# 4. Основные операторы С#

## 4.1.Базовые конструкции структурного программирования

Структурное программирование — это технология создания программ, позволяющая путем соблюдения определенных правил сократить время разработки и уменьшить количество ошибок, а также облегчить возможность модификации программы.

В теории программирования доказано, что программу для решения задачи любой сложности можно составить только из трех структур: линейной, разветвляющейся и циклической. Эти структуры называются базовыми конструкциями структурного программирования.

Линейной называется конструкция, представляющая собой последовательное соединение двух или более операторов.

Ветвление – задает выполнение одного из двух операторов, в зависимости от выполнения какого либо условия.

Цикл – задает многократное выполнение оператора.

|  |  |  |
| --- | --- | --- |
| Следование | Ветвление | Цикл |
|  |  |  |

Рисунок . Базовые конструкции структурного программирвания

Целью использования базовых конструкций является получение программы простой структуры. Такую программу легко читать, отлаживать и при необходимости вносить в нее изменения. Структурное программирование также называют программированием без goto, т. к. частое использование операторов перехода затрудняет понимание логики работы программы. Но иногда встречаются ситуации, в которых применение операторов перехода, наоборот, упрощает структуру программы.

Операторы управления работой программы называют управляющими конструкциями программы. К ним относят:

1. составные операторы;
2. операторы выбора;
3. операторы циклов;
4. операторы перехода.

### 4.2. Оператор «выражение»

Любое выражение, заканчивающееся точкой с запятой, рассматривается как оператор, выполнение которого заключается в вычислении этого выражения. Частным случаем выражения является пустой оператор ; (точка с запятой).

i++;

a+=2;

x=a+b;

### 4.3. Составные операторы

Составной оператор (блок) это последовательность операторов, заключенная в фигурные скобки.

{

int s=0;

do {

int a=Convert.ToInt32(Console.ReadLine());

s+=a;

} while(a!=0);

### 4.4. Операторы выбора

Операторы выбора – это условный оператор и переключатель.

1. Условный оператор имеет полную и сокращенную форму.

if (выражение-условие) оператор; //сокращенная форма

В качестве выражения-условия могут использоваться отношение и логическое выражение. Если значение выражения-условия истинно, то выполняется оператор.

if(a<b&&a<c) min=a;

if (выражение-условие) оператор1; //полная форма

else оператор2;

Если значение выражения-условия отлично от нуля, то выполняется оператор1, при нулевом значении выражения-условия выполняется оператор2.

double x1, x2;

double d = Math.Pow(b, 2) - 4 \* a \* c;

if (d < 0)

Console.WriteLine("Решения нет");

else

{

x1 = (-b + Math.Sqrt(d)) / (2 \* a);

x2 = (-b - Math.Sqrt(d)) / (2 \* a);

Console.WriteLine("x1={0}, x2={1}", x1.ToString("f5"), x2.ToString("f5"));

}

2. Переключатель определяет множественный выбор.

switch (выражение)

{

case константа1 : оператор1 ; break;

case константа2 : оператор2 ; break;

. . . . . . . . . . .

[default: операторы; break;]

}

При выполнении оператора switch, вычисляется выражение, записанное после switch, оно должно быть целочисленным. Полученное значение последовательно сравнивается с константами, которые записаны следом за case. При первом же совпадении выполняются операторы, помеченные данной меткой. Если выполненные операторы не содержат оператора перехода, то далее выполняются операторы всех следующих вариантов, пока не появится оператор перехода или не закончится переключатель. Если значение выражения, записанного после switch, не совпало ни с одной константой, то выполняются операторы, которые следуют за меткой default. Метка default может отсутствовать.

int i;

Console.WriteLine("Введите целое число");

i=Int32.Parse(Console.ReadLine());

switch (i)

{

case 1: Console.WriteLine("\nThe number is one");

break;

case 2: Console.WriteLine("\n2\*2={0}", i \* i); break;

case 3: Console.WriteLine("\n3\*3="+ i \* i); break;

case 4: Console.WriteLine("\n" + i + " is very beautiful!");

break;

default: Console.WriteLine("\nThe end of work"); break;

}

Результаты работы программы:

1. При вводе 1 будет выведено:

The number is one

2\*2=1

3\*3=1

1. При вводе 2 будет выведено:

2\*2=4

3\*3=4

1. При вводе 3 будет выведено:

3\*3=9

1. При вводе 4 будет выведено:

4 is very beautiful!

