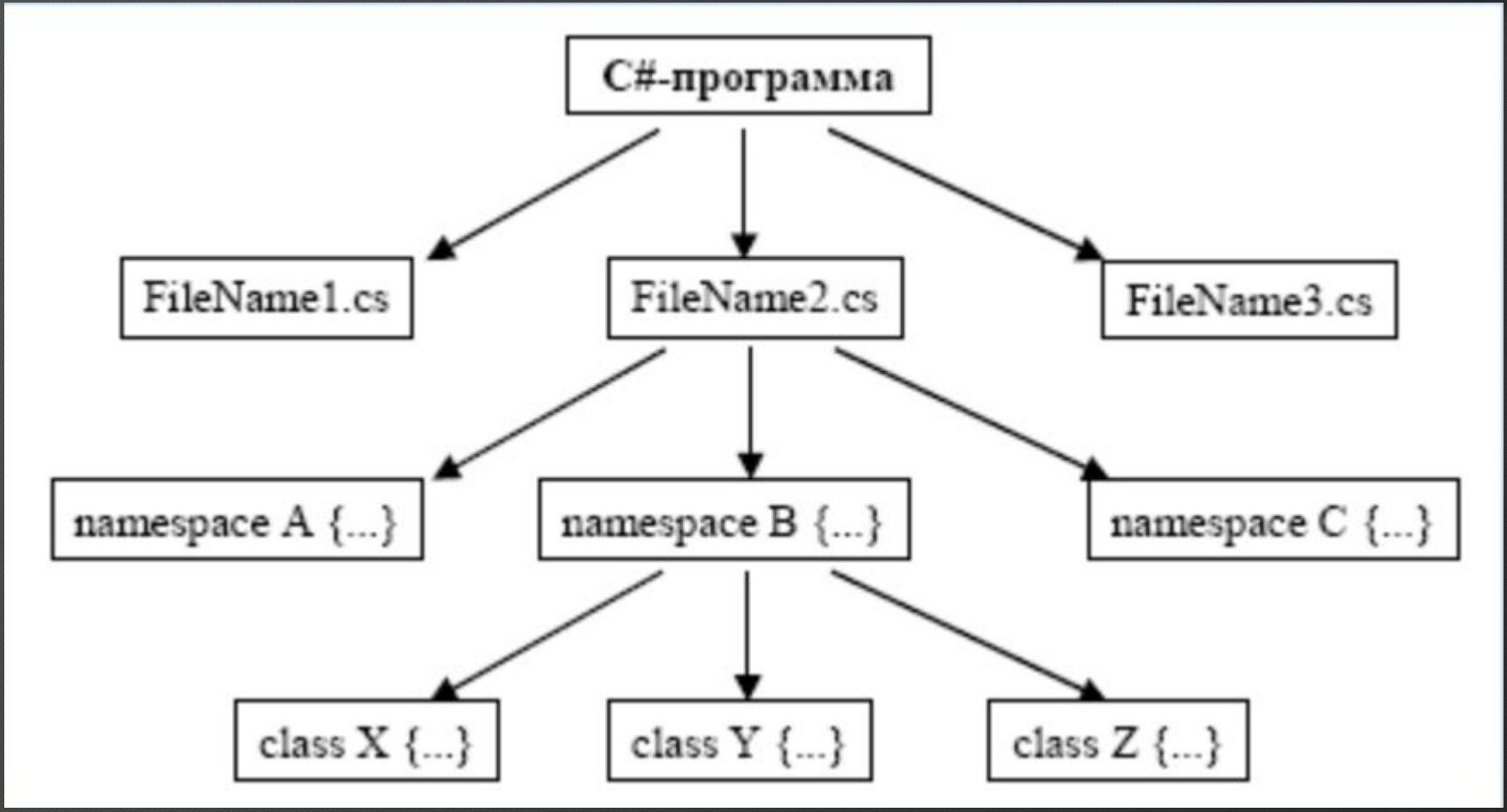
Язык программирования C#

Структура программы

```
using System;
using System.Collections.Generic;
using System.Text;
    namespace ConsoleApplication1
    {
        class Program
        {
            static void Main(string[] args)
            {
                Console.WriteLine("Введите Ваше имя");
                string name;
                name = Console.ReadLine();
                if (name == "")
                    Console.WriteLine("Здравствуй, мир!");
                else
                    Console.WriteLine("Здравствуй, " + name + "!");
            }
        }
    }
```

Структура проекта C



Элементы языка C

Алфавит языка C

- Прописные и строчные латинские буквы и знак подчеркивания;
- Арабские цифры от 0 до 9;
- Специальные знаки: { } , | [ ] ( ) + - / % * . \ ' : ; & ? < > = ! # ^
- Пробельные символы.

Идентификаторы языка C

- В идентификаторе могут быть использованы латинские буквы, цифры и знак подчеркивания.
- Прописные и строчные буквы различаются.
- Первым символом должна быть буква или знак подчеркивания (но не цифра).
- Пробелы в идентификаторах не допускаются.

