



# *Методи на Транслация*

## Въведение

*доц. д-р Александър Пенев*

# *Теми*

# Теми разглеждани в курса

1. Въведение. Основни понятия. Транслатори. Класификация. Проект. *(2 часа)*
2. Виртуални изчислителни машини (ВИМ). Език. Мета-език на Бекус-Наур. *(2 часа)*
3. Транслация. Синтактично управляема транслация. Методи и схеми на транслация. *(2 часа)*
4. Свързване. Време на свързване. *(2 часа)*
5. Базова ВИМ. Среда за изпълнение. Управление на паметта и операциите. *(2 часа)*
6. Лексикален анализ. *(2 часа)*
7. Синтактичен анализ. Обработка на грешки. *(2 часа)*
8. Семантичен анализ. Таблица на символите *(2 часа)*
9. Интерпретиране на програма. Генерация на изходна програма. *(2 часа)*
10. Оптимизация. *(2 часа)*



# *Въведение*

# Какво няма да научите в дисциплината?

- ❖ Програмиране на даден ЕП
- ❖ Дискретна математика
- ❖ Лингвистика
- ❖ Всичко за конкретна компютърна архитектура
- ❖ Възможностите на определена среда за програмиране
- ❖ Как е направен най-новия компилатор за C++, Pascal, C#, Java...

# Защо изучаваме Методи на трансляция?

- ❖ Основата на съвременните ЕП
- ❖ Как работят компилаторите
- ❖ Как работят компютрите
- ❖ Какъв машинен код се генерира за различните езикови конструкции в ЕП
- ❖ Как да пишем по-ефективни програми
- ❖ Опит с не-елементарен проект
- ❖ и много други

# Кратка история

В миналото “загадка”, сега една от най-добре познатите области в информатиката

1957	Fortran	първи компилатори (аритм. изрази, изречения, процедури)
1960	Algol	първата явна дефиниция на език (граматики във форма на Бекус-Наур, блокова структура, рекурсии)
1970	Pascal	потребителски дефинирани типове, виртуални машини
1985	C++	обектно-ориентиран, изключения, шаблони (templates)
1995	Java	just-in-time компилация

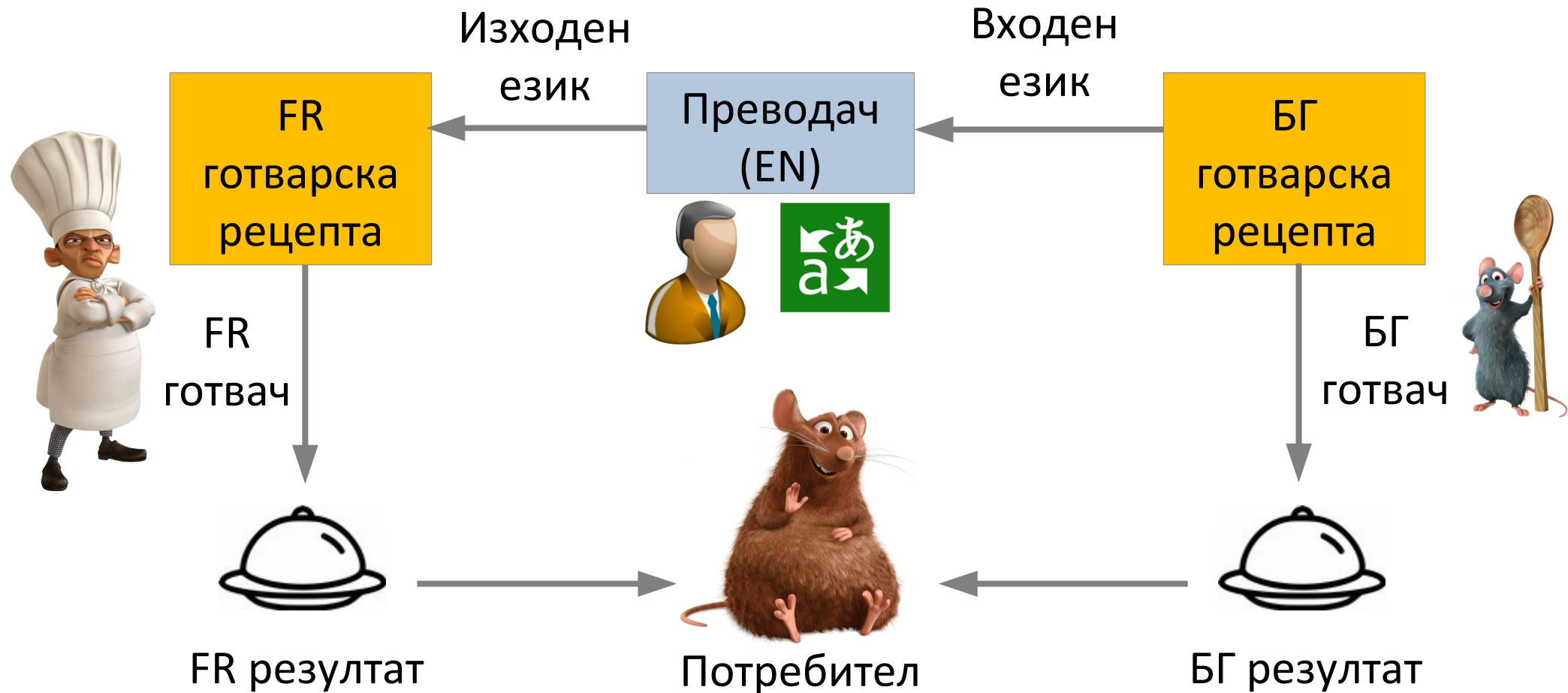
\*Разглеждат се само *императивни езици*



