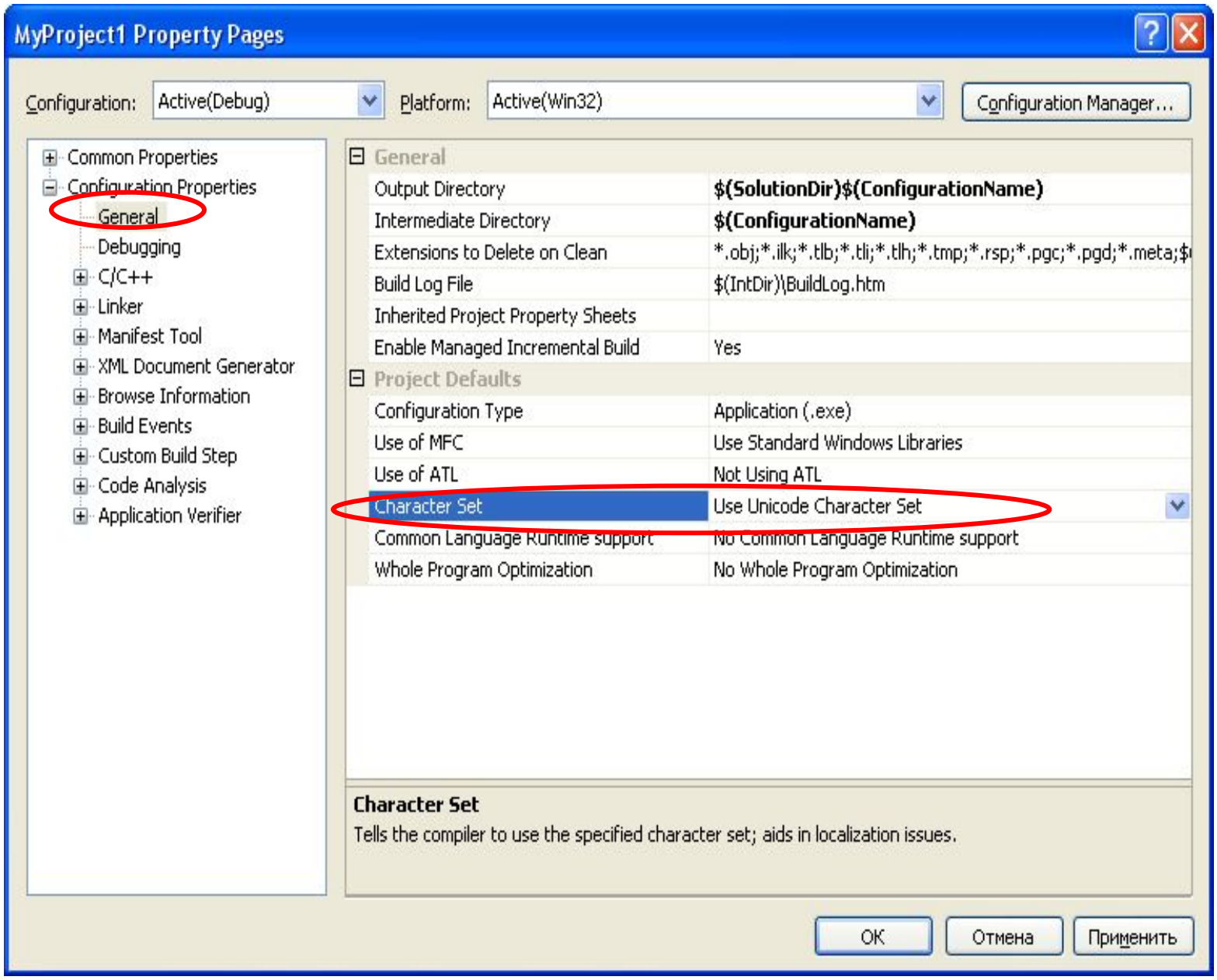
Математическая функция	ID функции в языке Си
$\sqrt{x}$	sqrt(x)
|x|	fabs(x)
e^x	exp(x)
x^y	pow(x,y)
ln(x)	log(x)
$lg_{10}(x)$	log10(x)
sin(x)	sin(x)
cos(x)	cos(x)
tg(x)	tan(x)
arcsin(x)	asin(x)
arccos(x)	acos(x)
arctg(x)	atan(x)
остаток от деления *x* на *y*	fmod(x,y)

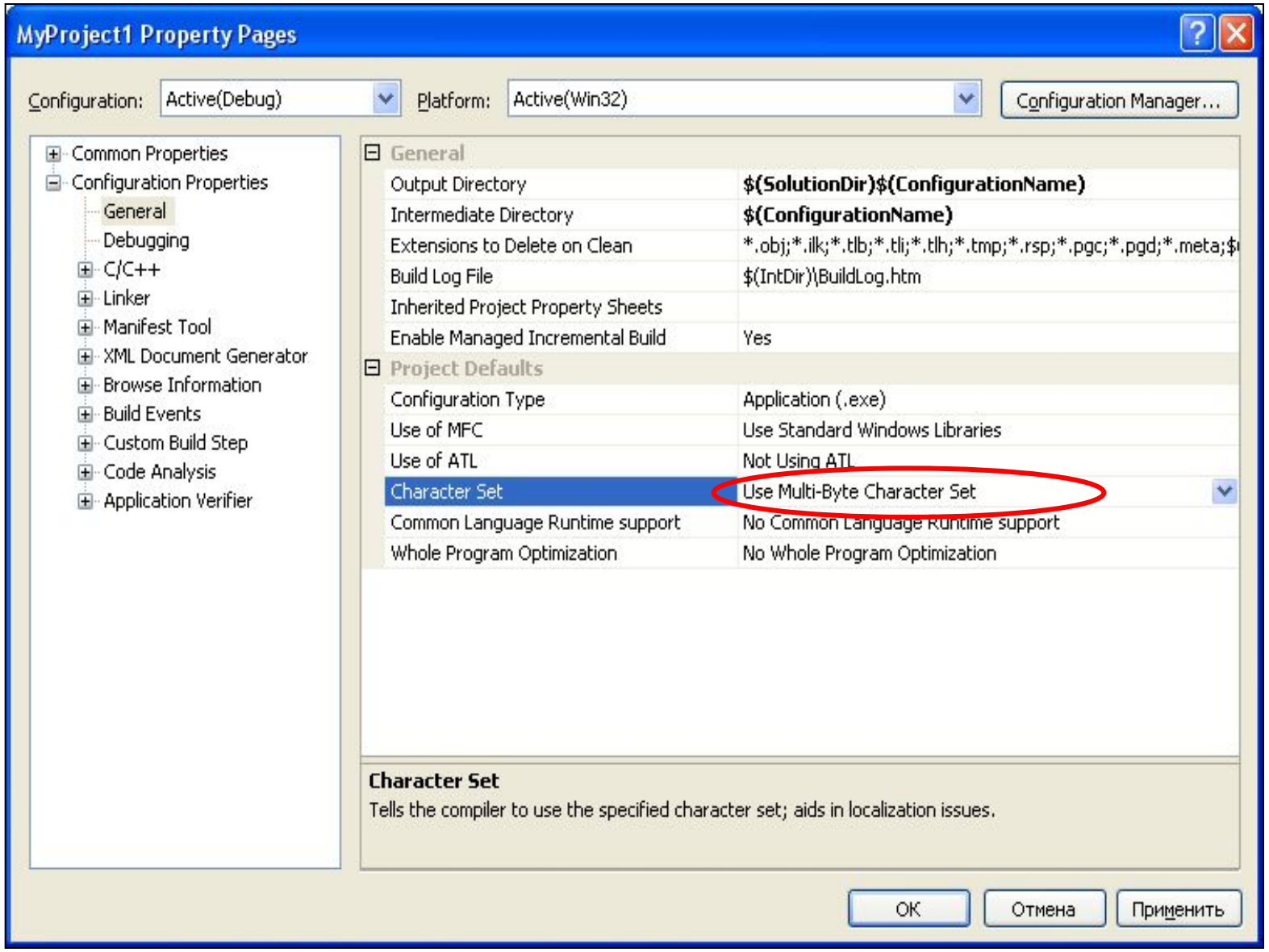
```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <process.h>    //для очистки экрана
#include <locale.h>      // подключение setlocale

/*
char *setlocale(int category, const char *locale);
Используется для установления или определения
текущей локали программы. Аргумент category
определяет, какая часть текущей локали программы
подлежит изменению.
LC_CTYPE – преобразование символов
*/
```

```
#define Pi 3.1415926

void main ()   {
//Установка поддержки русского языка
setlocale(LC_CTYPE, "Russian");
double a, a1, a2, z1, z2;
system("cls");
puts("Введите значение угла a:");
fflush(stdin);
scanf_s("%lf", &a);
a1=2*pow(sin(3*Pi-2*a),2);
a2=pow(cos(5*Pi+2*a),2);
z1=a1*a2;
z2=(double)1/4-1./4*sin(5./2*Pi-8*a);
printf("\n Результаты работы: z1=%.4lf z2=%.4lf\n",
    z1,z2);
_getch(); }
```