Элементы языка C

- Ключевые (зарезервированные) слова – это слова, которые имеют специальное значение для компилятора. Их нельзя использовать в качестве идентификаторов.

  true, false, int, float, switch … и.т.д.

- Знаки операций – это один или несколько символов, определяющих действие над операндами. Операции делятся на унарные, бинарные и тернарную по количеству участвующих в этой операции операндов. + - * / % < > >= <= == != << >> ! & | && || * ++ -- … и.т.д.
- Константы – неизменяемые величины.
- Разделители – скобки, точка, запятая пробельные символы.

Константы в C

Константы в c#

- Константа – это лексема, представляющая изображение фиксированного числового, строкового или символьного значения.
- Константы делятся на 5 групп:
- целые;
- вещественные (с плавающей точкой);
- перечислимые;
- символьные;
- строковые.
- Целые константы могут быть десятичными, восьмеричными (начинаются с 0) и шестнадцатеричными (начинаются с 0x). 10, 0xFF, 016
- Вещественные константы могут иметь две формы представления: с фиксированной точкой ([цифры].[цифры]). с плавающей точкой ([цифры][.][цифры]E|e[+|-][цифры]). 2.5, 0.5E10

Константы в c#

- Строковая константа – это последовательность символов, заключенная в кавычки. Внутри строк также могут использоваться управляющие символы. "\nНовая строка", "\n\" Алгоритмические языки программирования\"".
- Перечислимые константы вводятся с помощью ключевого слова enum. Это обычные целые константы, которым приписаны уникальные и удобные для использования обозначения. Пример: enum{ten=10,three=3,four,five,six}; enum{Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday};
- Символьные константы – это один или два символа, заключенные в апострофы.

Типы данных в C

Типы данных в C#

- Типы C Sharp можно разделить на простые и составные. К простым типам относят типы, которые характеризуются одним значением.

int	(целый)
char	(символьный)
string	(строковый)
bool	(логический)
float	(вещественный)
double	(вещественный с двойной точностью)

- Существует 4 спецификатора типа, уточняющих внутреннее представление и диапазон стандартных типов:

short	(короткий)
long	(длинный)
signed	(знаковый)
unsigned	(беззнаковый)

Тип int в C#

- Значениями этого типа являются целые числа.
- В 16-битных операционных системах под него отводится 2 байта, в 32-битных – 4 байта.
- Если перед int стоит спецификатор short, то под число отводится 2 байта, а если спецификатор long, то 4 байта. От количества отводимой под объект памяти зависит множество допустимых значений, которые может принимать объект: short int – занимает 2 байта, диапазон –32768 … +32767; long int – занимает 4 байта, диапазон –2 147 483 648 … +2 147 483 647.
- Тип int совпадает с типом short int на 16-разрядных ПК и с типом long int на 32-разрядных ПК.
- Модификаторы signed и unsigned также влияют на множество допустимых значений, которые может принимать объект: unsigned short int – занимает 2 байта, диапазон 0 … 65536; unsigned long int – занимает 4 байта, диапазон 0 … +4 294 967 295.

Тип char в C#

- Значениями этого типа являются элементы конечного упорядоченного множества символов. Каждому символу ставится в соответствие число, которое называется кодом символа.
- Под величину символьного типа отводится 1 байт. Тип char может использоваться со спецификаторами signed и unsigned. signed char - диапазон от –128 до 127. unsigned char - диапазон от 0 до 255.
- Для кодировки используется код ASCii (American Standard Code for international interchange).
- Символы с кодами от 0 до 31 относятся к служебным и имеют самостоятельное значение только в операторах ввода-вывода.
- Величины типа char также применяются для хранения чисел из указанных диапазонов.

Типы bool в C#

- Тип bool называется логическим. Его величины могут принимать значения true (истина) и false (ложь).
- Внутренняя форма представления false – 0 (ноль), любое другое значение интерпретируется как true.

Типы плавающей точкой в C#

- Внутреннее представление вещественного числа состоит из 2 частей: мантиссы и порядка.
- Мантисса – это численное значение со знаком, порядок – это целое со знаком, определяющее значимость мантиссы.
- Длина мантиссы определяет точность числа, а длина порядка его диапазон. 1.00000e+001//представление числа 10
- В IBM-совместимых ПК величины типа float занимают 4 байта, из которых один разряд отводится под знак мантиссы, 8 разрядов под порядок и 24 – под мантиссу.
- Величины типа double занимают 8 байтов, под порядок и мантиссу отводятся 11 и 52 разряда соответственно.

Тип void в C#

- К основным типам также относится тип void. Множество значений этого типа – пусто.
- Невозможно создать переменную этого типа, но можно использовать указатель.

```
void a; // нельзя создать переменную…
void *ptr; // но можно объявлять указатель.
```

Переменные в C#

Переменные в C#

- Переменная в C Sharp – именованная область памяти, в которой хранятся данные определенного типа. У переменной есть имя и значение. Имя служит для обращения к области памяти, в которой хранится значение.
- Перед использованием любая переменная должна быть описана.

```
int a;
float x;
```

- При описании можно присвоить переменной начальное значение (инициализация):

```
int a = 10;
float b = 20.5;
```

Переменные в C#

- Областью действия переменной a является вся программа, кроме тех строк, где используется локальная переменная a. Переменные b и c – локальные, область их видимости – блок. Время жизни различно: память под b выделяется при входе в блок (т. к. по умолчанию класс памяти auto), освобождается при выходе из него. Переменная c (static) существует в пределах функции, внутри которой она определена и сохраняет своё значение и при последующих вызовах этой функции.
- Имя переменной должно быть уникальным в своей области действия.