1. При вводе всех остальных чисел будет выведено:

The end of work

### 17.5. Операторы циклов

1. Цикл с предусловием:

оператор\_1; //инициализация

while (условие)

оператор\_2; //коррекция

В качестве <условия> используется отношение или логическое выражение. Если оно истинно, то тело цикла выполняется до тех пор, пока выражение-условие не станет ложным.

int a=1,s=0; //инициализация

while (a!=0)

{

a=Int32.Parse(Console.ReadLine()); //коррекция a

s+=a;

}

1. Цикл с постусловием:

оператор\_1; //инициализация

do

оператор\_2;//коррекция

while (условие);

Тело цикла выполняется до тех пор, пока условие истинно.

int a, s=0; //инициализация

do

{

a=Int32.Parse(Console.ReadLine()); //коррекция a

s+=a;

} while (a!=0);

1. Цикл с параметром:

for(выражение\_1; выражение\_2; выражение\_3) оператор;

выражение\_1 – инициализация,

выражение\_2 – условие,

выражение\_3 – коррекция.

Выражение\_1 и выражение\_3 могут состоять из нескольких выражений, разделенных запятыми. Выражение\_1 – задает начальные условия для цикла (инициализация). Условие определяет условие выполнения цикла, если оно истинно, цикл выполняется, а затем вычисляется значение выражения\_3. Выражение\_3 – задает изменение параметра цикла или других переменных (коррекция). Цикл продолжается до тех пор, пока выражение-условие не станет равно 0. Любое выражение может отсутствовать, но разделяющие их « ; » должны быть обязательно.

//1 – увеличение параметра

int a, s=0,i;

for(i=0; i<10; i++)

{

a=Int32.Parse(Console.ReadLine());

s+=a;

}

//2 - уменьшение параметра

s = 0;

for(i=n; i>0; i--)

{

a=Int32.Parse(Console.ReadLine());

s+=a;

}

//3 изменение шага корректировки

s=0;

Console.Write(0);

for (i = 2; i < 60; i += 13)

{

s += i;

Console.Write("+"+ i );

}

Console.WriteLine("=" + s);

/\*4 проверка условия отличного от того, которое налагается на число итераций\*/

s=0;

Console.Write(0);

for ( i=1;i\*i<100; i++)

{

s += i;

Console.Write("+" + i);

}

Console.WriteLine("=" + s);

//5 коррекция с помощью умножения

sd=0;

Console.Write(0);

for ( id=10.0; id<15.0; id\*=1.1)

{

sd += id;

Console.Write("+" + id);

}

Console.WriteLine("=" + sd);

### //6 коррекция с помощью арифметического выражения

### int x,y=0;

### for (x=1;y<=75;y=5\*(x++)+10)

### {

### Console.WriteLine("x={0}, y={1}",x,y);

### }

### // 7 использование нескольких корректирующих

### //выражений, тело цикла отсутствует

### for (x=1, y=0; x<10;x++, y+=x);

### 4.6. Операторы перехода

Операторы перехода выполняют безусловную передачу управления.

1. break – оператор прерывания цикла.

int summa=0;

for (int i = 0; i < 10; i++)

{

Console.Write(">");

int number = Int32.Parse(Console.ReadLine());

if (number == 0) break;

summa += number;

}

Console.WriteLine("Summa=" + summa);

Т. е. оператор break целесообразно использовать, когда условие продолжения итераций надо проверять в середине цикла.

2. continue – переход к следующей итерации цикла. Он используется, когда тело цикла содержит ветвления.

int summa\_pol = 0, kolich\_pol = 0;

for (tek = 0, number = 1; number != 0; tek++)

{

Console.Write(">");

number = Int32.Parse(Console.ReadLine());

if (number <= 0) continue;

summa\_pol += number; kolich\_pol++;

}

Console.WriteLine("summa\_pol=" + summa\_pol);

Console.WriteLine("kolich\_pol=" + kolich\_pol);

3. goto – безусловный переход, используется в трех формах:

* goto метка:
* goto case константа:
* goto default:

Оператор goto используется:

* необходим принудительный выход из нескольких вложенных циклов;
* необходим переход из нескольких точек функции в одну точку.