d:\Work\MyProject1\Debug\MyProject1.exe
Введите значение угла a:
30
Результаты работы: z1=0.1686 z2=0.1686


MyProject1 (Running) - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Test Analyze Window Help
Add Class...
Add New Item... Ctrl+Shift+A
Add Existing Item... Shift+Alt+A
Exclude From Project
Show All Files
Set as StartUp Project
Refresh Project Toolbox Items
MyProject1 Properties... Alt+F7
Solution Explorer - M.
Example1.cpp
#include <windows.h> // для поддержки русского
#define Pi 3.1415926
char* Rus(const char* text);
char bufRus[255];
void main () {
double a, a1, a2, z1, z2;
system("cls");
puts(Rus("Введите значение угла a:"));
fflush(stdin);
scanf("%lf", &a);
//клавиатуры
a1=2*pow(sin(3*Pi-2*a),2);
a2=pow(cos(5*Pi+2*a),2);
z1=a1*a2;
z2=(double)1/4-1./4*sin(5/2.*Pi-8*a);
printf(Rus("\n Результаты работы: z1=%.4lf z2=%.4lf\n"), z1,z2);
getch(); // Извлекает знак из буфера клавиатуры
}
char* Rus(const char* text) {
CharToOem(text, bufRus);
return bufRus;
}


MyProject1 Property Pages
Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...
Common Properties
Configuration Properties
General
Debugging
C/C++
Linker
Manifest Tool
XML Document Generator
Browse Information
Build Events
Custom Build Step
Code Analysis
Application Verifier
General
Output Directory $(SolutionDir)$(ConfigurationName)
Intermediate Directory $(ConfigurationName)
Extensions to Delete on Clean *.obj;*.ilk;*.tlb;*.tli;*.tlh;*.tmp;*.rsp;*.pgc;*.pgd;*.meta;$
Build Log File $(IntDir)\BuildLog.htm
Inherited Project Property Sheets
Enable Managed Incremental Build Yes
Project Defaults
Configuration Type Application (.exe)
Use of MFC Use Standard Windows Libraries
Use of ATL Not Using ATL
Character Set Use Unicode Character Set
Common Language Runtime support No Common Language Runtime support
Whole Program Optimization No Whole Program Optimization
Character Set
Tells the compiler to use the specified character set; aids in localization issues.
OK
Отмена
Применить


MyProject1 Property Pages
Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...
Common Properties
Configuration Properties
General
Debugging
C/C++
Linker
Manifest Tool
XML Document Generator
Browse Information
Build Events
Custom Build Step
Code Analysis
Application Verifier
General
Output Directory
$(SolutionDir)$(ConfigurationName)
Intermediate Directory
$(ConfigurationName)
Extensions to Delete on Clean
.obj;.ilk;*.tlb;*.tli;*.tlh;*.tmp;*.rsp;*.pgc;*.pgd;*.meta;$
Build Log File
$(IntDir)\BuildLog.htm
Inherited Project Property Sheets
Enable Managed Incremental Build
Yes
Project Defaults
Configuration Type
Application (.exe)
Use of MFC
Use Standard Windows Libraries
Use of ATL
Not Using ATL
Character Set
Use Multi-Byte Character Set
Common Language Runtime support
No Common Language Runtime support
Whole Program Optimization
No Whole Program Optimization
Character Set
Tells the compiler to use the specified character set; aids in localization issues.
OK
Отмена
Применить

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <process.h>
#include <windows.h>  // для поддержки русского
#define Pi 3.1415926

char* Rus(const char* text);
char bufRus[255];

void main ()   {
double a, a1, a2, z1, z2;
system("cls");
puts(Rus("Введите значение угла a:"));
fflush(stdin);
scanf("%lf", &a);
```

```
a1=2*pow(sin(3*Pi-2*a),2);
a2=pow(cos(5*Pi+2*a),2);
z1=a1*a2;
z2=(double)1/4-1./4*sin(5./2*Pi-8*a);
printf(Rus("\n Результаты работы: z1=%.4lf
   z2=%.4lf\n"), z1, z2);
getch();    // Извлекает знак из буфера клавиатуры
}
```

```
char* Rus(const char* text)   {
   CharToOem(text, bufRus);
   return bufRus;
                                }
```

/* Аналогично, если в программе есть консольный ввод текста и этот текст в дальнейшем надо сохранять в файлах с кодировкой ANSI, то перед сохранением нужно воспользоваться API-функцией обратного преобразования OemToChar(). */

Новые заголовки в программах на C++

При работе с библиотекой C++ в соответствии с новым стилем вместо имен заголовочных файлов указываются стандартные идентификаторы, по которым компилятор находит требуемые файлы.

Новые заголовки C++ являются абстракциями, гарантирующими объявление соответствующих прототипов и определений библиотеки языка C++.
Поскольку новые заголовки не являются именами файлов, для них не нужно указывать расширение, а только имя заголовка в угловых скобках.

- **Содержание заголовков нового стиля помещается в пространстве имен std. Традиционно имена библиотечных функций располагались в глобальном пространстве имен.**

Пространство имен (namespace) — это некая объявляемая область, необходимая для того, чтобы избежать конфликтов имен идентификаторов.

- **Чтобы пространство имен std стало видимым, используйте следующую инструкцию:**

  using namespace std;

- **Эта инструкция помещает std в глобальное пространство имен. После того как компилятор обработает эту инструкцию, вы сможете работать с заголовками как старого, так и нового стиля.**

Ввод-вывод потоками