Выражения в C#

Выражения в C#

- Из констант, переменных, разделителей и знаков операций можно конструировать выражения. Каждое выражение представляет собой правило вычисления нового значения.
- Если выражение формирует целое или вещественное число, то оно называется арифметическим. Пара арифметических выражений, объединенная операцией сравнения, называется отношением.
- Если отношение имеет ненулевое значение, то оно – истинно, иначе – ложно.

```
a+b+64 // арифметическое выражение
(c-4) > (d*e) // отношение
```

Операторы языка C#

Типы операторов языка C#

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

- ❑ составные операторы;
- ❑ операторы выбора;
- ❑ операторы циклов;
- ❑ операторы перехода.

Оператор "выражение"

- Любое выражение, заканчивающееся точкой с запятой, рассматривается как оператор, выполнение которого заключается в вычислении этого выражения.

```
i++;
a += 2;
x = a+b;
```

Составные операторы

- К составным операторам относят собственно составные операторы и блоки. В обоих случаях это последовательность операторов, заключенная в фигурные скобки.
- Блок отличается от составного оператора наличием определений в теле блока.

```
{
   n++;
   summa += n;
}
{
   int n = 0;
   n++;
   summa += n;
}
```

— Составной оператор

— Блок

Условный оператор

- Условный оператор имеет полную и сокращенную форму.
- if (выражение-условие) оператор1; // сокращенная форма
- if (выражение-условие) оператор1;// полная форма
- else оператор2;
- Если значение выражения-условия отлично от нуля, то выполняется оператор1, иначе выполняется оператор2.

Условный оператор. Пример.

```
if (d>=0)
{
x1=(-b-sqrt(d))/(2*a);
x2=(-b+sqrt(d))/(2*a);
Console.Write("\nx1=");
Console.Write(X1);
Console.Write("\nx2=");
Console.Write(X2);
}
else Console.Write("\nРешения нет“);
```

Оператор выбора

- Переключатель определяет множественный выбор.

```
switch (выражение)
{
case константа1: оператор1;
case константа2: оператор2; . . . . . .
[default: операторы;]
}
```

- 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.

Оператор выбора. Пример.

```
void main()
{
 int i;
 Console.Write("\nEnter the number“);
 i= Convert.ToInt16(Console.ReadLine());
 switch(i)
 {
  case 1: Console.Write("\n the number is one“); break;
  case 2: Console.Write("\n2*2= {0:D}“, i*i); break;
  case 3: Console.Write("\n3*3= {0:D}“, i*i); break;
  case 4: Console.Write("\n {0:D} is very beautiful !“,i); break;
  default: Console.Write("\n The end of work“); break;
 }
}
```

Цикл с предусловием

- while (выражение-условие) оператор;
- Если выражение-условие истинно, то тело цикла выполняется до тех пор, пока выражение-условие не станет ложным.

```
while (a!=0)
{
 a= Convert.ToInt16(Console.ReadLine());
 s+=a;
}
```

Цикл с постусловием

```
do
оператор;
while (выражение-условие);
```

- Тело цикла выполняется до тех пор, пока выражение-условие истинно.

```
do
{
  a= Convert.ToInt16(Console.ReadLine());
  s+=a;
}
while(a!=0);
```

Цикл с параметром

- for (выражение_1;выражение-условие;выражение_3) оператор;
- Выражение_1 – задает начальные условия для цикла (инициализация).
- Выражение-условие определяет условие выполнения цикла, если оно не равно 0, цикл выполняется, а затем вычисляется значение выражения_3.
- Выражение_3 – задает изменение параметра цикла или других переменных (коррекция).
- Выражение_1 и выражение_3 могут состоять из нескольких выражений, разделенных запятыми.
- Любое выражение может отсутствовать, но разделяющие их " ; " должны быть обязательно.

Цикл с параметром. Пример.

- Уменьшение параметра:

```
for (int n=10; n>0; n--)
{
   оператор;
}
```

- Проверка условия отличного от того, которое налагается на число итераций:

```
for (num=1; num*num*num<216; num++)
{
   оператор;
}
```

Цикл с параметром. Пример

- Коррекция с помощью умножения:

```
for ( d=100.0; d<150.0;d*=1.1)
{
   оператор;
}
```

- Коррекция с помощью арифметического выражения:

```
for (x=1; y<=75; y=5*(x++)+10)
{
    оператор;
}
```

Операторы перехода

- break – оператор прерывания цикла.
- continue – переход к следующей итерации цикла. Используется, когда тело цикла содержит ветвления.
- goto <метка> – передает управление оператору, который содержит метку.

