СТРУКТУРА ПРОГРАММЫ, АБСТРАКЦИИ И КОНСТРУКЦИИ ЯЗЫКА

C# - ЛЕКЦИЯ 2

1. Структуру программы
2. Абстракции языка
3. Конструкции языка
4. Типы данных
5. Библиотеки и фреймворки
6. Алгоритмы
7. Практики и приёмы программирования (технология)

СТРУКТУРА ПРОГРАММЫ

Программа на C# состоит из набора `*.cs` файлов, содержащих объявления типов данных.

В корневой папке обычно находится файл `ИмяПроекта.csproj`

Результатом построения проекта (build) является сборка (assembly).

В коде программы могут использоваться типы из подключаемых сборок. Зависимости от сборок и проектов указываются в файле `.csproj`

Объявления типов обычно вложены в пространство имён.

Объявления типов могут быть вложены в объявления других типов.

Объявление типа в глобальном пространстве имён

```
class Foo
{

}
```

Объявление типа в пространстве имён Acme

```
namespace Acme
{
    class Foo
    {

    }
}
```

Объявление типа в пространстве имён Acme.Ordering

```
namespace Acme.Ordering
{
    class Order
    {

    }
}
```

Объявление типа в пространстве имён Acme.Ordering

```
namespace Acme
{
    namespace Ordering
    {
        class Order
        {

        }
    }
}
```

Вложенное объявление типа

```
class A
{
    class B
    {

    }
}
```

```
namespace Acme
{
    class A
    {
        class B {}

        class C
        {
            class D {}
        }
    }
}
```

АБСТРАКЦИИ ЯЗЫКА

АБСТРАКЦИИ ЯЗЫКА

- По данным
- По действиям

АБСТРАКЦИИ ПО ДАННЫМ

- Переменные
- Константы
- Поля типов

ПЕРЕМЕННЫЕ

Хранят значения, используемые в ходе выполнения программы

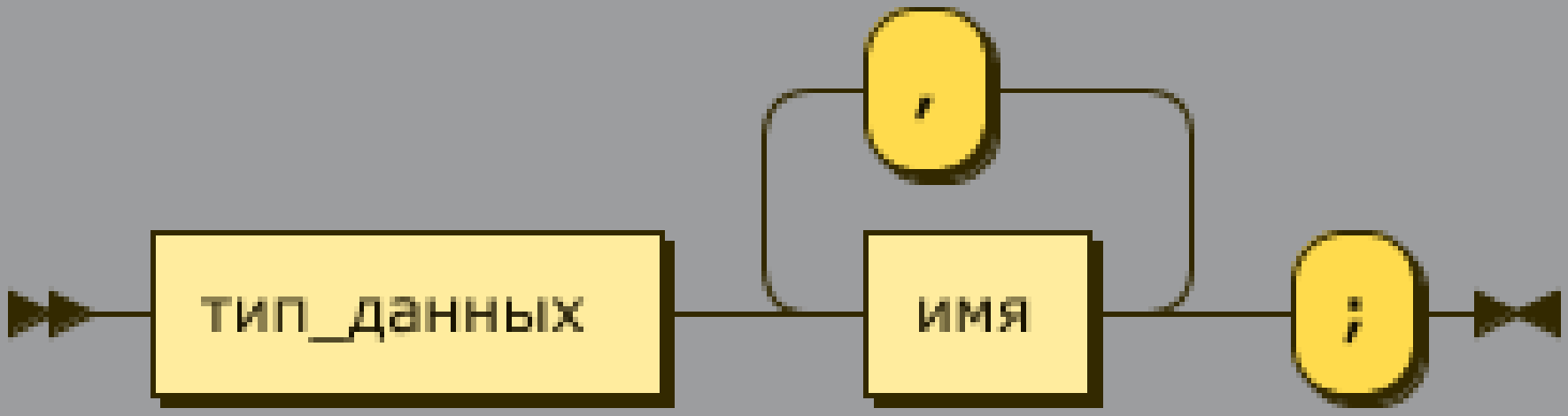
Значение переменной может быть изменено

ЖИЗНЕННЫЙ ЦИКЛ ПЕРЕМЕННОЙ

1. Объявление
2. Присваивания значений
3. Уничтожение

```
объявление_переменной
    ::= тип_данных имя ( ',' имя )* ';'
```