```
#include<iostream>
#include<conio.h>

using namespace std;

void  main() {
 cout << " Hello! " << endl; //endl - переход на новую строку
 cout << "Input  i, j ";

int  i, j, k;
      cin  >> i >> j ;
      k = i + j ;
cout << " Sum  i , j = " << k << endl;
      _getch();
    }
```

Тема 4. Операторы передачи управления. Операторы цикла

4.1 Простой условный оператор

4.2 Синтаксис полного оператора условного выполнения

4.3 Условная операция

4.4 Примеры, реализующие ветвящиеся алгоритмы

4.5 Область видимости идентификатора

4.6 Спецификаторы классов памяти

4.7 Составление циклических алгоритмов

4.7.1 Оператор с предусловием while

4.7.2 Оператор с постусловием do - while

4.7.3 Оператор с предусловием и коррекцией for

4.7.4 Пример программирования циклических алгоритмов

4.8 Оператор выбора альтернатив (переключатель)

4.9 Оператор безусловного перехода

Управляющие операторы

К управляющим операторам относятся:

- **операторы условного и безусловного перехода;**
- **оператор выбора альтернатив (переключатель);**
- **операторы организации циклов и передачи управления.**

Допустима вложенность операторов. В случае необходимости можно использовать составной оператор – блок { }

4.1 Простой условный оператор

if (<выражение>) <оператор1>;

Примеры записи:

```
if (x>0)  x=0;

if (i!=1)   j++,  s=1;

if (i!=1) {  j++;  s=1;  }

if (getch()!=27) //Если нажата не Esc
    k=0;
```

4.2 Синтаксис полного оператора условного выполнения

```
if (<выражение>) <оператор1>;
                 else <оператор2>;
```

Примеры записи:

```
if (x>0)   j=k+10;
       else    m=i+10;
```

Если есть вложенная последовательность операторов **if-else**, то **else** связывается с ближайшим предыдущим **if**, не содержащим **else**.

```
if (n>0)
        if (a>b)  z=a;
         else   z=b;
```

Если необходимо связать фразу **else** с внешним **if**, то используем операторные скобки:

```
if (n>0)
        {
          if (a>b)  z=a;
        }
      else   z=b;
```

4.3 Условная операция ?

Тернарная операция ? имеет следующее представление:

<выражение1> ? <выражение2> : <выражение3>;

если <выражение1> отлично от нуля (истинно), то результатом операции является <выражение2>, в противном случае – результатом операции является <выражения3>.

Каждый раз вычисляется только одно из выражений.

4.4 Примеры, реализующие ветвящиеся алгоритмы

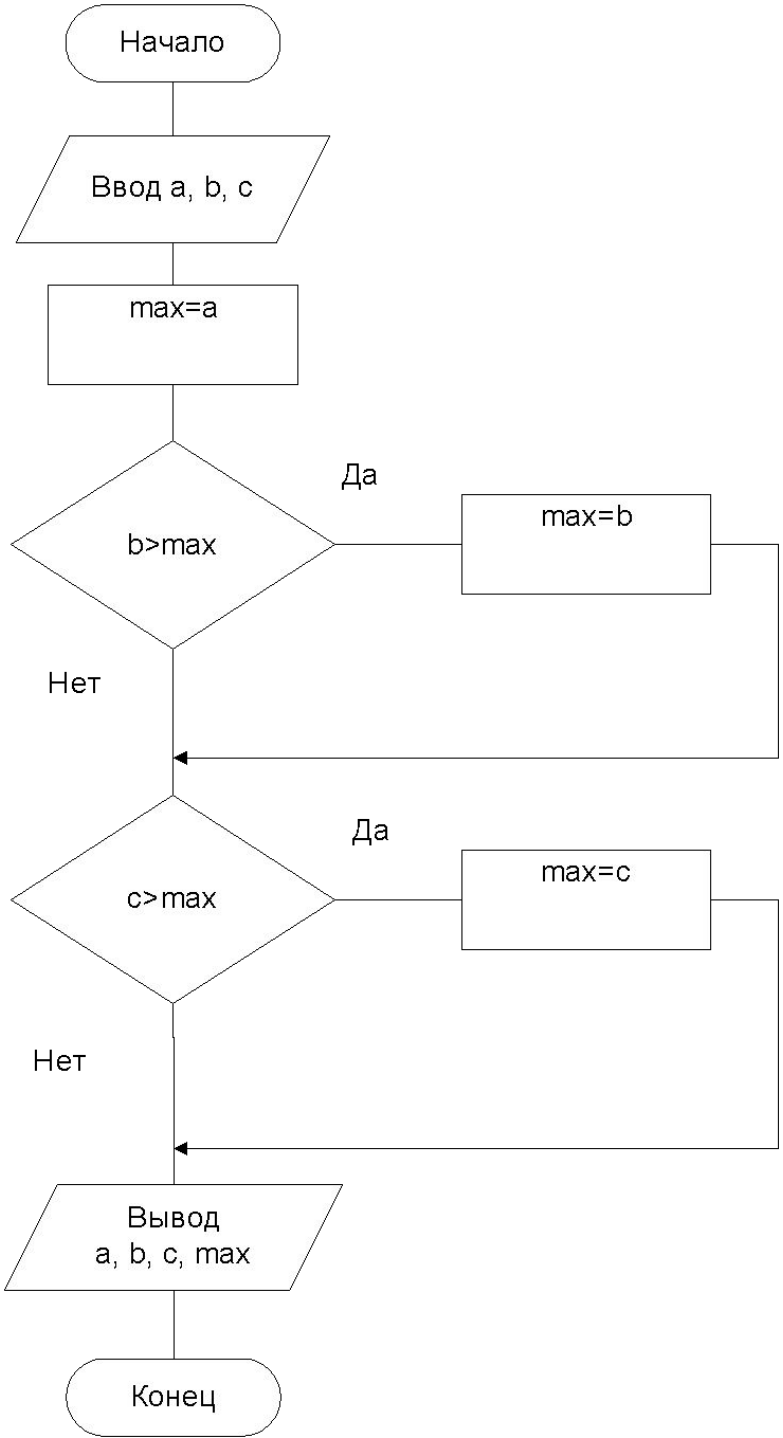
- **Максимум из a и b можно найти, используя:**

```
if (a > b)  z=a;
            else  z=b;
```

- **Используя условную операцию, этот пример можно записать:**