Оператор break

```
{
   оператор;
   if (<выражение_условие>) break;
   оператор;
}
```

- Оператор break целесообразно использовать, когда условие продолжения итераций надо проверять в середине цикла.
- Пример: найти сумму чисел, числа вводятся с клавиатуры до тех пор, пока не будет введено 100 чисел или 0.

```
for(s=0, i=1; i<100; i++)
{
   x= Convert.ToInt16(Console.ReadLine());
   if(!x) break;// если ввели 0, то суммирование
   // заканчивается.
   s+=x;
}
```

Оператор continue

- Пример: найти количество и сумму положительных чисел.

```
for(k=0, s=0, x=1; x!=0;)
{
   x= Convert.ToInt16(Console.ReadLine());
   if (x<=0) continue;
   k++; s+=x;
}
```

Оператор goto

- В теле той же функции должна присутствовать конструкция: <метка>: оператор;
- Применение goto нарушает принципы структурного и модульного программирования.
- Нельзя передавать управление внутрь операторов if, switch и циклов. Нельзя переходить внутрь блоков, содержащих инициализацию, на операторы, которые стоят после инициализации.

Оператор return

- return – оператор возврата из функции. Он всегда завершает выполнение функции и передает управление в точку ее вызова. Вид оператора:
- return [выражение];

Массивы в C#

Массивы в C

- В языке C Sharp, кроме базовых типов, разрешено вводить и использовать производные типы, полученные на основе базовых.
- Стандарт языка определяет три способа получения производных типов:
  - массив элементов заданного типа;
  - указатель на объект заданного типа;
  - функция, возвращающая значение заданного типа.
- Массив – это упорядоченная последовательность переменных одного типа. Каждому элементу массива отводится одна ячейка памяти. Элементы одного массива занимают последовательно расположенные ячейки памяти.
- Все элементы имеют одно имя – имя массива и отличаются индексами – порядковыми номерами в массиве.
- Количество элементов в массиве называется его размером. Чтобы отвести в памяти нужное количество ячеек для размещения массива, надо заранее знать его размер. Резервирование памяти для массива выполняется на этапе компиляции программы.

Массивы в C#

- Массивы определяются следующим образом: тип имя[размер];
- Примеры:

```
int[] a=new int[100];
float[] b=new float[20];
char[] c=new char[32];
```

- Элементы массива всегда нумеруются с 0.
- Чтобы обратиться к элементу массива, надо указать имя массива и номер элемента в массиве (индекс):

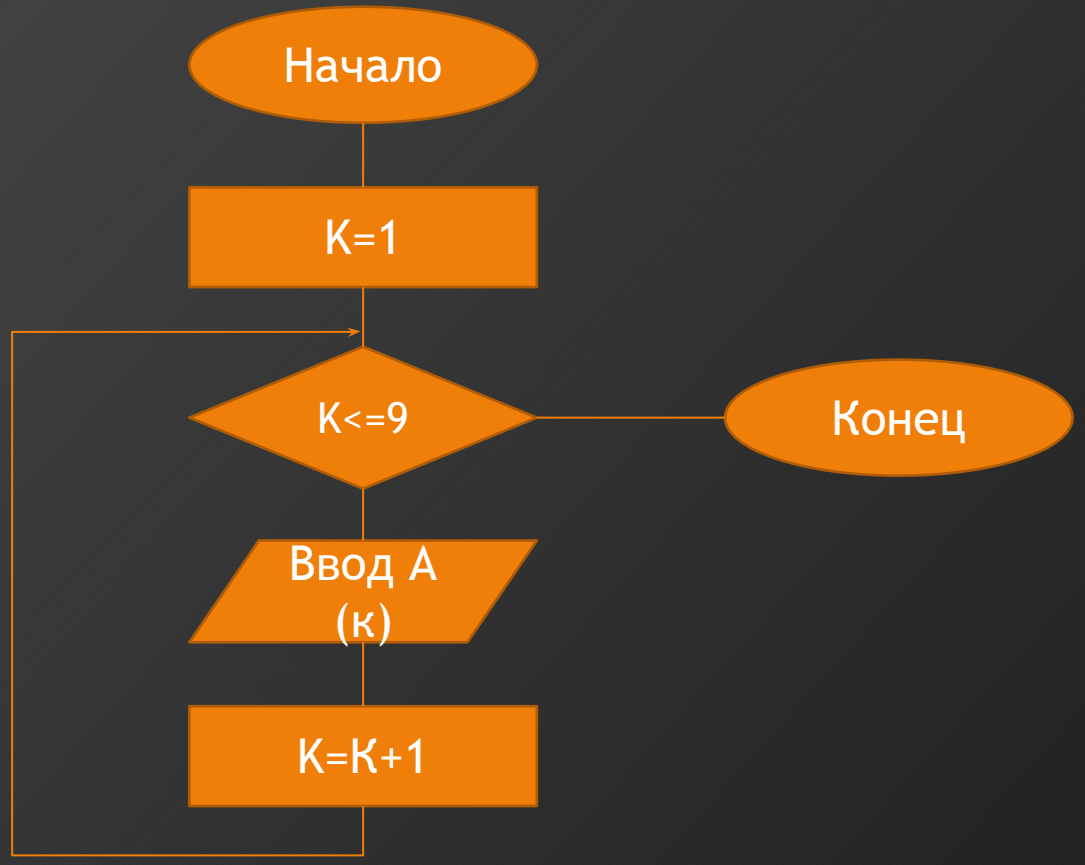
```
a[55] // индекс задается как константа
a[i] // индекс задается как переменная
a[2*i] // индекс задается как выражение
```