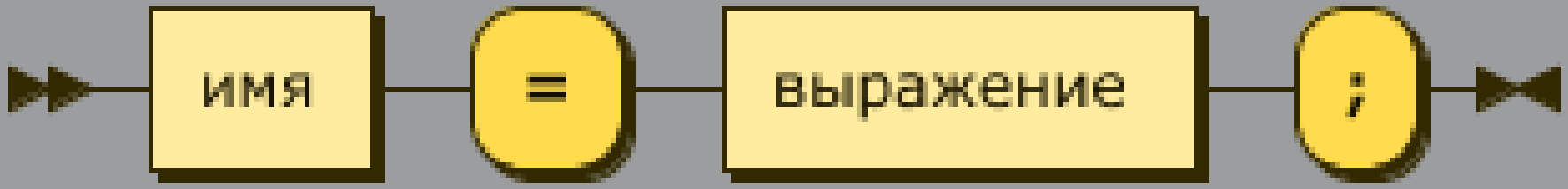
объявление_переменной



```
int i;

string name, name2, someOtherName2;

DateTime startTime;
```

```
присваивание_переменной
    ::= имя '=' выражение ';'
```

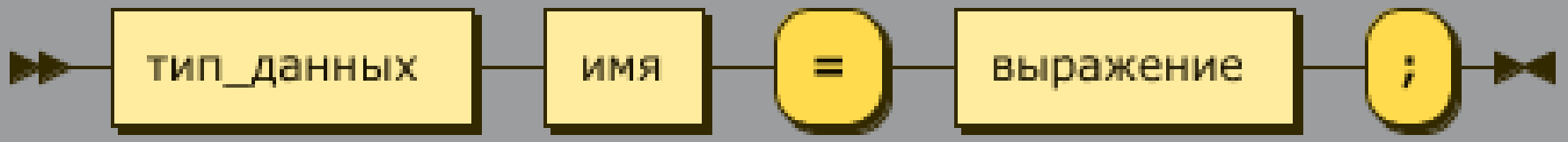
присваивание_переменной



```
i = 7 + j * 42;

name = "Jack " + surname;

startTime = DateTime.Now.AddDays(7);
```

```
объявление_и_инициализация_переменной
    ::= тип_данных имя '=' выражение ';'
```

объявление_и_инициализация_переменной



```
int i = 17;

string name = "Peter";
```

УНИЧТОЖЕНИЕ ПЕРЕМЕННЫХ

Происходит автоматически при выходе из области видимости

Области видимости образуют фигурные скобки и некоторые конструкции: циклы, using.

```
if (age < 18)
{
    string code = Console.ReadLine();

    //  ...
}
```

КОНСТАНТЫ

Хранят значения, используемые в ходе выполнения программы

Значение константы не может быть изменено

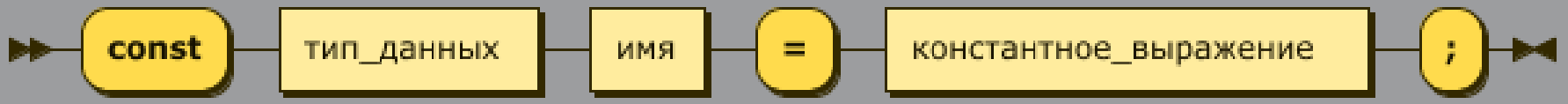
Значение константы должно быть известно на момент компиляции

КОНСТАНТЫ

- Локальные
- Члены типов

```
объявление_и_инициализация_константы ::=
    тип_данных имя '=' константное_выражение ';'
```

объявление_и_инициализация_константы



```
// можно использовать символы Unicode в именах переменных
const double π = 3.1415;
```

ПОЛЯ ТИПОВ

Это данные экземпляра типа

```
class CurrencyConverter
{
    decimal rubInUsd = 72.24M;

    public decimal ConvertRubToUsd(decimal rub)
    {
        return rub * rubInUsd;
    }
}
```

АБСТРАКЦИИ ПО ДЕЙСТВИЯМ

- Методы типов
- Свойства
- Операторы
- Индексаторы

МЕТОДЫ ТИПОВ

Это функции, подобно y = f(x)

Принимают на вход аргументы

Могут возвращать значение

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello world!");
    }
}
```

СВОЙСТВА

Для внешних типов неотличимы от полей

Реализация - методы чтения и записи

```
//  реализация свойства Now обращается к ОС для получения системного време
DateTime now = DateTime.Now;
```

ОПЕРАТОРЫ

Позволяют типа определять реализацию для операторов

```
string fullname = name + " " + surname;
int i = j + k - 7;
```

ИНДЕКСАТОРЫ

Позволяют типа определять реализацию обращения по индексу