Метка – это обычный идентификатор, областью видимости которого является функция. Оператор goto передает управления оператору, стоящему после метки. Использование оператора goto оправдано, если необходимо выполнить переход из нескольких вложенных циклов или переключателей вниз по тексту программы или перейти в одно место функции после выполнения различных действий.

Применение goto нарушает принципы структурного и модульного программирования, по которым все блоки, из которых состоит программа, должны иметь только один вход и только один выход.

Нельзя передавать управление внутрь операторов if, switch и циклов. Нельзя переходить внутрь блоков, содержащих инициализацию, на операторы, которые стоят после инициализации.

int k;

goto m;

. . .

{

int a=3,b=4;

k=a+b;

m: int c=k+1;

. . .

}

В этом примере при переходе на метку m не будет выполняться инициализация переменных a , b и k.

1. return – оператор возврата из функции. Он всегда завершает выполнение функции и передает управление в точку ее вызова. Вид оператора:

return [выражение];

int func1()

{

. . . .

return 1;

}

### 4.7. Оператор try

При выполнении операторов программы может возникнуть исключение, т.е. ошибочная ситуация (деление на 0, обращение к элементу массива с несуществующим индексом и т. п.).

Общая схема обработки исключений следующая:

В одной части программы, где обнаружена аварийная ситуация, исключение порождается. другая часть программы контролирует возникновение исключения, перехватывает и обрабатывает его.

Управле**ние механизмом обработки исключений основано на четы**рех ключевых словах: **try, catch, throw и finally** . Они образуют взаимосвязанную подсистему:

* Программные инструкции, которые нужно проконтролировать на предмет исключений, помещаются в try-блок.
* Если исключение таки возникает в этом блоке, оно дает знать о себе генерацией определенного рода информации. Системные исключения автоматически генерируются С#-системой динамического управления. Чтобы сгенерировать исключение вручную,используется ключевое слово throw.
* Сгенерированное исключение может быть перехвачено программным путем с помощью catch-блока и обработано соответствующим образом.
* Любой код, который должен быть обязательно выполнен при выходе из try-блока, помещается в блок finally.

Ядром обработки исключений являются блоки try и catch. Эти ключевые слова работают "в одной связке"; нельзя использовать слово try без catch или catch без try.

Формат записи try/catch-блоков обработки исключений:

try

{

// Блок кода, подлежащий проверке на наличие ошибок.

}

catch {ExceptionTypel ex)

{

// Обработчик для исключения типа ExceptionTypel.

}

catch (ExceptionType2 ex)

{

// Обработчик для исключения типа ЕхсерtionТуре2.

}

ЕхсерtionТуре — это тип сгенерированного исключения. После генерации исключение перехватывается соответствующей инструкцией catch, которая его обрабатывает. С try-блоком может быть связана не одна, а несколько catch-инструкций. Какая именно из них будет выполнена, определит тип исключения. Другими словами, будет выполнена та catch-инструкция, тип исключения которой совпадает с типом сгенерированного исключения (а все остальные будут проигнорированы). После перехвата исключения параметр ех примет его значение. Задавать параметр ех необязательно.

Важно понимать следующее. Если исключение не генерируется, try-блок завершается нормально, и все его catch-инструкции игнорируются. Выполнение программы продолжается с первой инструкции, которая стоит после последней инструкции catch. Таким образом, catch-инструкция (из предложенных после try-блока) выполняется только в случае, если сгенерировано соответствующее исключение.

**Пример обработки исключения**

do

{

try

{

Console.WriteLine("Введите кординату точки х");

x = Convert.ToDouble(Console.ReadLine());

ok = true;

}

catch (FormatException)

{

Console.WriteLine("Направильный ввод, введите вещественное число!");

ok = false;

}

}

while (!ok);

Иногда требуется перехватывать все исключения, независимо от их типа. Для этого используется catch-инструкция без параметров. В этом случае создается обработчик "глобального перехвата", который используется, чтобы программа гарантированно обработала все исключения.

catch

{

Console.WriteLine("Произошло исключение.");

}

В пространстве имен System определено несколько стандартных встроенных исключений. Все они выведены из класса SystemException, поскольку генерируются системой динамического управления (Common Language Runtime) при возникновении динамических ошибок.