# *Пример*



# Пример



# Пример

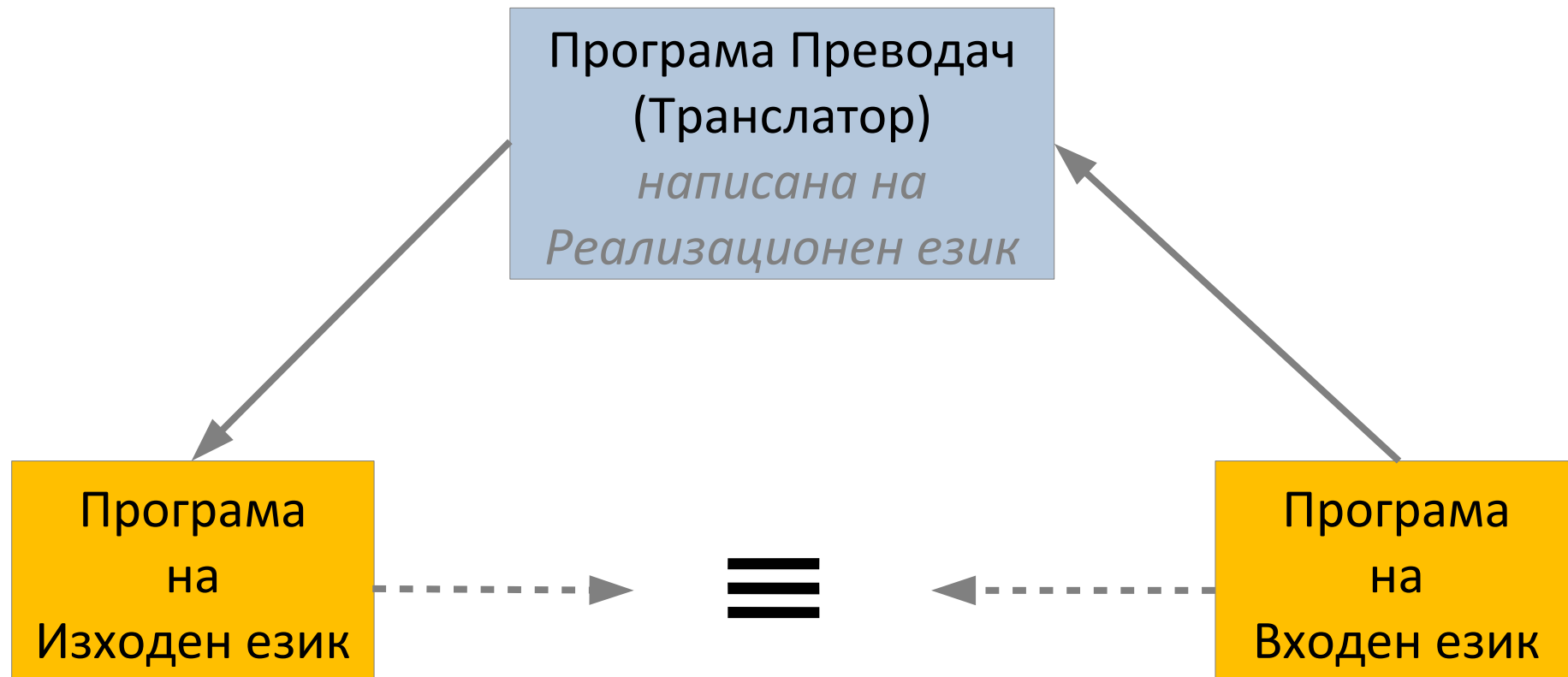
- ❖ **Защо казваме че готвачът французин разбира френски?**  
*Защото разпознава структурата на текста и различните негови под структури и се свързва с действия, които може да извърши при определени условия.*
- ❖ **Как би изпълнил художник-сюрреалист следната проста готварска рецепта: разбийте две яйца; добавете щипка сол; изпечете?**  
*Извод: текстът неявно се класифицира към дадена предметна област*
- ❖ **Трябва ли готвачът да знае какъв ще бъде крайния резултат? (Не)**
- ❖ **Можем ли да наречем готварската рецепта **Програма**? За кого?**  
*Да. Програма за съответния готвач.*
- ❖ **Можем ли да наречен съответната готварска рецепта данни?**  
*Да. За преводача.*
- ❖ **Колко езика трябва да владее преводача?**
- ❖ **Трябва ли преводача да разбира от готварство?**