Массивы в C#

- Элементы массива можно задавать при его определении:
- int[] a={1,2,3,4,5,6,7,8,9,10};
- Длина массива может вычисляться компилятором по количеству значений, перечисленных при инициализации:
- int[] a={1,2,3,4,5}; // длина массива – 5 элементов.

Ввод элементов

- В этом циклическом алгоритме условие выхода из цикла определяется значением специальной переменной К, которая называется счетчиком элементов массива А, эта же переменная К определяет количество итераций циклического алгоритма ввода элементов массива. На каждом шаге итерации переменная К(значение номера элемента массива А) изменяется на 1, то есть происходит переход к новому элементу



Ввод элементов

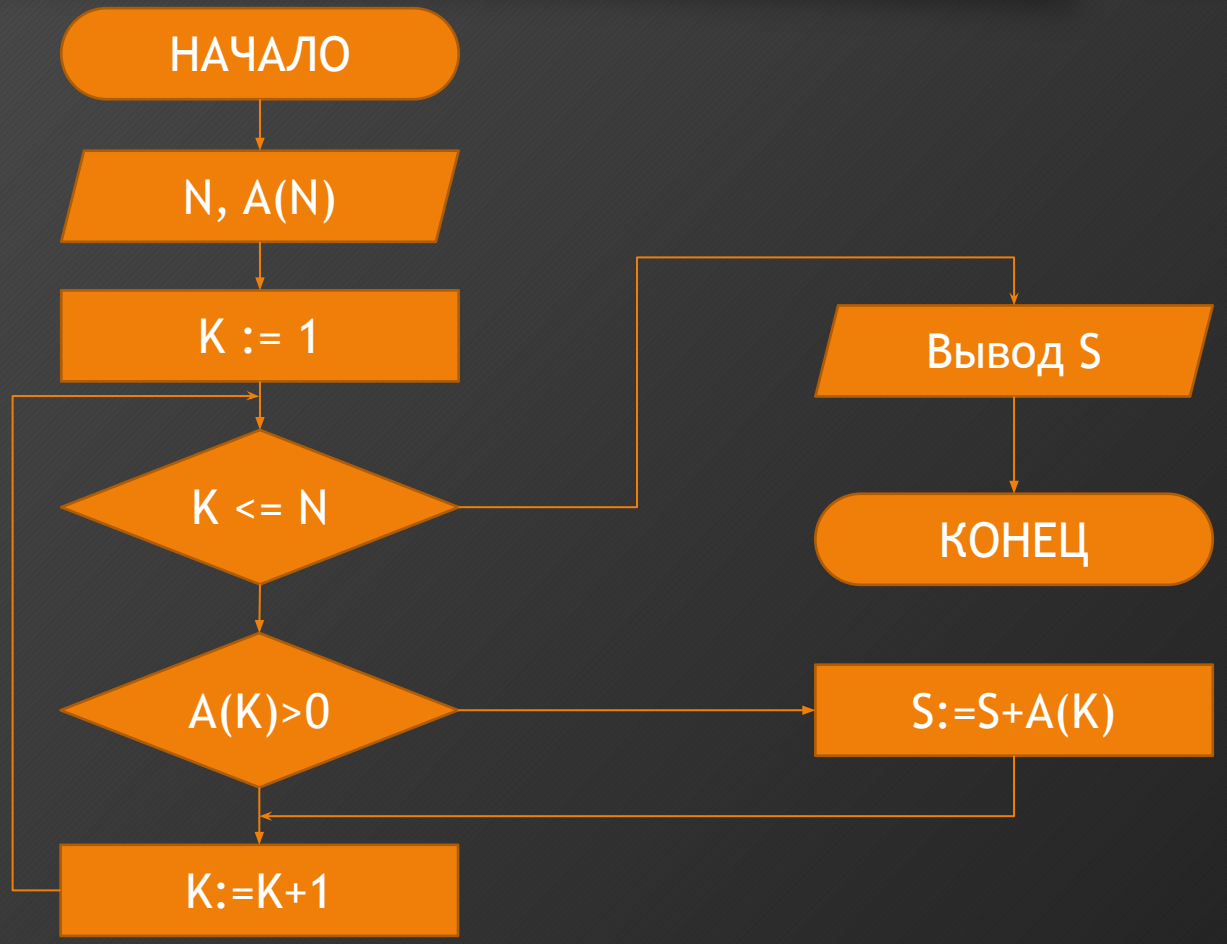
```
void main()
{
int n;
Console.Write("\nEnter the size of array: ");
n=Convert.ToInt16(Console.ReadLine());
int[] a=new int[n];
for(int i=0; i<n; i++)
a[i]= Convert.ToInt16(Console.ReadLine());
}
```

Формирование массива с помощью датчика случайных чисел

```
void main()
{
int n; Random rnd= new Random();
Console.Write("Введите размерность массива:");
n=Convert.ToInt16(Console.ReadLine());
int[] a=new int[n];
for(int i=0; i<n; i++)
{
a[i]= rnd.Next(100); Console.WriteLine(a[i]);
}
}
```

Сумма положительных элементов

- Пример. Составить алгоритм определения в одномерном числовом массиве A из N элементов суммы положительных элементов.
- Решение. В этом алгоритме переменная K - является счетчиком элементов массива, S - сумма элементов массива, она вычисляется по реккурентной формуле S=S+A(K).