|  |  |
| --- | --- |
| ***Исключение*** | ***Значение*** |
| ArrayTypeMismatchException | Тип сохраняемого значения несовместим с типом массива |
| DivideByZeroException | Попытка деления на нуль |
| IndexOutOfRangeException | Индекс массива оказался вне диапазона |
| InvalidCastException | Неверно выполнено динамическое приведение типов |
| Outof MemoryException | Обращение к оператору new оказалось неудачным из-за недостаточного объема свободной памяти |
| OverflowException | Имеет место арифметическое переполнение |
| NullReferenceException | Была сделана попытка использовать нулевую ссылку, т.е. ссылку, которая не указывает ни на какой объект |
| StackoverflowException | Переполнение стека |

**Использование ключевых слов checked и unchecked**

В С# предусмотрено специальное средство, которое связано с генерированием исключений, связанных с переполнением в арифметических вычислениях (в некоторых случаях при вычислении арифметических выражений получается результат, который выходит за пределы диапазона, определенного для типа данных в выражении). С# позволяет управлять генерированием исключений при возникновении переполнения с помощью ключевых слов checked и unchecked. Ключевое слово checked имеет две формы. Одна проверяет конкретное выражение и называется операторной checked-формой. Другая же проверяет блок инструкций.

checked (expr)

checked {

// Инструкции, подлежащие проверке.

}

Здесь expr — выражение, которое необходимо контролировать. Если значение контролируемого выражения переполнилось, генерируется исключение типа OverflowException.

Ключевое слово unchecked имеет две формы. Одна из них — операторная форма, которая позволяет игнорировать переполнение для заданного выражения. Вторая игнорирует переполнение, которое возможно в блоке инструкций.

unchecked (expr)

unchecked {

// Инструкции, для которых переполнение игнорируется.

}

Здесь expr — выражение, которое не проверяется на предмет переполнения. В случае переполнения это выражение усекается.

static void Main(string[] args)

{

byte a, b;

byte result;

a = 127;

b = 127;

try

{

unchecked

{

a = 127;

b = 127;

result = unchecked((byte)(a \* b));//=16129

Console.WriteLine("Unchecked-результат: " + result);

a = 125;

b = 5;

result = unchecked((byte)(a \* b));//=625

Console.WriteLine("Unchecked-результат: " + result);

}

checked

{

a = 2;

b = 7;

result = checked((byte)(a \* b)); // Все в порядке.

Console.WriteLine("Checked-результат: " + result);

a = 127;

b = 127;

result = checked((byte)(a \* b)); // Здесь должно быть сгенерировано исключение.

Console.WriteLine("Checked-результат: " + result); // Эта инструкция не выполнится.

}

}

catch (OverflowException exc)

{

// Перехватываем исключение.

Console.WriteLine(exc);

}

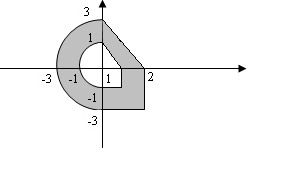
}

}

## 5. Примеры решения задач с использованием основных операторов С#

### 5.1. Программирование ветвлений

**Задача №1**. Определить, попадет ли точка с координатами (х, у ) в заштрихованную область.

****

1. **Анализ.**

*Исходные данные: х,у*

*Результат:* да или нет

*Классы входных данных:*

Точка (х,у) попадает

1. в 1 четверть.
   1. в заштрихованную область,
   2. на границу области,
   3. выше прямой, ограничивающей область,
   4. ниже прямой, ограничивающей область.
2. в 2 и 3 четверть.
3. в заштрихованную область,
4. на границу области,
5. за окружность, ограничивающую область,
6. внутрь окружности, ограничивающей область.
7. в 4 четверть.
8. в заштрихованную область,
9. на границу области,
10. правее прямой х=2 и ниже прямой у=-3, ограничивающих область,
11. левее прямой х=1 и выше прямой у=-1, ограничивающей область.

*Классы выходных данных:*

1. Да.
2. Нет.
3. Ошибка при вводе х.
4. Ошибка при у.

*Анализ условия.*

Ok=I || II || III || VI, где I, II, III, IV – условия попадания точки в заштрихованную область для каждой четверти.

Четверть I: Область формируется прямыми 0Х и 0У, прямой, проходящей через точки (0,1)и(1,0) и прямой, проходящей через точки (0,3) и (2,0).