# Пример

- ❖ Кога казваме, че превода е точен или правилен?
- ❖ Как изпълнява готвачът-французин рецептата?  
*Итеративно, стъпка по стъпка (той интерпретира рецептата).*
- ❖ Възможно ли е двойката <преводач, БГ-готвач> да се съвмести от едно физическо лице?  
*Да. Да работи като псевдо-френски готвач (Интерпретация); Да преведе цялата рецептата на БГ и да я изпълни (Компиляция).*
- ❖ Може ли една рецепта да се изпълни от колектив от готвачи?
- ❖ Може ли преводачът да владее повече от два езика?
- ❖ От гледна точка на преводача кой език е входен, кой изходен и кой собствен?  
*Вътрешния език е собствен, входния е началния език, изходния е крайния.*

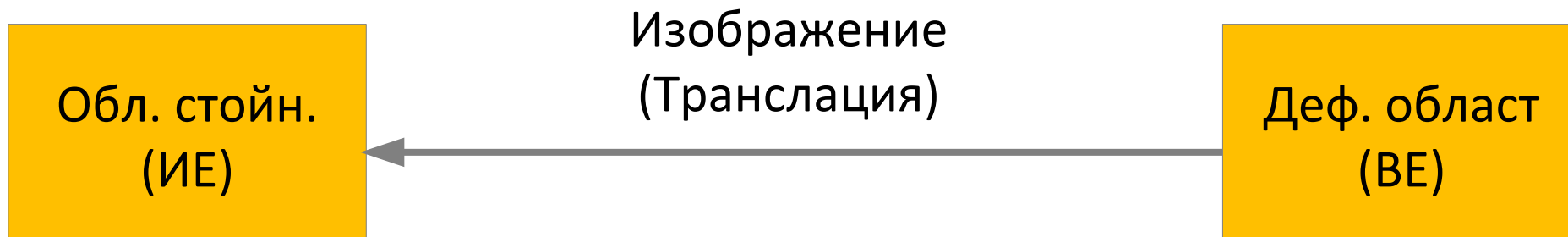
# *Формална Постановка*

# Формална постановка



# Транслатор

Реализацията на изображението от дефиниционната област на ВЕ в областта от стойности на ИЕ, която е реализирана в термините на някакъв реализационен език (РЕ), се нарича **Транслатор**.



# Методи на транслация

- ❖ Метод на транслация наричаме:
  - ❖ Отговор на въпроса как да извършим превода;
  - ❖ Алгоритъм, чиято реализация извършва превода;

За да се овладее методите на транслация е необходимо да се знаят отговорите на следните въпроси:

- ❖ Що е компютър (изчислителна машина)?
- ❖ Как функционира компютъра?
- ❖ Що е език (Език за програмиране – ЕП)?
- ❖ Какви видове ЕП има?
- ❖ Какви са структурата и съставът на всеки ЕП?
- ❖ Как се описва един ЕП?

# *Работа на Компилятора*



# Динамична структура на компилатор

Поток от символи

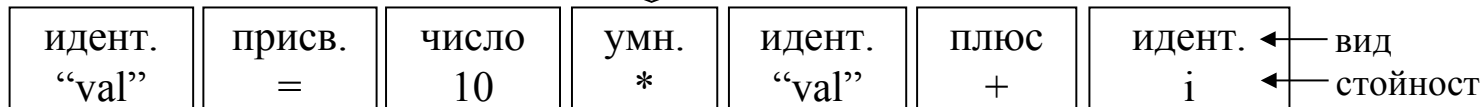
v	a	l	=	1	0	*	v	a	l	+	i
---	---	---	---	---	---	---	---	---	---	---	---



Лексикален анализ (скарниране)

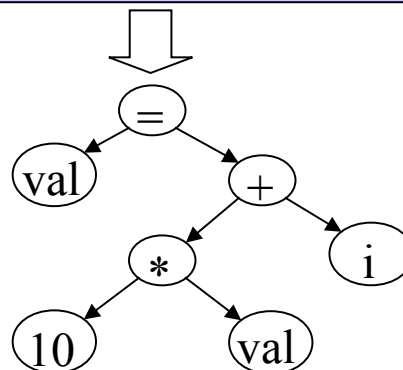


Поток от лексеми



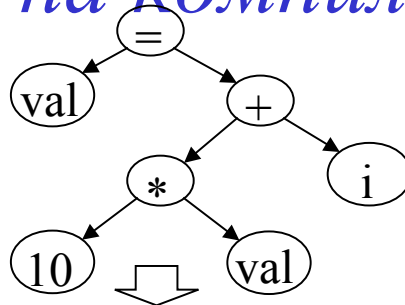
Синтактичен анализ (разпознаване)

Синтактично дърво



# Динамична структура на компилатор

Синтактично дърво



Семантичен анализ

Междинно  
представяне

Синтактично дърво, символна таблица, ...