Сумма положительных элементов

```
void main()
{
int n,s=0;
Console.Write(" Введите размерность массива:");
n=Convert.ToInt16(Console.ReadLine());
int[] a=new int[n];
for(int i=0; i<n; i++) a[i]= Convert.ToInt16(Console.ReadLine());
for(int i=0; i<n; i++) if (a[i]>0) s+= a[i]; Console.WriteLine(s);
}
```

Сумма элементов массива с четными индексами.

```
void main()
{
int n; Random rnd = new Random();
Console.Write("\nEnter the size of array: ");
n=Convert.ToInt16(Console.ReadLine());
int[ ] a=new int[n]; for (int i = 0; i < n; i++) // заполнение массива {
a[i] = rnd.Next(100);
Console.WriteLine(a[i]);
}
int summ = 0; // обнуляем сумму
// суммируются элементы с индексами 0,2,4 и т. д.
for (int i = 0; i < n; i+=2)
summ += a[i];
Console.Write("summ=");
Console.WriteLine(summ);
Console.ReadLine();
}
```

Поиск

Поиск - обнаружение нужного элемента в некотором наборе (структуре) данных.

- Элемент данных - это запись, состоящая из ключа (целое положительное число) и тела, содержащего некоторую информацию. Задача поиска состоит в том, чтобы обнаружить запись с нужным ключом.

Линейный поиск.

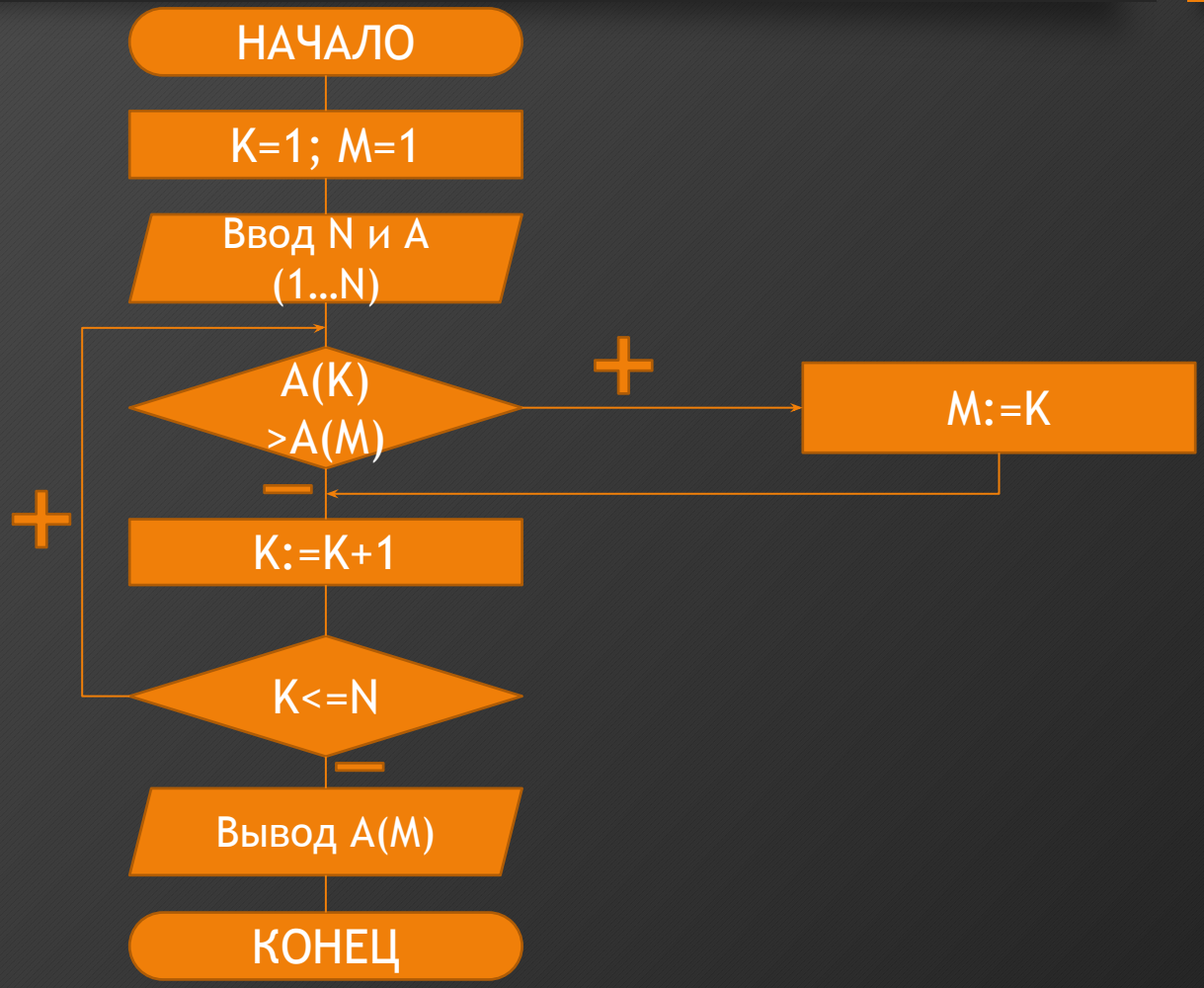
- Элементы проверяются последовательно, по одному, до тех пор, пока нужный элемент не будет найден. Для массива из N элементов требуется, в среднем, (N+1)/2 сравнений (вычислительная сложность Ocp(N)). Легко программиру-ется, подходит для коротких массивов.