```
z = (a>b) ? a : b;
```

Поиск максимального из трех чисел

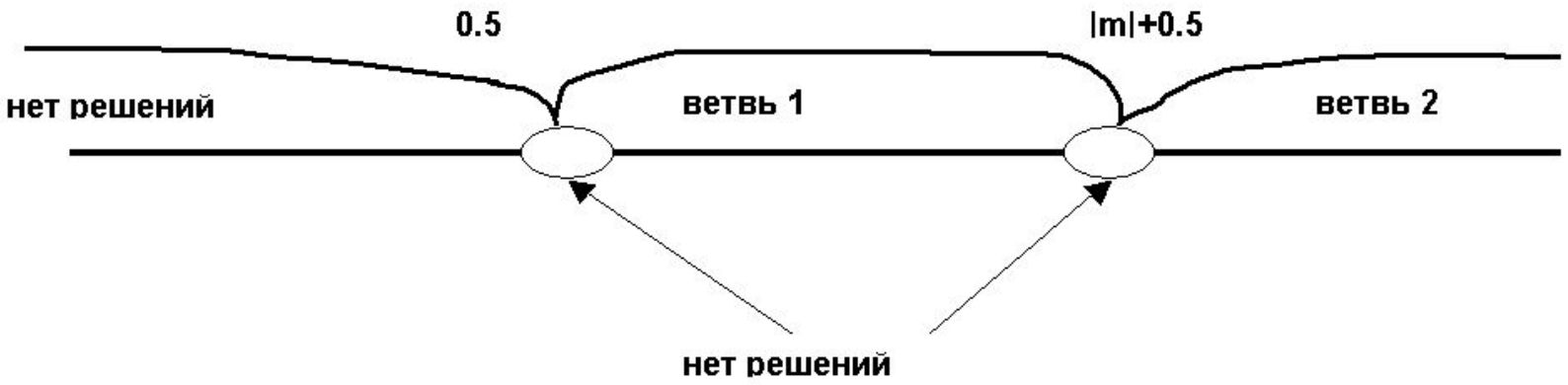


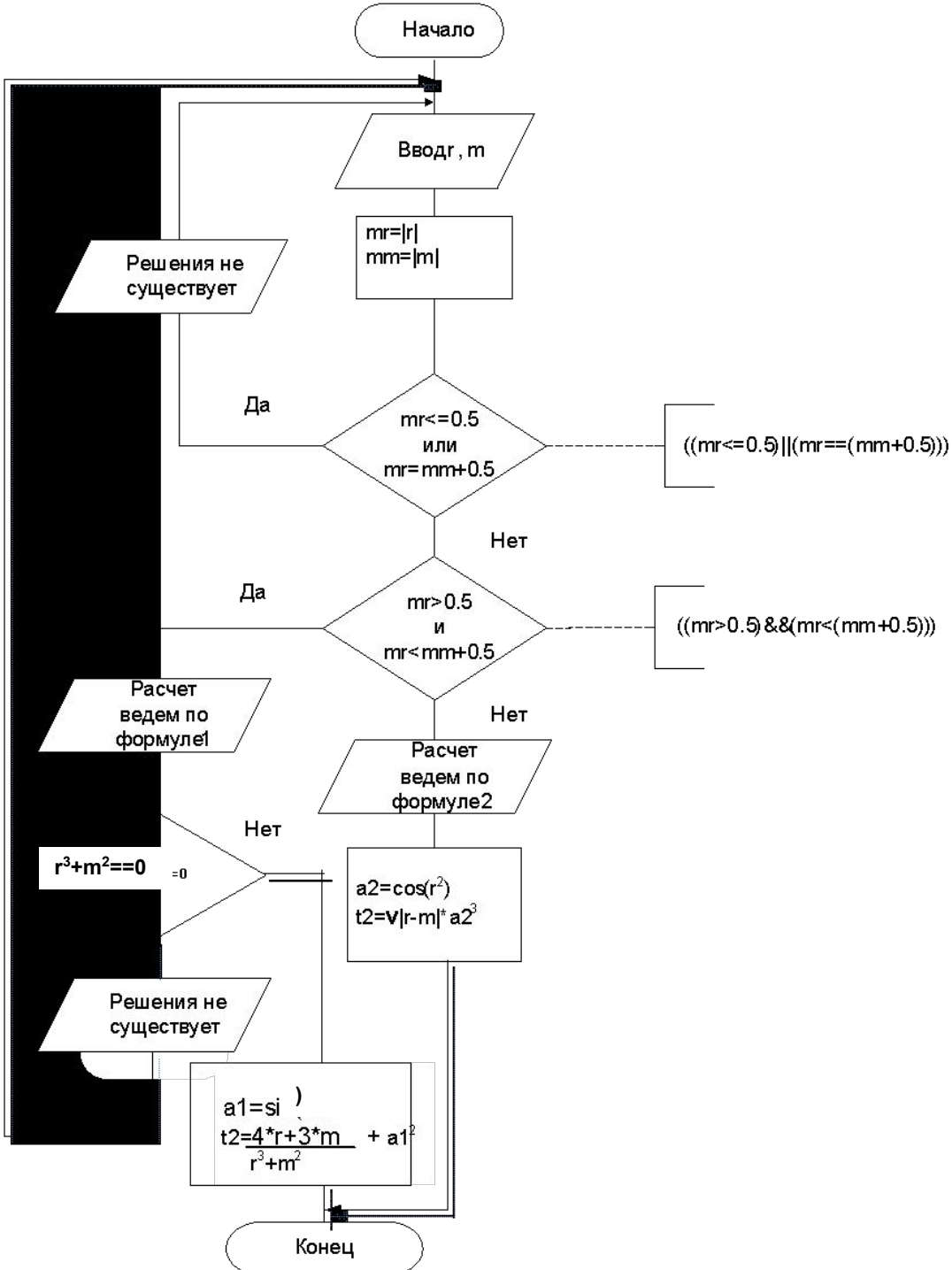
Найти максимальное из трех чисел

```
#include<stdio.h>
#include<conio.h>
#include <locale.h>
void main ()   {   setlocale(LC_CTYPE, "Russian");
double a, b, c, max;
puts("\n Введите a, b, c:");
scanf_s("%lf%lf%lf", &a, &b, &c);
max=a;
if(b>max) max=b;    if(c>max) max=c;
printf("\n Для a=%4.2lf, b=%4.2lf, c=%4.2lf
максимальным является max=%4.2f", a, b, c,
max);
_getch(); }
```

Составить программу для вычисления составной функции. Самостоятельно выбрать необходимое количество исходных данных для того, чтобы в программе выполнялись все возможные ветви алгоритма. Перед выводом полученного результата программа должна сообщать о ветви, при прохождении которой он получен (или сообщить о том, что решения не существует).

$$t2= \begin{cases} \dfrac{4r+3m}{r^3+m^2} * \sin^2 m^3, & \text{если } 0.5 < |r| < |m| + 0.5; \\ \sqrt{|r-m|} * \cos^3 r^2, & \text{если } |r| > |m| + 0.5; \end{cases}$$




Начало
Вводr , m
mr=|r|
mm=|m|
Решения не существует
Да
mr<=0.5
или
mr=mm+0.5
((mr<=0.5)||(mr==(mm+0.5)))
Нет
Да
mr>0.5
и
mr<mm+0.5
((mr>0.5)&&(mr<(mm+0.5)))
Расчет ведем по формуле1
Нет
Расчет ведем по формуле2
Нет
r3+m2==0
=0
a2=cos(r2)
t2=√|r-m|* a23
Решения не существует
a1=si)
t2=4*r+3*m / r3+m2 + a12
Конец

4.5 Область видимости идентификатора

Если переменная объявлена вне какого-либо блока, то такая переменная называется глобальной, в противном случае – локальной. Областью видимости идентификатора называется часть программы, в которой можно сослаться на этот идентификатор. Существуют четыре типа областей видимости:

- **блок;**
- **функция;**
- **прототип функции;**
- **исходный файл.**

- **Глобальные переменные** видимы внутри исходного файла, в котором они определены;
- **Локальные переменные** видимы только внутри блока или функции, в которой они определены;
- **Параметры функции** видны только внутри функции или объявления этой функции;

- **Функции всегда определяются вне какого-либо блока;**
- **Областью определения функции является исходный файл, в котором эта функция определена;**

4.6 Спецификаторы классов памяти

Класс памяти программного объекта определяет время его существования (время жизни) и область видимости (действия). Может принимать одно из значений:

- **auto - автоматический**
- **extern - внешний**
- **register - регистровый**
- **static - статический .**

Класс памяти и область видимости объектов по умолчанию зависят от места их размещения в коде программы.

Локальная переменная скрывает любую переменную с тем же именем, объявленную вне этого блока.