Оптимизация

Машинно независима

Генерация на код

Машинен код  
(или IR)

```
ld.i4.s 10
ldloc.1
mul
```

Оптимизация

Машинно зависима

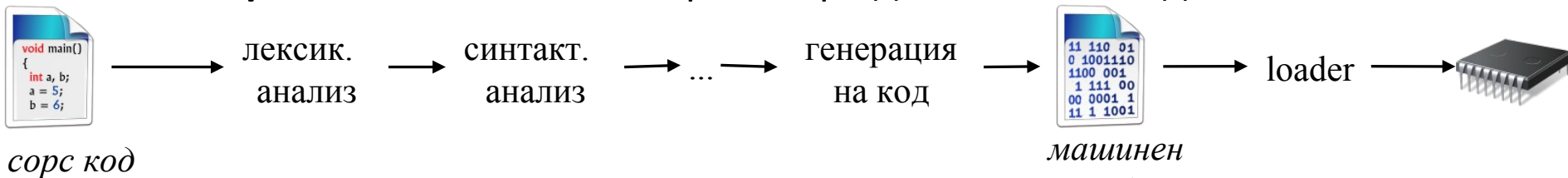


# *Видове Транслатори*

# Компилятор или интерпретатор

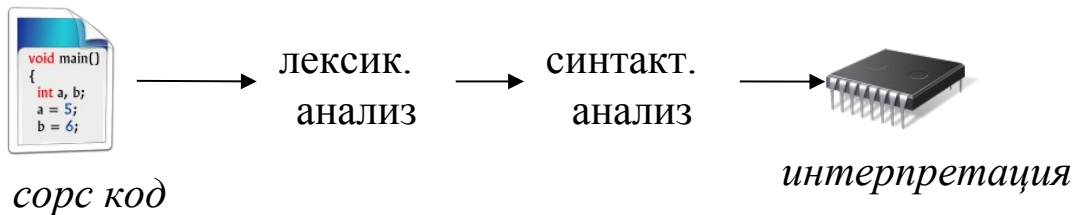
## Компилятор

транслира до машинен код



## Интерпретатор

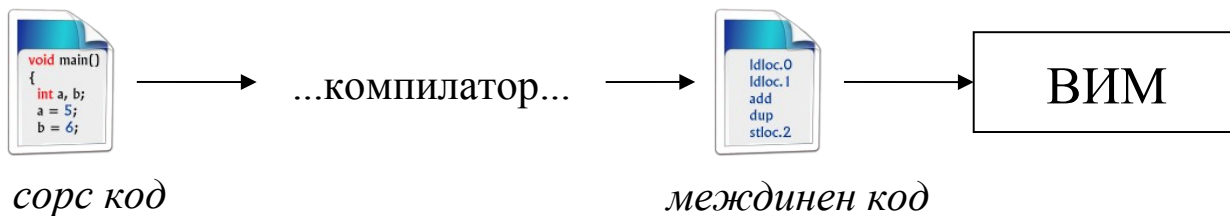
изпълнява сорс кода "директно"



❖ Операторите в цикъл се анализират отново и отново

## Хибриден компилатор

интерпретира междинен код



❖ Сорс кода се транслира до код за виртуално-изчислителна машина (ВИМ)

❖ ВИМ интерпретира кода, симулирайки реална машина

(напр. Common Intermediate Language (CIL))