```
int[] a = new int[10];

a[4] = 7;   //  обращение по индексу
```

ИТАК, ПОДЫТОЖИМ

C# - ООП язык.

Программа состоит из объектов, описанных типами данных.

Данные объектов хранятся в полях. Так реализуются связи между ними: ассоциация, композиция, агрегация

ИТАК, ПОДЫТОЖИМ

Поведение объектов реализуется в виде методов, операторов, индексаторов и, иногда, свойств

Задачи решаются программой в процессе взаимодействия объектов: вызовы методов, использование операторов, индексаторов, свойств.

ИТАК, ПОДЫТОЖИМ

Входная точка в программу запускает это взаимодействие.

Обычно это метод Main в классе Program.

КОНСТРУКЦИИ ЯЗЫКА

ТЕОРЕМА БЁМА — ЯКОПИНИ

Любой исполняемый алгоритм может быть преобразован к структурированному виду, когда ход его выполнения определяется только при помощи трёх структур управления:

- Последовательной
- Ветвлений
- Циклов

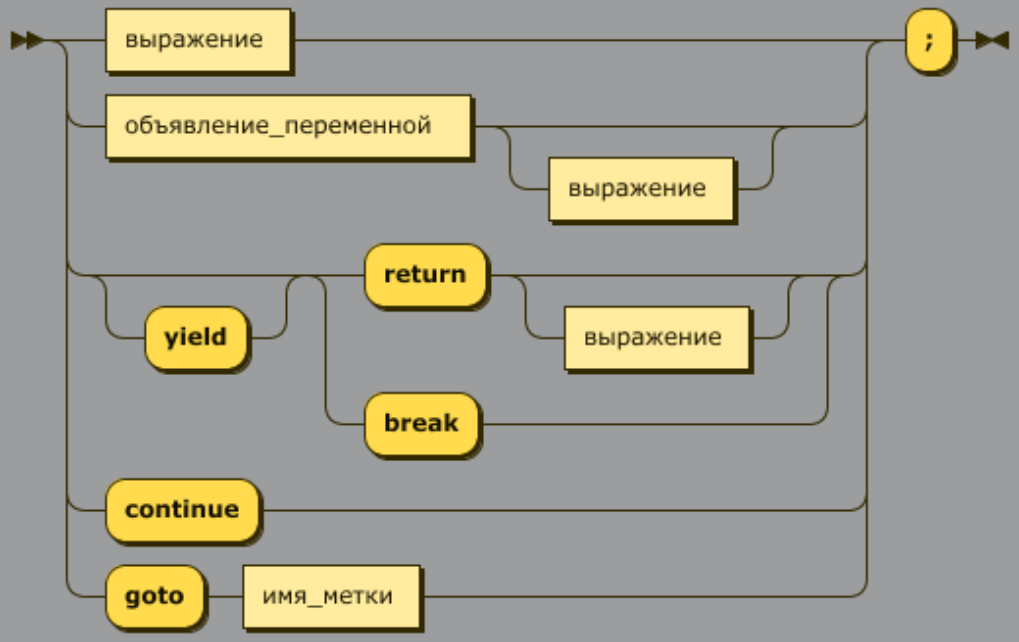
ИНСТРУКЦИЯ

Это строительный блок, из которого складывается программа

- Простые
- Составные
- Ветвления, циклы, checked/unchecked, try-catch, using, lock

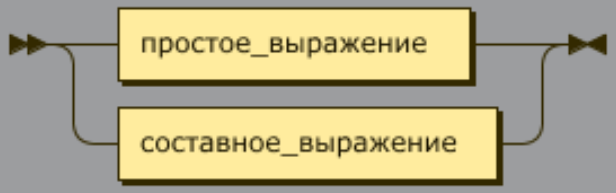
```
простая_инструкция
    ::= (выражение | объявление_переменной выражение? | 'yield'? 'return' 
```

простая_инструкция



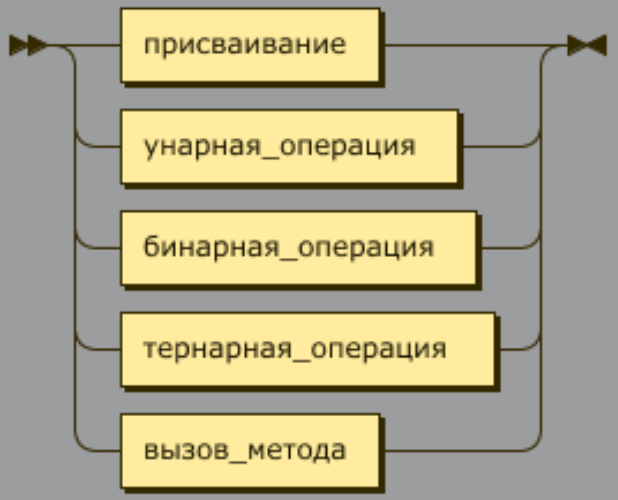
```
выражение
    ::= простое_выражение
      | составное_выражение
```

выражение