```
#include <stdio.h>
// Сокрытие имен переменных
int n=1;    //Глобальная
void main() {
   printf("n=%d\n",n);   //n=1
   { int n=2; //Локальная
    printf("n=%d\n",n); //n=2
    {n=3;
    printf("n=%d\n",n); //n=3
        }
     }          }
```

К скрытому глобальному имени можно обратиться с помощью оператора разрешения области видимости ::

```
#include <stdio.h>
int n=1;    //Глобальная
void main() {
   printf("n=%d\n",n);   //n=1
   { int n=2; //Локальная
printf("\n LOCAL n=%d GLOBAL n=%d \n", n, ::n);
//LOCAL n=2  GLOBAL n=1
    }          }
```

- **Время существования переменной или функции определяется как время, в течение которого эта переменная или функция хранится в памяти компьютера;**

- **Глобальные переменные существуют в течение всего времени выполнения программы и хранятся в фиксированной области памяти программы, которая задается во время ее компиляции;**

- **Если глобальная переменная не проинициализирована, то компилятор устанавливает ее значение в ноль;**

- **Память для локальных переменных выделяется динамически во время исполнения программы. При вызове функции или входе в блок, в котором объявлены локальные переменные, память под эти переменные распределяется, а при возврате из функции или выходе из блока эта память освобождается;**

- **Если локальная переменная не проинициализирована при ее объявлении, то ее начальное значение не определено.**

4.7 Составление циклических алгоритмов

Под циклом понимается организованное повторение некоторой последовательности операторов.

Перечень разновидностей операторов цикла:

- **оператор цикла с предусловием;**
- **оператор цикла с постусловием;**
- **оператор цикла с предусловием и коррекцией.**

- **Один проход цикла называется итерацией;**
- **Проверка условия выполняется на каждой итерации либо до кода цикла (с предусловием), либо после кода цикла (с постусловием).**

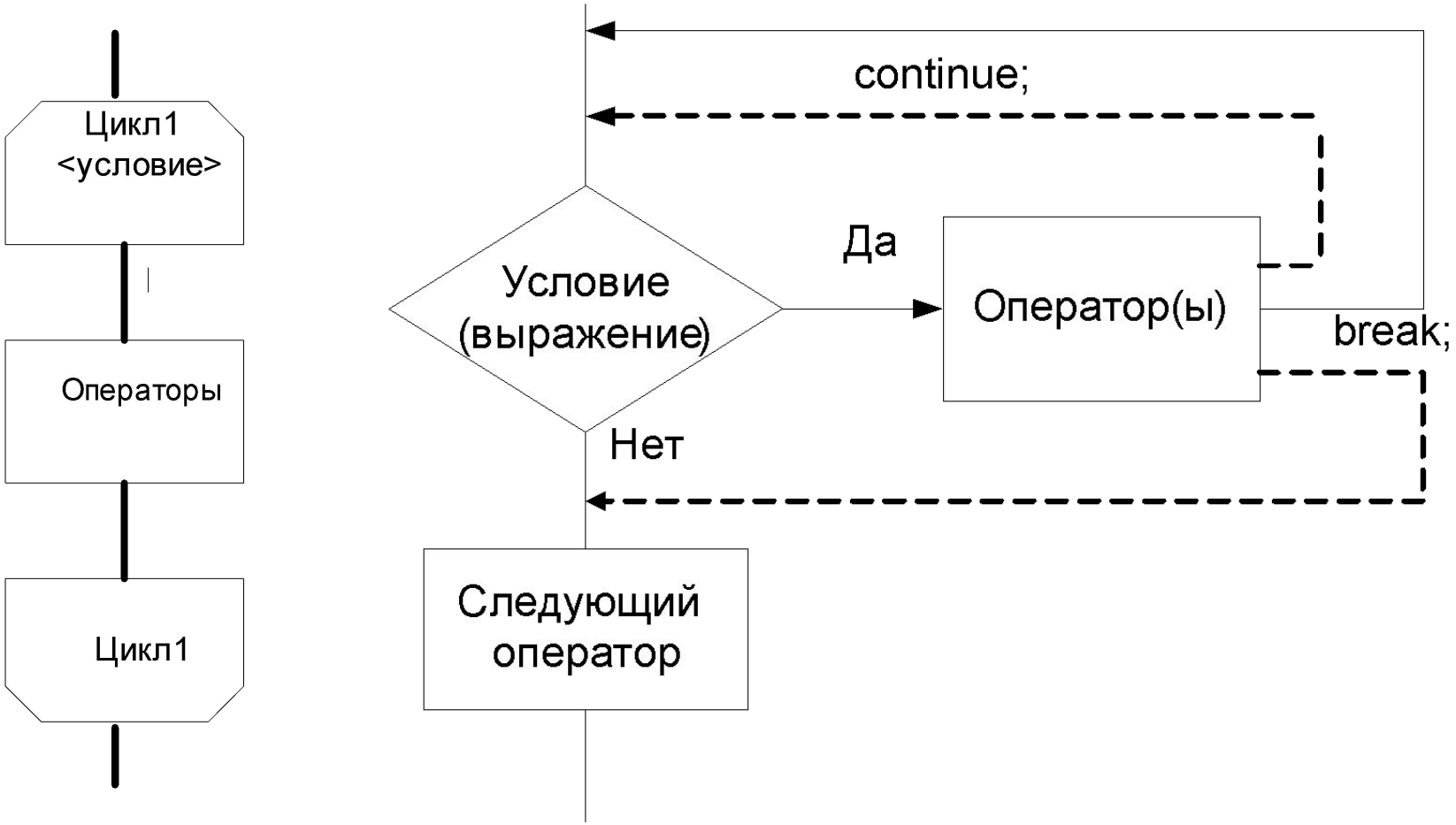
4.7.1 Оператор с предусловием while

while (<выражение>)

<код_цикла>;

Повторное выполнение кода цикла, пока <выражение> не примет значение ложь (0).

Структура оператора while



Особенности оператора while

1. **Если изначально выражение ложно, то операторы цикла (код цикла) ни разу не выполняется.**

2. **В коде цикла необходимо предусмотреть изменение величины выражения (для предотвращения зацикливания).**

```
int i=5, j=0;
while (i>0)
 { j++;}
```

```
int i=5, j=0;
while  (i>0)
 { j++; i--;}
```

3. Используя оператор **break** можно досрочно завершить цикл.

Присутствующий в теле цикла оператор **break** прекращает выполнение цикла и передает управление на следующий оператор, стоящий после данного цикла.