# Видове Транслатори

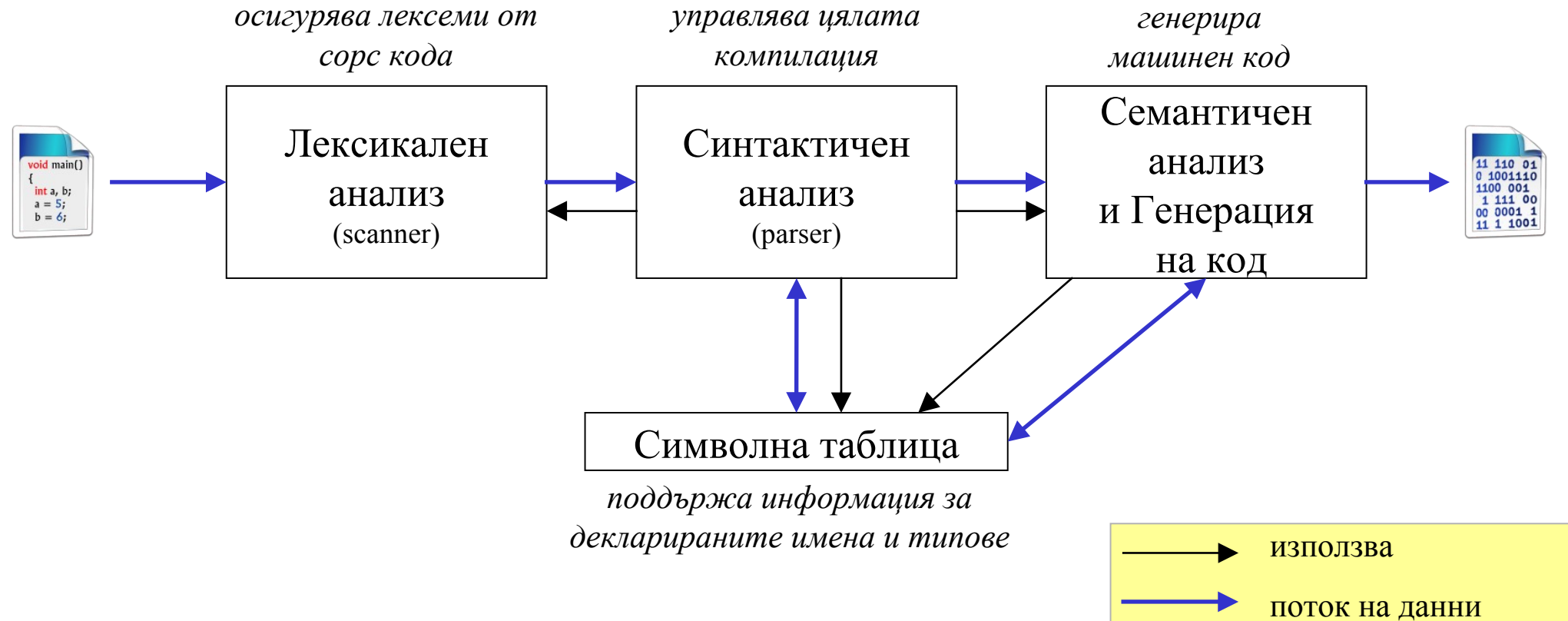
- ❖ Транслатор – ако преводачът не изпълнява, то превода се нарича транслация, а преводачът транслатор:
  - ❖ Конвертор – ВЕ и ИЕ са езици от високо ниво;
  - ❖ Компилатор – ВЕ е език от високо ниво, а ИЕ от ниско ниво;
  - ❖ Макро-асемблер – ВЕ е макро-език, а ИЕ е модификация на машинния език;
  - ❖ Асемблер – ВЕ е мнемокод, а ИЕ е машинен език;
  - ❖ Съвързващ редактор (Linker) – ВЕ съвпада с ИЕ на асемблера, а ИЕ е макро-асемблер в преместваем формат;
  - ❖ Зареждач (Loader) – ВЕ съвпада с ИЕ на линкера, ИЕ му е машинен език;
  - ❖ Изпълнител – самата ВИМ;

# Видове Транслатори

- ❖ Интерпретатор – Програма написана на машинен език, която изпълнява написана на ВЕ се нарича интерпретатор, а работата и се нарича интерпретация на ВЕ или входната програма;
- ❖ Хибриден компилатор – Сорс кода се транслира до код за виртуално-изчислителна машина (ВИМ), която интерпретира (компилира при нужда части от) кода, симулирайки реална машина;
- ❖ Декомпилатор – ВЕ е с по-ниско ниво, отколкото ИЕ;

# *Класическа Структура на Компилятор*

# Статична структура на компилатор





# *Някои Основни Понятия*

# Какво е граматика?

**Пример:**

```
Statement = 'if' '(' Condition ')' Statement ['else' Statement].
```

**Има четири компонента**

**терминални  
символи (ТС)**

атомарни са

"if", "<=", Ident, Number, ...

**нетерминални  
символи (НТС)**

сводими до по-  
малки единици

Statement, Expr, Type, ...

**правила за  
извод**

правила как да  
се декомпозират  
нетерминали

Statement = Designator "=" Expr ";"  
Designator = Ident ["." Ident]  
...