```
простое_выражение
    ::=  присваивание
       | унарная_операция
       | бинарная_операция
       | тернарная_операция
       | вызов_метода
```

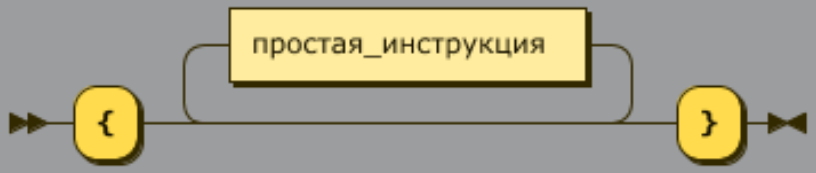
простое_выражение



```
//  присваивание
i = 7

//  унарная операция
!isVisible

// бинарная операция
a + b

// тернарная операция
isAuthorized ? 200 : 403

//  вызов метода
Math.Sin(3.1415)
```

```
составная_инструкция
    ::= '{' простая_инструкция* '}'
```

составная_инструкция



```
{
    int i;

    i = 7;

    Console.WriteLine(i);
}
```

ВЕТВЛЕНИЕ

Позволяет реализовать развилку:

- если-то
- если-то, иначе-это
- если-то, иначе-если-то, ..., иначе-это

ВЕТВЛЕНИЕ

```
if (<логическое_выражение1>)
    <инструкция1>
[ else if (<логическое_выражение2>)
    <инструкция2> ]
[ ... ]
[ else
    <инструкцияN> ]

логическое_выражение - такое, результатом которого
                       является тип bool (System.Boolean)
```

ВЕТВЛЕНИЕ

```
bool condition = 5 > 7;

if (condition)
    Console.WriteLine("Condition is true");
```

ВЕТВЛЕНИЕ

```
bool condition = 5 > 7;

if (condition)
    Console.WriteLine("Condition is true");
else
    Console.WriteLine("Condition is false");
```

ВЕТВЛЕНИЕ

```
bool condition1 = 5 > 7;
bool condition2 = 7 < 10;

if (condition1)
    Console.WriteLine("Condition 1 is true");
else if (condition2)
    Console.WriteLine("Condition 1 is false, condition 2 is true");
else
    Console.WriteLine("Both conditions are false");
```

ВЕТВЛЕНИЕ

```
bool condition = 5 > 7;

if (condition)
{
    Console.WriteLine("Condition is true");
    //  some more actions
}
```

ВЫБОР

```
switch (<выражение>)
{
    [ case <константное_выражение1>:
        [<инструкция>] ]
    [ ... ]
    [ default:
        <инструкция> ]
}
```

инструкция в case опциональна, если после него следует case или default

ВЫБОР

- Удобен для ветвления в зависимости от значения
- Сравнение значений в блоках case возможно только с константами
- Управление может зайти только в 1 case, либо в default, либо ни в один
- А значит, в блоках case и default должно быть прерывание: break, return или throw
- Блок default необязателен, но является хорошей практикой

ВЫБОР

```
int code = 7;

switch (code)
{
    case 3:
        Console.WriteLine("code == 3");
        break;
    case 4:
    case 5:
        Console.WriteLine("code == 4");
        break;
    default:
        Console.WriteLine("Unexpected code: {0}", code);
        break;
}
```

ЦИКЛ С ПРЕДУСЛОВИЕМ

```
while (<логическое_выражение>)
    <инструкция>
```

ЦИКЛ С ПРЕДУСЛОВИЕМ

```
bool condition = // some condition check

while (condition)
{
    //  some actions
    condition = //  some condition check
}
```

ЦИКЛ С ПОСТУСЛОВИЕМ

```
do
    <инструкция>
while (<логическое_выражение>);
```

ЦИКЛ С ПОСТУСЛОВИЕМ

```
bool condition;

do
{
    //  some actions
    condition = //  some condition check
} while (condition);
```

ЦИКЛ FOR

```
for ([<инициализация>]; [<логическое_выражение>]; [<пост_действие>])
    <инструкция>
```

ЦИКЛ FOR

```
for (int i = 0; i < 10; i++)
    Console.WriteLine(i);
```

ЦИКЛ FOR

```
for (int i = 0; i < 10; i++)
{
    Console.WriteLine(i);
}
```

ЦИКЛ FOR - БЕСКОНЕЧНЫЙ

```
for (;;);
```

ЦИКЛ FOR - БЕСКОНЕЧНЫЙ

```
for (;;)
    ;
```

ЦИКЛ FOR - ПРОВЕРКА РАЗ В СЕКУНДУ

```
for (; Check();)
    Thread.Sleep(1000);
```

ЦИКЛ FOREACH

```
foreach (<тип_элемента> <имя_переменной> in <перечеслимое>)
    <инструкция>

где <перечеслимое> реализует интерфейс System.IEnumerable или System.IEnum
```

ЦИКЛ FOREACH

```
string str = "Hello World!";

foreach (char ch in str)
{
    Console.WriteLine(ch);
}
```

ВОПРОСЫ