```
int i=5, j=0;
while  (i>0) {
    j++;
    if (j==2) break;
    i--;            }
```

4. Используя дополнительное условие и оператор **continue** можно досрочно прервать текущий шаг и передать управление на начало цикла.

При использовании оператора **continue** в **while** часть цикла, расположенная после **continue**, не выполняется, а управление сразу передается на проверку условия.

```
int x=-10, y=0;  //вар 2: int x=-2, y=1;
while  (x) {
++x;
if (!(x+y)) continue;   // вар 2: x=-1 y=1    !(-1+1)
--y;            }
```

1. x=-10 y=0 x=-9 -9+0=-9 y=-1
2. x=-9 y=-1 x=-8 -8+(-1)=-9 y=-2
3. x=-8 y=-2 x=-7 -7+(-2)=-9 y=-3
4. x=-7 y=-3 x=-6 -6+(-3)=-9 y=-4

x=0 y=-10 после выхода из цикла

Особенности оператора while

5. Запрещается передача управления из вне во внутрь цикла.

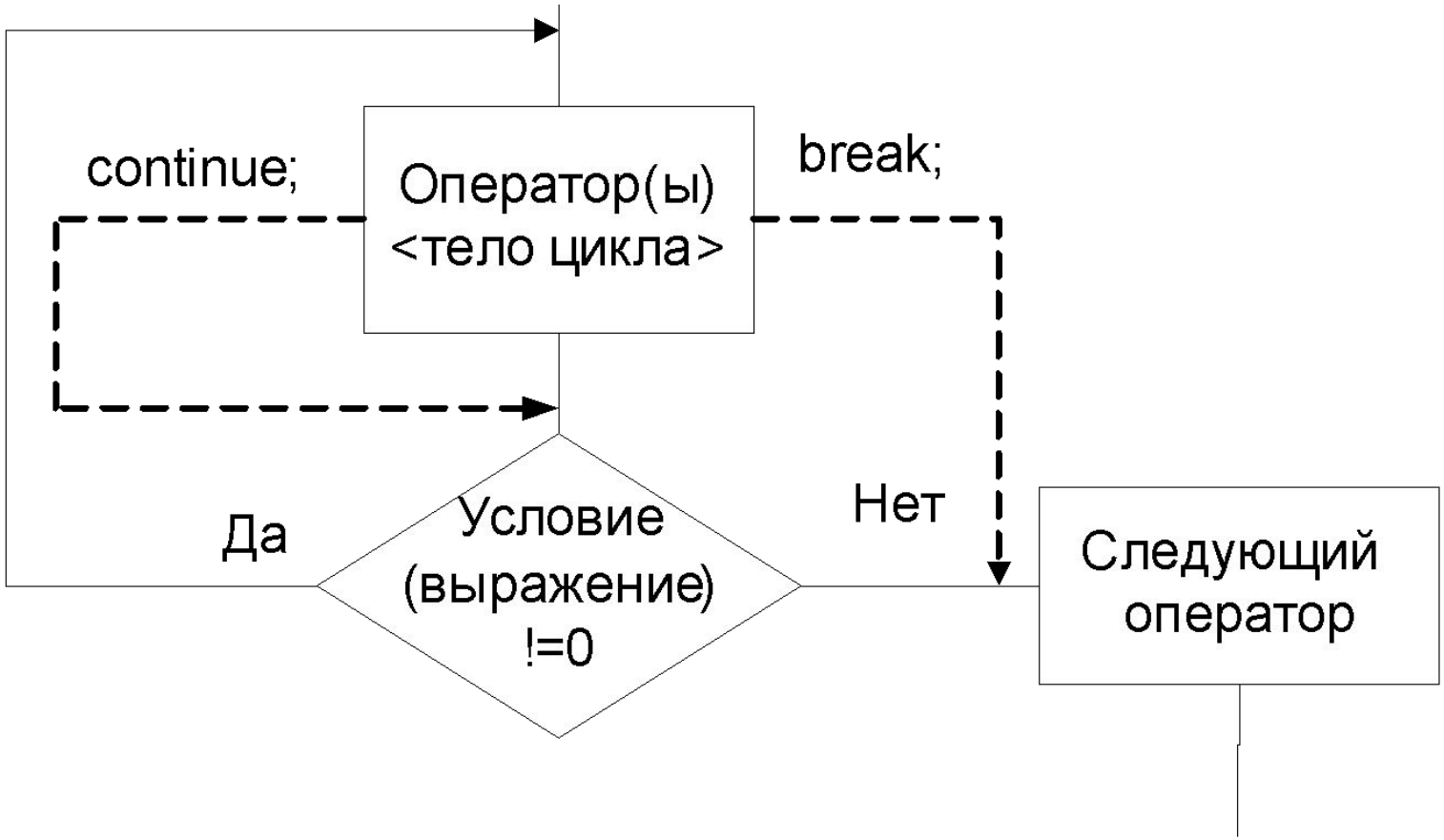
Организация бесконечного цикла

```
    while (1) {                                    ...
  if (getch()==27)      break;
// Если нажата клавиша и код ее равен 27
//(код клавиши "Esc"),
// то выходим из цикла
            ...
      }
```

4.7.2 Оператор с постусловием do - while

```
do <код_цикла>
   while
   (<выражение>);
```

Структура оператора do - while



Особенности оператора do - while

Код цикла будет выполняется до тех пор , пока выражение истинно. Всегда выполняется хотя бы один раз.