Двоичный (бинарный) поиск.

- Применим, если массив заранее отсортирован (по возрастанию ключей). Ключ поиска сравнивается с ключом среднего элемента в массиве. Если значение ключа поиска больше, то та же самая операция повторяется для второй поло-вины массива, если меньше - то для первой. Операция повторяется до нахож-дения нужного элемента. На каждом шаге диапазон элементов в поиске умень-шается вдвое. Требуется, в среднем, (log2N+1)/2 сравнений (выч. сложность Ocp(log2N)). Применяется для поиска (многократного) в больших массивах.

Поиск максимального элемента

- Пример. Составить алгоритм поиска элемента с максимальным значением в одномерном массиве A(1..n).
- Решение. Введем обозначения K- текущий номер элемента, A[K] - текущее значение элемента массива, N количество элементов одномерного массива, M-номер максимального элемента массива, A[M] - значение максимального элемента массива.
- Основной идеей алгоритма является выполнение сравнения текущего элемента массива A[K] и элемента с максимальным значением A[M], определенным на предыдущем шаге итерации



Поиск максимального элемента

```
 void main()
{
      int n;
      Random rnd = new Random();
      Console.Write("\nEnter the size of array: ");
      n=Convert.ToInt16(Console.ReadLine());
      int[ ] a=new int[n]; // заполнение массива
      for (int i = 0; i < n; i++)
            {
            a[i] = rnd.Next(100);
            Console.WriteLine(a[i]);
            } // задаем начальное значение переменной max
      int k = 0; // сравниваем элементы массива с переменной max:
            for (int i = 0; i < n; i++)
                  if (a[i] > a[k])
            k = i;
            Console.Write("Max=");
            Console.WriteLine(a[k]);
            Console.ReadLine();
}
```

Сортировка

- Сортировка (упорядочение) - размещение элементов данных в возрастающем или убывающем порядке. При выборе метода сортировки необходимо учитывать число сортируемых элементов (N) и до какой степени элементы уже отсортированы.
- Критерии оценки метода сортировки: - количество необходимых операций сравнения в зависимости от числа элементов N, вычислительная сложность алгоритма характеризуется с помощью O-функции, аргументом которой является другая функция от N; - эффективность использования памяти

$$f = \frac{S(N)}{S(N)+\Delta S(N)},$$

- где S(N) - объем памяти, занимаемый элементами данных до сортировки, ΔS(N) - объем дополнительной памяти, требуемой в процессе сортировки.

Двумерные массивы в C#

- Примеры:

```
int[,] a=new int[100,100];
n1=10;n2=20;n3=5;
float[,,] b=new float[n1,n2,n3];
```