Необходимо определить уравнения прямых у=ax+b. Решаем две системы уравнений:

1. 1=a\*0+b; 0=a\*1+b;
2. 2=a\*0+b; 0=a\*3+b;

Из этих систем получаем следующие уравнения прямых:

y=-1x+1;

y=-2/3x+1;

Тогда условие попадания точки в I квадрант будет выглядеть следующим образом:

y>=-x+1&&y<=-2/3x+2&&y>=0&&x>=0.

Четверти II и III: Область формируется прямыми 0Х и 0У и двумя окружностями, описываемыми формулами x2+y2=1, x2+y2=9.

Тогда условие попадания точки во II и III квадранты будет выглядеть следующим образом:

x2+y2>=1&& x2+y2<=9&&&&x<=0.

Четверть IV:

Область формируется двумя прямоугольниками. Точка может попадать либо в первый прямоугольник, либо во второй.

Условие попадания точки в IV квадрант будет выглядеть следующим образом:

(x>=0&&x<=1&&y<=-1&&y>=-3)|| (x>=1&&x<=3&&y<=0&&y>=-3) .

*Алгоритм. Нарисовать блок-схему самостоятельно.*

*Программа.*

class Program

{

static void Main(string[] args)

{

double x=0, y=0;

bool ok=true;

do

{

try

{

Console.WriteLine("Введите кординату точки х");

x = Convert.ToDouble(Console.ReadLine());

ok = true;

}

catch (FormatException)

{

Console.WriteLine("Направильный ввод, введите вещественное число!");

ok = false;

}

}

while (!ok);

do

{

try

{

Console.WriteLine("Введите кординату точки y");

y = Convert.ToDouble(Console.ReadLine());

ok = true;

}

catch (FormatException)

{

Console.WriteLine("Направильный ввод, введите вещественное число!");

ok = false;

}

}

while (!ok);

if ((y >= -x + 1 && y <= 2.0 / 3.0 \* x + 2 && x >= 0 && y >= 0) || (Math.Pow(x, 2) + Math.Pow(y, 2) >= 1 && Math.Pow(x, 2) + Math.Pow(y, 2) <= 9 && x <= 0) ||(x >= 0 && x <= 1 && y <= -1 && y >= -3) || (x >= 1 && x <= 2 && y <= 0 && y >= -3))

Console.WriteLine("Точка попадает в заданную область");

else

Console.WriteLine("Точка не попадает в заданную область");

}

}

*Тесты:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T1 | x=1,y=0.5 | попадает | попадает | 1 четверть |
| T2 | x=-1,y=0.5 | попадает | попадает | 2 четверть |
| T3 | x=-1,y=-0.5 | попадает | попадает | 3 четверть |
| T4 | х=-0,5, у=-2 | попадает | попадает | 4 четверть |
| T5 | х=1,5, у=-0,5 | попадает | попадает | 4 четверть |
| T6 | х=0, у=0 | не попадает | не попадает | центр координат |
| T7 | х=2, у=3 | не попадает | не попадает | 1 четверть |
| T8 | х=-3, у=3 | не попадает | не попадает | 2 четверть |
| T9 | х=-3, у=-3 | не попадает | не попадает | 3 четверть |
| T10 | х=3, у=-3 | не попадает | не попадает | 4 четверть |
| T11 | х=1, у=0 | попадает | попадает | граница |
| T12 | х=-1, у=0 | попадает | попадает | граница |
| T13 | х=1, у=0 | попадает | попадает | граница |
| T14 | х=-1, у=0 | попадает | попадает | граница |

*Проверить полноту тестов по критериям черного и белого ящиков самостоятельно.*

### 5.2. Программирование арифметических циклов

Для арифметического цикла заранее известно сколько раз выполняется тело цикла.

**Задача №2.** Дана последовательность целых чисел из n элементов. Найти среднее арифметическое этой последовательности.

*Анализ.*

*Исходные данные:*

size – количество чисел,

number1, number2,…numberN – последовательность целых чисел.

*Результат:*

srednee – среднее арифметическое, вещественное число.

*Анализ решения.*

Используется цикл с параметром, т.к. известно количество чисел.

Инициализация: i=1, s=0;

Условие: i<=size.

Коррекция: i++;

Тело цикла: ввод number; s+=number.