```
int a=-2;
do
 ++a;
while (a);
```

Итерация1: a=-1

Итерация2: a=0 Выход из цикла

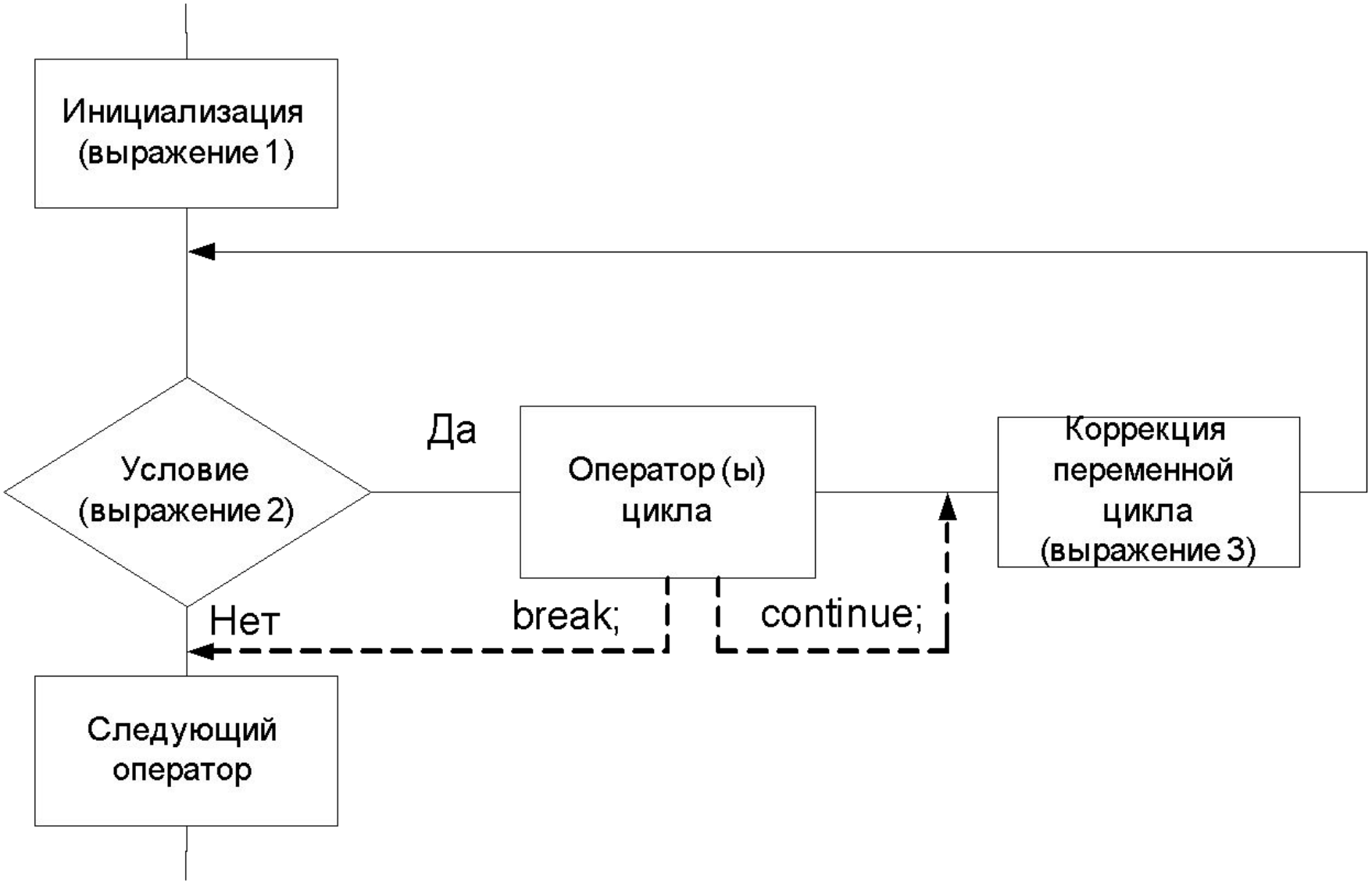
4.7.3 Оператор с предусловием и коррекцией for

for (<выражение1>; <выражение2>; <выражение3>)

<код_цикла>;

- **выражение1 – инициализация счетчика (начальное значение);**
- **выражение2 – условие продолжения счета;**
- **выражение3 – увеличение счетчика;**
- **выражения 1,2 и 3 могут отсутствовать (пустые выражения), но символы «;» опускать нельзя.**

Структура оператора for



Для суммирования первых натуральных чисел можно записать:

```
...
int sum = 0;
for ( int i=1; i<=N; i++)
sum+=i;
...
```

Отсутствует выражение1

```
int i=1;
for ( ; i<=5; i++)
printf ("\n%d", i);
```

Отсутствует выражение3

```
for (printf("Введите числа по порядку! \n");
    num!=6;)
scanf("%d", &num);
printf("Последнее число-6\n");
```

Первое сообщение выводиться только раз, а затем осуществляется прием вводимых чисел, пока не поступит 6.

Вечный цикл

```
for ( ; ; )
        printf ("Бесконечно
        повторяющаяся строка");
```

Особенности оператора for

1. Оператор for решение о выполнении цикла принимает до начала его прохождения. Поэтому он может не выполниться ни разу.

2. Можно использовать символы.

```
for (let= 'z'; let>='a'; let--)
printf("Код ASCII=%d, символ %c\n", let, let);
```

На экран выводятся код и изображение символа.

Особенности оператора for

3. Операция "," позволяет включить в оператор for несколько инициализирующих или корректирующих выражений.

Операторы, записанные через "," , рассматриваются как один составной оператор.

```
for (i= 1, j=5; i<j; i++, j--)      printf("\nHello");
```

1. i=1, j=5; 1<5; Hello; i++(2), j--(4)
2. i=2, j=4; 2<4; Hello; i++(3), j--(3)
3. i=3, j=3; 3<3; выход из цикла

Особенности оператора for

4. В цикле с использованием оператора for после выполнения continue, управление передается на вычисление третьего оператора в скобках, затем на проверку условия.

Вывести на экран числа кратные 10, но меньше 100.

```
for (int a=0; a<100; a++)
    {
      if (a%10)  continue;  //остаток !0
      printf("\n%d",a);
    }
```

Результат: 0 10 20 30 40 50 60 70 80 90

Особенности оператора for

5. **Выражение1 может содержать объявление переменных. В этом случае объявленная переменная видима только в цикле for.**

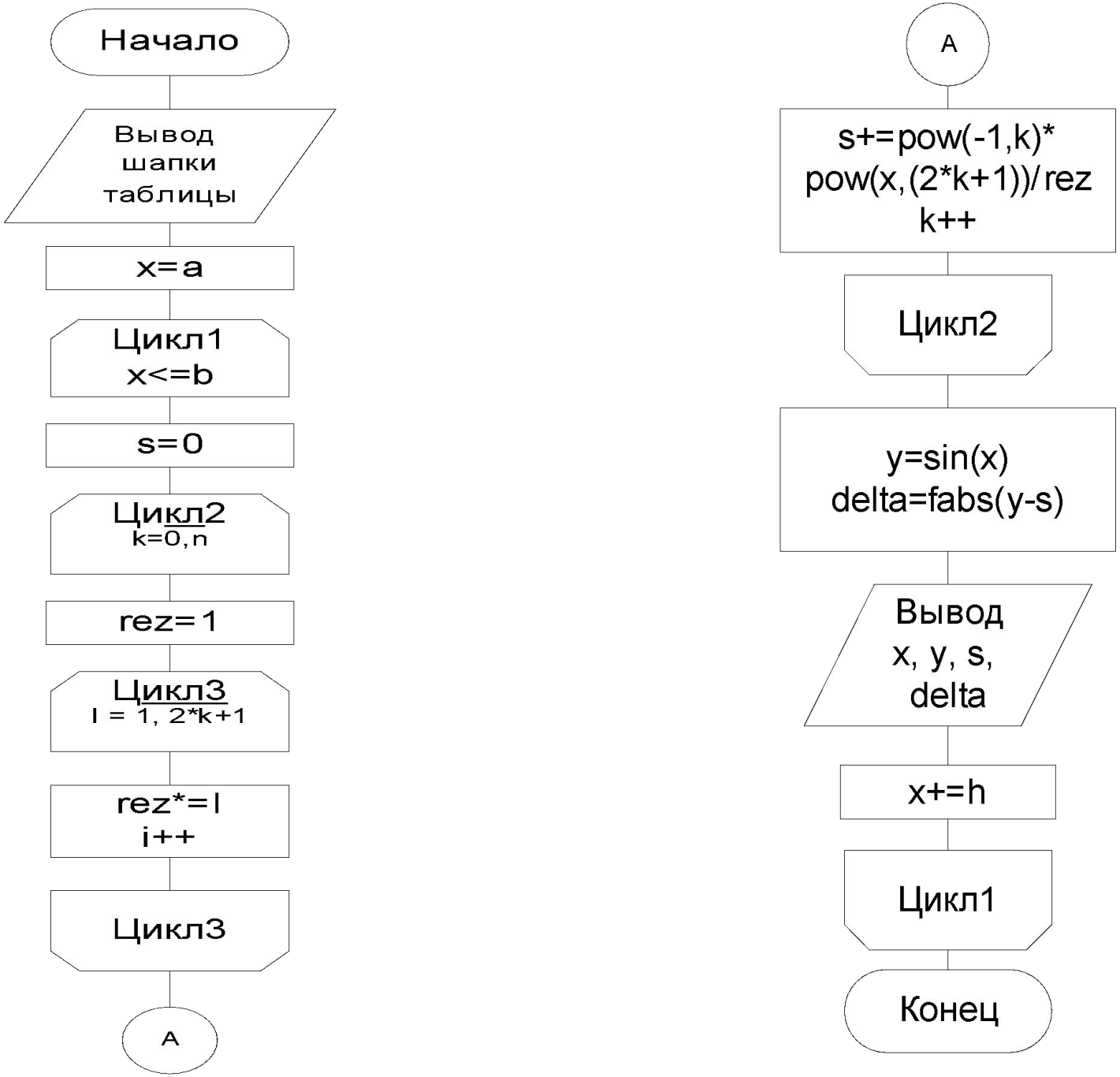
   Инициализация переменной цикла выполняется в программе только один раз.

4.7.4 Пример программирования циклических алгоритмов.

Значение аргумента **x** изменяется от **a** до **b** с шагом **h**. Для каждого x найти значения функции Y(x), суммы **S(x)** и **|Y(x)-S(x)|** и вывести в виде таблицы. Значения **a, b, h** и **n** вводятся с клавиатуры. Так как значение **S(x)** является рядом разложения функции **Y(x)**, то значения **S** и **Y** для данного аргумента **x** должны **совпадать в целой части и в первых двух-четырех позициях после десятичной точки**.

Работу программы проверить для a=0.1; b=1.0; h=0.1; n выбрать самостоятельно.

$$S(x)=\sum_{k=0}^{n}(-1)^k\frac{x^{2k+1}}{(2k+1)!},\qquad Y(x)=\sin(x)$$


Начало
Вывод шапки таблицы
x=a
Цикл1 x<=b
s=0
Цикл2 k=0,n
rez=1
Цикл3 I = 1, 2*k+1
rez*=I i++
Цикл3
A
A
s+=pow(-1,k)* pow(x,(2*k+1))/rez k++
Цикл2
y=sin(x) delta=fabs(y-s)
Вывод x, y, s, delta
x+=h
Цикл1
Конец

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <process.h>
#include <locale.h>
void main ()   {   setlocale(LC_CTYPE, "Russian");
double a, b, h, x, rez, y, s, delta;
int n, i, k;
system("cls");  fflush(stdin);
printf("Введите a:");
scanf_s("%lf", &a);
printf("Введите b:");
scanf_s("%lf", &b);
printf("Введите h:");
scanf_s("%lf", &h);
printf("Введите n:");
scanf_s("%d", &n);
```