**стартов символ**

първия  
нетерминал

Program



# EBNF Номинация

Разширена Форма на Бекус-Наур

John Backus: 1-ви компилатор за Fortran

Peter Naur: Algol60

Символ	Значение	Примери
низ	описва себе си	'=', 'while'
име	описва ТС НТС	Ident, Statement
=	отделя страните на произведението	A = b c d .
.	прекъсва произведение	
	отделя алтернативи	a   b   c $\equiv$ a или b или c
(...)	групира алтернативи	a ( b   c ) $\equiv$ ab   ac
[...]	опционална част	[ a ] b $\equiv$ ab   b
{...}	повтаряща се част	{ a } b $\equiv$ b   ab   aab   aaab

## Конвенции:

- ❖ Терминалните символи са заградени в ‘ ’ (напр. ‘while’)
- ❖ Нетерминалните символи се задават с имена (напр. Statement)



# Пример: Граматика за аритметични изрази

## Правила/Произведения (Productions)

$\text{Expr} = [ '+' \mid '-' ] \text{Term} \{ ( '+' \mid '-' ) \text{Term} \}.$

$\text{Term} = \text{Factor} \{ ( '*' \mid '/' ) \text{Factor} \}.$

$\text{Factor} = \text{Ident} \mid \text{Number} \mid '( \text{Expr} )'.$

## Терминални символи

### Прости ТС

'+', '-', '\*', '/', '(', ')'

(само една инстанция)

### Терминални класове

Ident, Number

(множество инстанции)

## Нетерминални символи

Expr, Term, Factor

## Стартов символ

Expr



# Терминология

## Азбука (Alphabet)

Множеството от терминални и нетерминални символи на граматиката

## Дума (String)

Безкрайна последователност от символи на азбуката

Думите се отбелязват с букви от гръцката азбука ( $\alpha$ ,  $\beta$ ,  $\gamma$ , ...)

например:

$\alpha = \text{Ident} + \text{Number}$

$\beta = - \text{AdditiveExpr} + \text{MultiplicativeExpr} + \text{Number}$

## Празна дума (Empty String)

Дума, която не съдържа нито един символ (отбелязва се с  $\epsilon$ ).



# Изводи и редукции

## Извод (Derivation)

$\alpha \Rightarrow \beta$  (директен извод)

$\alpha \Rightarrow^* \beta$  (индиректен извод)  $\alpha \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \dots \Rightarrow \gamma_n \Rightarrow \beta$

$\alpha \Rightarrow^L \beta$  (ляв каноничен извод) най-левият НТС в  $\alpha$  се извежда

$\alpha \Rightarrow^R \beta$  (десен каноничен извод) най-десния НТС в  $\alpha$  се извежда

## Редукция (Reduction)

Противоположно на извода:

Ако дясната страна на правилото се срещне в  $\beta$ , се замества със съответния НТС.

$$\begin{array}{c} \overbrace{\text{AdditiveExpr} + \text{Literal} * \text{Literal}}^{\alpha} \Rightarrow \\ \text{НТС} \\ \overbrace{\text{AdditiveExpr} + \text{Ident} * \text{Literal}}^{\beta} \\ \text{дясната страна на} \\ \text{правилото на НТС} \end{array}$$



# Още Терминология

## Фраза (Phrase)

Всяка дума, която може да се изведе от нетерминален символ.

пр.: MultiplicativeExpr фрази: `Literal`; `Literal * Literal`; `Number * Literal`; ...

## Междинна форма на разбор (Sentential Form)

Всяка дума, която може да бъде изведена от стартовия символ на граматиката.

пр.: `Expr`; `Literal + Literal * Number`; ...

## Изречение (Sentence)

Междинна форма на разбор, която има само терминални символи.

пр.: `Number * Ident * Number`

## Език (формален език) (formal language)

Множеството от всички изречения на граматиката (обикновено безкрайно)

пр.: Езикът C# е множеството от всички валидни C# програми



# Рекурсия

Произведението е рекурсивно ако:  $A \Rightarrow^* \omega_1 A \omega_2$

Може да бъде използвано за представяне на повторения и вложени структури

Пряка рекурсия  $A \Rightarrow^* \omega_1 A \omega_2$

Лява рекурсия  $A = b \mid A a.$   $A \Rightarrow Aa \Rightarrow Aaa \Rightarrow baaaa...$

Дясна рекурсия  $A = b \mid a A.$   $A \Rightarrow aA \Rightarrow aaA \Rightarrow ...aaaab$

Централна рекурсия  $A = b \mid a (' A ')$ .  $A \Rightarrow (A) \Rightarrow ((A)) \Rightarrow (((... (b) ...)))$

Косвена рекурсия  $A \Rightarrow^* \omega_1 A \omega_2$

пример:

Expr = Term { '+' Term }.  
Term = Factor { '\*' Factor }.  
Factor = id | '(' Expr ')

Expr  $\Rightarrow$  Term  $\Rightarrow$  Factor  $\Rightarrow$  '(' Expr ')





# Как се премахва лявата рекурсия

**Лявата рекурсия е проблем при анализа отгоре надолу:**

Може да бъде използвано за представяне на повторения и вложени структури

$A = b \mid A a.$       Двете алтернативи започват с  $b$ .

Синтактичният анализатор не може да определи коя да избере

**Лявата рекурсия може да бъде трансформирана до итерация**

$E = T \mid E '+' T.$

Изречения, които могат да се получат

$T$

$T + T$

$T + T + T$

...