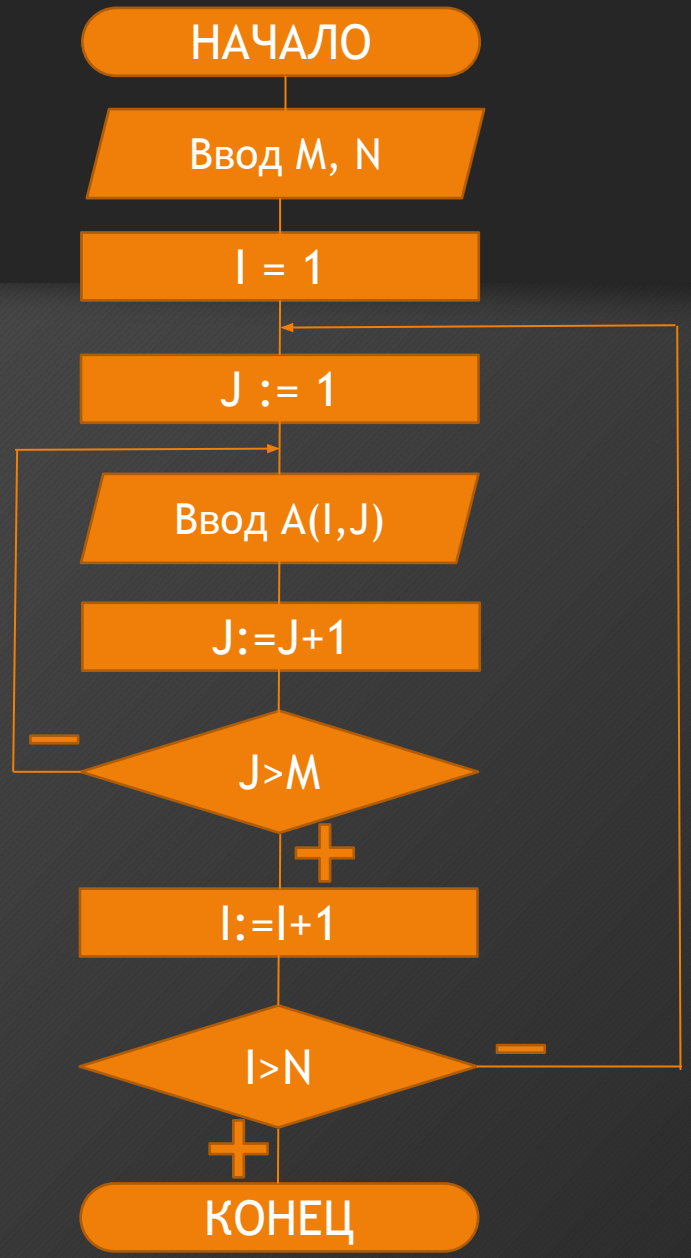
- Чтобы обратиться к элементу массива, надо указать имя массива и все его индексы:

```
a[5,5] // индексы задаются как константы
a[i,j] // индексы задаются как переменные
a[2*i,3*j] // индексы задаются как выражения
```

- В дальнейшем будем считать, что для двумерного массива A(N,M) в обозначении элемента A(i,j) первое значение i соответствует номеру строки и изменяется от1 до N, а j - номеру столбца и изменяется от 1 до M.

Ввод элементов(59)

- В отличие от одномерного массива, в котором использовался только один номер для определения местоположения элемента и требовался только один цикл для ввода элементов, в двумерном массиве для обработки элементов необходимы два вложенных друг в друга цикла. Внешний цикл предназначен для изменения номера строки i, а второй, внутренний, - для изменения номера столбца j в текущей строке i



ООП в C

Принципы объектно-ориентированного программирования:

- 1. Абстракция данных
  2. Наследование конкретных атрибутов объектов и функций оперирования объектами на основе иерархии
  3. Инкапсуляция (свойства и методы «спрятаны» внутри объекта)
  4. Полиморфизм (функции с возможностью обработки данных переменного типа)

Абстракция и методы ее моделирования

Вообще говоря, под абстракцией понимается выражение языка программирования, отличное от идентификатора.

Значение функции или переменной может быть присвоено абстракции и является значением последней.

Поведение абстракции заключается в приложении функции к аргументу.

Абстракция адекватно моделируется лямбдаисчислением (а именно, посредством операции абстракции).

Наследование и методы его моделирования

Вообще говоря, под наследованием понимается свойство производного объекта сохранять поведение (атрибуты и операции) базового (родительского).

В языках программирования понятие наследование означает применимость (некоторых) свойств или методов базового класса для классов, производных от него (а также для их конкретизаций).

Наследование моделируется (иерархическим) отношением частичного порядка и адекватно формализуется посредством:

1) фреймовой нотации Руссопулоса (N.D. Roussopulos);
2) диаграмм Хассе (Hasse)

Пример единичного наследования на C

```
class A { // базовый класс
int a;
public A() {...}
public void F() {...}
}
class B : A { // подкласс (наследует свойства
класса A, расширяет класс A)
int b;
public B() {...}
public void G() {...}
}
```

Понятие инкапсуляции в программировании

- Вообще говоря, под инкапсуляцией понимается доступность объекта исключительно посредством его свойств и методов.
- Таким образом, свойствами объекта (явно описанными или производными) возможно оперировать исключительно посредством его методов.
- Свойства инкапсуляции:
  - совместное хранение данных и функций;
  - сокрытие внутренней информации от пользователя;
  - изоляция пользователя от особенностей реализации

Понятие полиморфизма в программировании

Вообще говоря, под полиморфизмом понимается возможность оперировать объектами, не обладая точным знанием их типов.

Рассмотрим пример полиморфной функции:

```
void Poly(object o) {
Console.WriteLine(o.ToString());
}
```

а также вариантов ее использования:

```
Poly(25);
Poly(“John Smith”);
Poly(3.141592536m);
Poly(new Point(12,45));
```

Литература к лекции

- Т. Арчер "Основы C#", Русская редакция, 2001. 448 с.
- Э. Гуннерсон "Введение в C#", СПб.: Питер, 2001. 304 с.
- "Microsoft C# Language Specification", Microsoft Press, 2001. 412 p.
- J. Trupin "Sharp New Language. C# Offers the Power of C++ and Simplicity of Visual Basic", MSDN Magazine, September 2000.