Лабораторная работа №3

```
printf("\n");
puts("   x  |   Y(x)  |  S(x)  |    Y(x)-S(x)  |");
puts("---------------------------------- ");
x=a;
while (x<=b){
s=0;
for (k=0;k<=n; k++) {
rez=1;
for (i=1;i<=2*k+1;i++)
    rez*=i;
s+=pow(-1.,k)*pow(x,(2*k+1))/rez;
}
y=sin(x);
delta=fabs(y-s);
printf("\n %.2lf | %.5lf | %.5lf|    %.5lf    | \n", x, y, s, delta);
x+=h;
}   _getch();  }
```

Результаты

```
Введите n:10

   x  |   Y(x)   |  S(x)   |    Y(x)-S(x)   |
-----------------------------------------------

 0.10 | 0.09983 | 0.09983|    0.00000    |

 0.20 | 0.19867 | 0.19867|    0.00000    |

 0.30 | 0.29552 | 0.29552|    0.00000    |

 0.40 | 0.38942 | 0.38942|    0.00000    |

 0.50 | 0.47943 | 0.47943|    0.00000    |

 0.60 | 0.56464 | 0.56464|    0.00000    |

 0.70 | 0.64422 | 0.64422|    0.00000    |

 0.80 | 0.71736 | 0.71736|    0.00000    |

 0.90 | 0.78333 | 0.78333|    0.00000    |

 1.00 | 0.84147 | 0.84147|    0.00000    |
```

4.8 Оператор выбора альтернатив (переключатель)

```
switch (выражение) {
    case константа1: оператор1; [break;]
    case константа2: оператор2;  [break;]
            ...
    case константаN: операторN;  [break;]
    [default:  оператор(N+1);  break;]
                        }
```

- **Значение выражения должно быть целого типа (символьного);**

- **После вычисления оно сравнивается со значениями констант и при совпадении с одной из них выполняется передача управления соответствующему оператору;**

- **В случае несовпадения значения выражения ни с одной из констант происходит переход на метку default, либо, при ее отсутствии, к оператору, следующему за оператором switch;**

- **Управляющий оператор break позволяет организовать выход из оператора switch на первый выполняемый оператор, следующий после данной конструкции.**

Пример : результатом будет – Случай 2.

```
void main()
    {   int i = 2;
    switch(i)      {
    case 1: puts ( "Случай 1. "); break;
    case 2: puts ( "Случай 2. "); break;
    case 3: puts ( "Случай 3. "); break;
    default: puts ( "Случай default. "); break;
                        }
 }
```

Пример : результатом будет –

Случай 2.
Случай 3.
Случай default.

```
void main()
    {   int i = 2;
    switch(i)      {
    case 1: puts ( "Случай 1. ");
    case 2: puts ( "Случай 2. ");
    case 3: puts ( "Случай 3. ");
   default: puts ( "Случай default. ");
                   }
    }
```

4.9 Оператор безусловного перехода

Оператор безусловного перехода goto < метка >;
предназначен для передачи управления на оператор, помеченный меткой.

Метка представляет собой идентификатор с символом «двоеточие» после него.
m1: ;

Наиболее характерный случай использования оператора goto – выполнение выхода во вложенной структуре при возникновении неисправимых ошибок во входных данных.

Пример: программа – простейший калькулятор

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <process.h>
#include <locale.h>
void main ()   {   setlocale(LC_CTYPE, "Russian");
double a, b, c;
char s;
m1:fflush(stdin);
puts("\nВведите операнд1, символ операции,
    операнд2:");
scanf_s("%lf%c%lf", &a, &s,1, &b);
```

```
switch(s) {
case'+': c=a+b; break;
    case'-': c=a-b; break;
     case'*': c=a*b; break;
          case'/': c=a/b; break;
default: printf("\n Ошибка, повторите ввод!\n");
                        goto m1;       }
printf("\n a %c b = %lf", s, c);
printf("\n Продолжим? (Y/y)\n");
s=_getch();
 if((s=='Y')||(s=='y')) goto m1;
 printf("\n The end\n");
_getch();
}
```

Результаты

```
Введите операнд1, символ операции, операнд2:
1+6

 a + b = 7.000000
 Продолжим? (Y/y)

Введите операнд1, символ операции, операнд2:
3*20

 a * b = 60.000000
 Продолжим? (Y/y)

Введите операнд1, символ операции, операнд2:
34%5

 Ошибка, повторите ввод!

Введите операнд1, символ операции, операнд2:
34-5

 a - b = 29.000000
 Продолжим? (Y/y)
```