*Классы входных данных:*

1. size>0, результат – среднее арифметическое последовательности чисел.
2. size<=0, результат – неправильный ввод size.
3. size – не целое число, результат – неправильный ввод size.
4. number – целое число, результат – среднее арифметическое последовательности чисел.
5. number – не целое число, результат – неправильный ввод number.

*Классы выходных данных:*

1. Число.
2. Неправильный ввод size.
3. Неправильный ввод number.

*Алгоритм. Нарисовать блок-схему самостоятельно.*

*Программа.*

class Program

{

static void Main(string[] args)

{

int size = 0;

bool ok=true;

do

{

try

{

Console.WriteLine("Введите количество чисел");

size = Convert.ToInt32(Console.ReadLine());

ok = true;

}

catch (FormatException)

{

Console.WriteLine("Неправильный ввод, введите целое число!");

ok = false;

}

if(size<=0)

{

Console.WriteLine("Число должно быть больше 0!");

ok=false;

}

}

while(!ok);

double srednee=0;

int number;

for(int i=0;i<size;i++)

{

try

{

Console.WriteLine("Введите {0} число",i+1);

number = Convert.ToInt32(Console.ReadLine());

srednee += number;

}

catch (FormatException)

{

Console.WriteLine("Неправильный ввод, введите целое число!");

i--;

}

}

srednee /= size;

Console.WriteLine("Среднее арифметическое последовательности равно: " + srednee);

}

}

*Тесты*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T1 | Size=0 | ошибка | ошибка | 0 раз |
| T2 | Size=1, number=2 | Srednee=2 | Srednee=2 | 1 раз |
| T3 | Number=0,5 | ошибка | ошибка |  |
| T4 | Size=5, number=1,2,3,4,5 | Srednee=3 | Srednee=3 | Результат целое число |
| T5 | Size=2, number=1,2 | Srednee=1,5 | Srednee=1,5 | Результат вещественное число |
| T6 | Size=2, number=1.2 | ошибка | ошибка | number – не целое |

*Проверить полноту тестов по критериям черного и белого ящиков самостоятельно.*

**Задача №3.** Найти значение S=1+2+3+4+. . . +N

*Анализ, алгоритм – самостоятельно.*

class Program

{

static void Main(string[] args)

{

int size = 0;

bool ok=true;

do

{

try

{

Console.WriteLine("Введите количество чисел");

size = Convert.ToInt32(Console.ReadLine());

ok = true;

}

catch (FormatException)

{

Console.WriteLine("Направильный ввод, введите целое число!");

ok = false;

}

}

while(!ok);

int summa=0;

for (int number = 1; number <= size; number++)

summa += number;

Console.WriteLine("Сумма {0} первых целых чисел равна={1}",size, summa);

}

}

*Тесты*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T1 | Size=0 | 0 | 0 | 0 раз |
| T2 | Size=1 | summa=1 | summa=1 | 1 раз |
| T3 | Size=5 | summa=15 | summa=15 | 5 раз |

*Проверить полноту тестов по критериям черного и белого ящиков самостоятельно.*

### 5.3. Программирование итерационных циклов

Для итерационного цикла должно быть известно условие выполнения цикла. При использовании цикла с предусловием (while) тело цикла может не выполняться ни разу, если сразу же не выполняется условие цикла. При использовании цикла с постусловием тело цикла будет выполнено хотя бы один раз. И в том, и в другом случае используется условие выполнения цикла.

**Задача №5.** Дана последовательность целых чисел, за которой следует 0. Найти минимальный элемент этой последовательности.

Для решения этой задачи используем цикл с предусловием, т. к. ноль не входит в последовательность и его не надо обрабатывать при поиске минимального значения.

*Анализ.*

*Исходные данные:*

number1, number2,…,0 – последовательность целых чисел.

*Результат:*

min – минимальное значение, целое число.

Используется цикл с условием, т.к. известен признак конца последовательности 0.

Инициализация: ввод number; min=number;

Условие: number!=0.

Коррекция: ввод number;

Тело цикла: ввод number; сравнение min и number.

*Классы входных данных:*

1. number – целые числа положительные, результат – минимальное число>0.
2. number – не целое число, результат – неправильный ввод number.
3. number – целые числа отрицательные, результат – минимальное число<0.
4. number – целые числа положительные и отрицательные, результат – минимальное число<0.
5. number – целые числа положительные и/или отрицательные, несколько одинаковых чисел, которые являются минимальными, результат – первое минимальное число.