Може да се запише чрез итеративното правило на EBNF

$E = T \{ '+' T \}.$

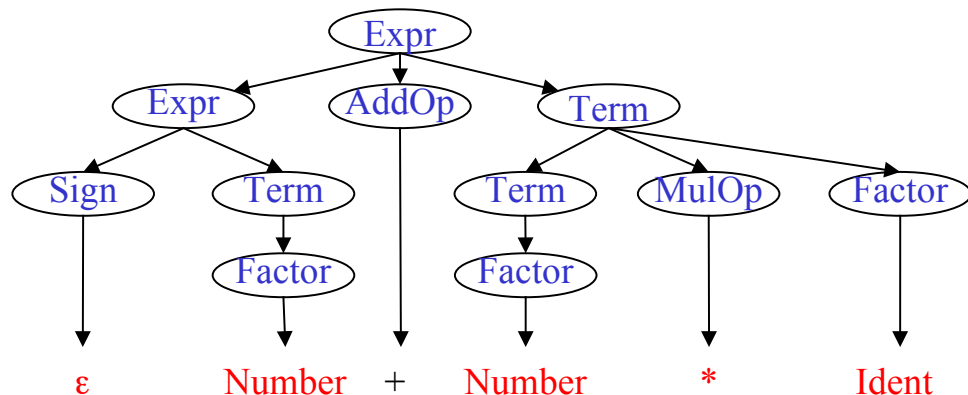


# Синтактично дърво и нееднозначност

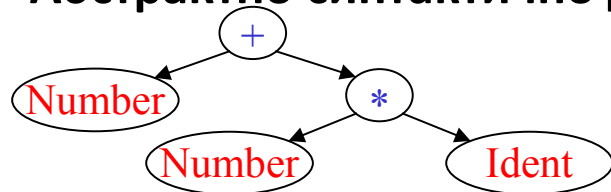
Показва структурата на определено изречение

пример за:  $10 + 3 * i$

Конкретно синтактично дърво (дърво на разбора)



Абстрактно синтактично дърво



Също влияе върху приоритета на операциите:

Операторите, които са по-навътре в дървото имат по-висок приоритет.

Листата са операнди, а върховете са операторите.

Често се използва като вътрешно представяне на програмата. То се използва за прилагане на оптимизации

# Нееднозначност

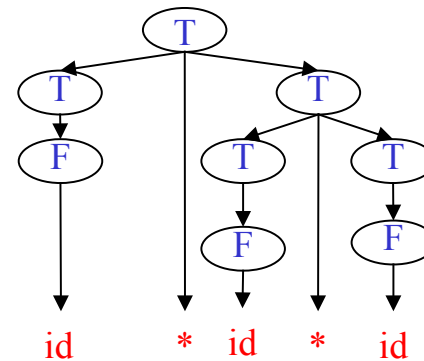
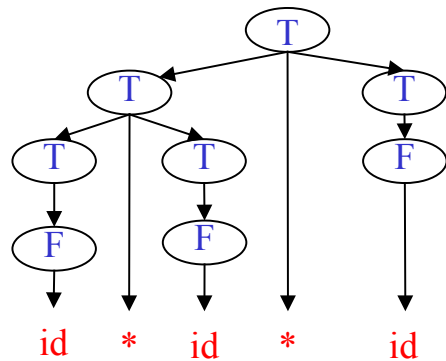
Една граматика е нееднозначна, когато може да се създаде повече от едно синтактично дърво за дадено изречение

Например:

$T = F \mid T '*' T$ . Изречението е:  $id * id * id$

$F = id$ .

Могат да се създадат две синтактични дървета за това изречение:



*Нееднозначните граматики довеждат до проблеми по време на синтактичния анализ!*

# Премахване на нееднозначността

Пример:

$T = F \mid T '*' T.$

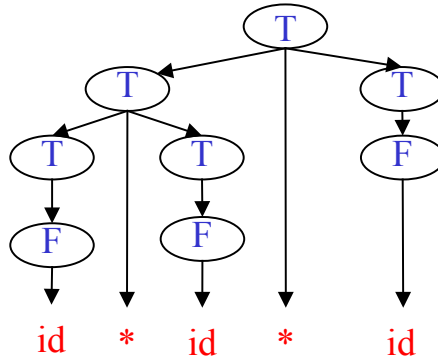
$F = \text{id}.$

Само граматиката е нееднозначна, а не езика.

Грамматиката може да бъде преобразувана до:

$T = F \mid T '*' F.$

$F = \text{id}.$



*T има приоритет пред F.*

*Само това синтактично дърво е възможно да се създаде.*

Дори по-добро преобразуване в EBNF е възможно:

$T = F \{ '*' F \}.$

$F = \text{id}.$



# Премахване на нееднозначността

**Има езици, които са нееднозначни**

Пример: Висящ Else (*Dangling Else*)

Няма граматика, която да е еднозначна


**Решение:**

Да се избере едно от синтактичните дървета на базата на допълнително съглашение.