*Классы выходных данных:*

1. Минимальное число.
2. Последовательность пустая.
3. Неправильный ввод числа последовательности.

*Алгоритм. Нарисовать блок-схему самостоятельно.*

*Программа.*

class Program

{

static void Main(string[] args)

{

int number=0, min=0,i=0;

bool ok = false;

while (!ok)

{

try

{

Console.WriteLine("Введите 1-e число");

number = Convert.ToInt32(Console.ReadLine());

ok = true;

}

catch (FormatException)

{

Console.WriteLine("Нe правильный ввод, введите целое число!");

ok = false;

}

}

min = number;

while (number != 0)

{

try

{ i++;

if (min > number) min = number;

Console.WriteLine("Введите {0} число", i+1);

number = Convert.ToInt32(Console.ReadLine());

}

catch (FormatException)

{

Console.WriteLine("Нe правильный ввод, введите целое число!");

i--;

}

}

if (i == 0) Console.WriteLine("Последовательность пустая!");

else Console.WriteLine("Min=" + min);

}

}

Для решения этой же задачи можно написать программу с использованием цикла с постусловием.

class Program

{

static void Main(string[] args)

{

int number=0, min=0,i=0;

do

{

try

{

i++;

Console.WriteLine("Введите {0} число", i );

number = Convert.ToInt32(Console.ReadLine());

if (number == 0) break;

if (i == 1) min = number;

else

if (min > number) min = number;

}

catch (FormatException)

{

Console.WriteLine("Нeправильный ввод, введите целое число!");

i--;

}

} while (true);

if (min == 0) Console.WriteLine("Последовательность пустая!");

else Console.WriteLine("Min=" + min);

}

}

*Тесты*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T1 | 0 | Последовательность пустая | Последовательность пустая | 0 раз |
| T2 | 1,0 | Min=1 | Min=1 | 1 раз |
| T3 | 1,2,3,0 | Min=1 | Min=1 | 3 разa |
| T4 | 3,2,1,0 | Min=1 | Min=1 |  |
| T5 | 2,1,3,0 | Min=1 | Min=1 |  |
| T6 | a | Нeправильный ввод, введите целое число! | Нeправильный ввод, введите целое число! |  |
| T7 | 1,1,2,0 | Min=1 | Min=1 | Два мин. |
| T8 | -1,-2,-3,0 | Min=-3 | Min=-3 | Отрицат. числа |
| T9 | 1,2,-3,0 | Min=-3 | Min=-3 | Отрицат. и положит. числа |

*Проверить полноту тестов по критериям черного и белого ящиков самостоятельно.*

**Задача №6**. Найти сумму чисел Фибоначчи, меньших заданного числа Q. Числа Фибоначчи – это последовательность чисел: 1, 1, 2, 3, 5, 8, 13, …., т. е. каждое следующее число – это сумма двух предыдущих.

*Анализ.*

*Исходные данные:*

Q – целое число.

*Результат:*

Последовательность целых чисел.

*Анализ решения.*

Используется цикл с условием.

Инициализация: a=1, b=1, c=a+b, s=a+b+c;

Условие цикла c<Q.

Коррекция: a=b,b=c, c=a+b;

Тело цикла: s+=c;

*Классы входных данных.*

1. Q<=0, результат – сообщение об ошибке.

2. Q=1, результат – одно число 1.

3. Q=2, результат – два числа 1,1.

4. Q>2, результат – последовательность целых чисел.

5. Q – не является целым числом.

*Классы выходных данных:*

1. Последовательность чисел Фибоначчи.

2. Q<=0, нельзя сформировать последовательность.

3. Q – не является целым числом.

*Алгоритм. Нарисовать блок-схему самостоятельно.*

*Программа.*

class Program

{

static void Main(string[] args)

{

int fib1 = 1, fib2 = 1, fib3 = fib1 + fib2;

int Q=0;

int sum = 0;

bool ok = true;

do

{

try

{

Console.WriteLine("Введите целое число Q");

Q = Convert.ToInt32(Console.ReadLine());

ok = true;

}

catch (FormatException)

{

Console.WriteLine("Неправильный ввод, введите целое число!");

ok = false;

}

}

while (!ok);

if(Q<=0)

{

Console.WriteLine("Q не может быть меньше или равно 0!");

return;

}

if (Q == 1)

{

sum = 0;

}

else

if (Q == 2)

{

sum = fib1 + fib2;

}

else

{

sum = fib1 + fib2;

while (fib3 < Q)

{

sum += fib3;

fib1 = fib2;

fib2 = fib3;

fib3 = fib1 + fib2;

}

}

Console.WriteLine("Сумма чисел Фибоначчи меньших числа {0} равна {1}", Q, sum);

}

}

}

*Тесты*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T1 | Q=0 | Q не может быть меньше или равно 0 | Q не может быть меньше или равно 0 |  |
| T2 | Q=1 | Сумма чисел Фибоначчи меньших числа 1 равна 0 | Сумма чисел Фибоначчи меньших числа 1 равна 0 |  |
| T3 | Q=2 | Сумма чисел Фибоначчи меньших числа 2 равна 2 | Сумма чисел Фибоначчи меньших числа 2 равна 2 |  |
| T4 | Q=5 | Сумма чисел Фибоначчи меньших числа 5 равна 7 | Сумма чисел Фибоначчи меньших числа 5 равна 7 |  |
| T5 | Q=a | Нeправильный ввод, введите целое число! | Нeправильный ввод, введите целое число! |  |

*Проверить полноту тестов по критериям черного и белого ящиков самостоятельно.*

### 5.4. Программирование вложенных циклов

Тело цикла может содержать любые операторы, в том числе и другие циклы. Оператор цикла, который содержится в теле другого цикла, называется вложенным.

**Задача №7**: Напечатать N простых чисел.

*Анализ.*

*Исходные данные:*

N – целое число.

*Результат*: последовательность простых чисел.

Разделим эту задачу на две подзадачи:

* + 1. определить является ли число простым;
    2. напечатать N чисел, удовлетворяющих заданному условию.

Простым называется число, которое делится только само на себя и на единицу (сама единица простым числом не является). Тогда, чтобы определить является число простым или нет, нужно проверить есть ли у него другие делители. Для этого будем делить число number на все числа от 2 до number включительно, используя цикл с постусловием. Если number разделится на divisor без остатка только, когда d станет равно number, значит, число простое:

int divisor = 1;

do

{

divisor++;

}

while (number % divisor != 0);

if (number == divisor)

Console.WriteLine("Yes");

else Console.WriteLine("No");

Для решения второй подзадачи, нам нужно перебирать все числа, начиная с 2, и печатать только те, которые являются простыми. При этом, нужно подсчитывать напечатанные числа. Выполнение цикла закончится, когда будет напечатано size чисел.

int number = 2;

for(int count=1; count<=size;)

{

int divisor = 1;

if (number простое число)

{

Console.WriteLine(number + " ");

count++;

}

number++;

}

Объединив эти два фрагмента вместе, получим решение поставленной задачи.

*Классы входных данных:*

1. size<=0, результат "Пустая последовательность!"
2. size>0, результат сформированная последовательность простых чисел.
3. number – не целое число, результат – неправильный ввод number.

*Программа.*

class Program

{

static void Main(string[] args)

{

int size=0;

bool ok = true;

do

{

try

{

Console.WriteLine("Введите количество чисел");

size = Convert.ToInt32(Console.ReadLine());

ok = true;

}

catch (FormatException)

{

Console.WriteLine("Направильный ввод, введите целое число!");

ok = false;

}

}

while (!ok);

if (size <= 0)

{

Console.WriteLine("Пустая последовательность!");

return;

}

int number = 2;

for(int count=1; count<=size;)

{

int divisor = 1;

do

{

divisor++;

}

while (number % divisor != 0);

if (number == divisor)

{

Console.WriteLine(number + " ");

count++;

}

number++;

}

}

}

*Тесты*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T1 | size=0 | Пустая последовательность | Пустая последовательность |  |
| T2 | size=1 | 2 | 2 |  |
| T3 | size=1 | 2,3,5,7,11 | 2,3,5,7,11 |  |
| T5 | size =a | Нeправильный ввод, введите целое число! | Нeправильный ввод, введите целое число! |  |

*Проверить полноту тестов по критериям черного и белого ящиков самостоятельно.*