*Решение в C#*

*Винаги се разпознава (избира) правилото с най-дълга дясна страна. Предимството на този подход е, че довежда до създаване на най-ниски (малки) синтактични дървета.*

```
if (a<b)
  if (b<c)
    x=c
? else
  x=b
```



# *Проект*

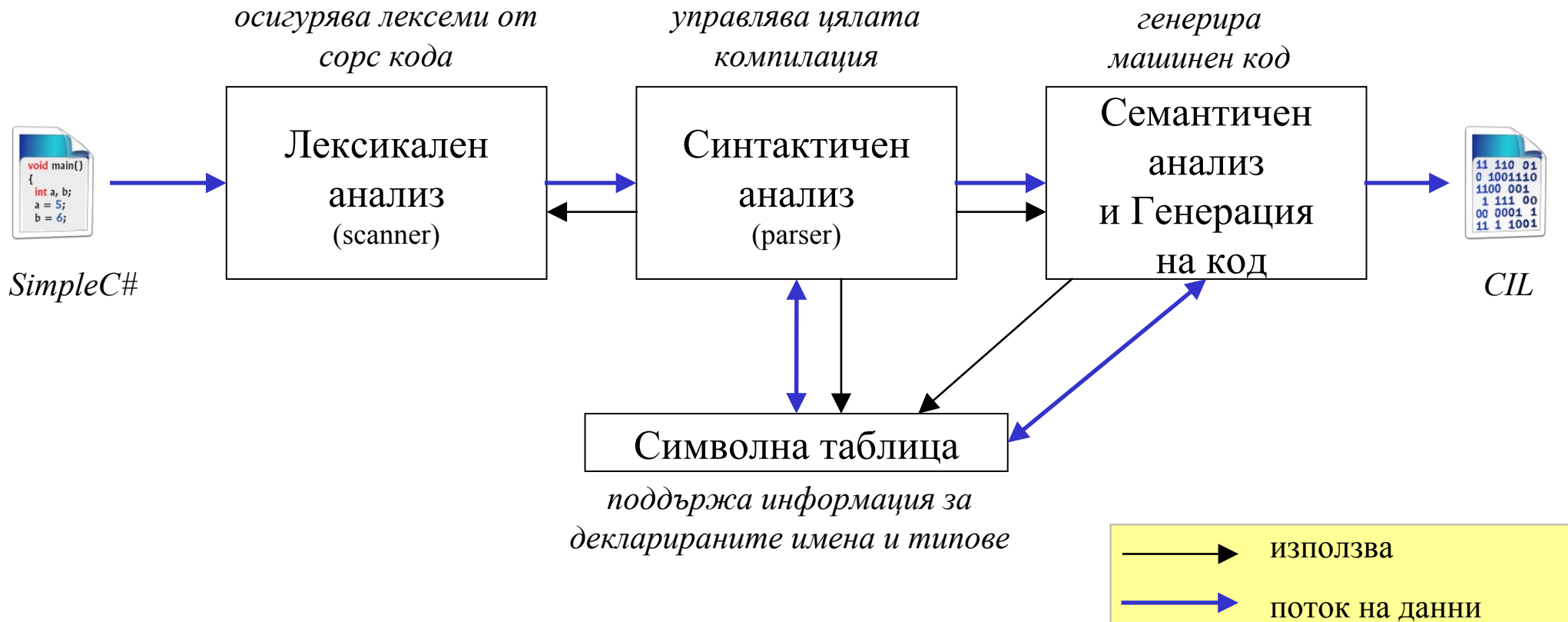
# Проект

Всеки студент трябва да разработи **курсов проект** по Методи на трансляция, който представлява **Компилятор**, превеждащ от език на високо ниво на език от ниско ниво (машинен или асемблер).

*Помощните материали към курса съдържат примерен проект компилатор, както и синтаксисите на възможните входни езици на разработвания компилатор.*



# Действие на разглеждания компилатор





# Примерна програма на SimpleC#

```
using System;
class Program {
    int HW[10];
    int Area(int a, int b) {
        return a*b;
    }
    void Main() {
        int i;
        i = 9;
        while (i>=0) {
            HW[i] = Area(i, 2 * i);
            if (HW[i] == 0)
                System.Console.WriteLine("Zero");
            else
                System.Console.WriteLine(HW[i]);
            i = i - 1;
        }
    }
}
```

- ❖ Включване на модули
- ❖ Глобални променливи
- ❖ Локални променливи
- ❖ Масиви
- ❖ Методи



# Лексикална структура на SimpleC#

<b>Ident</b>	Letter {Letter   Digit}.	Abc1
<b>Number</b>	Digit {Digit}.	15
<b>Float</b>	Digit {Digit} '.' {Digit}.	1.5
<b>Boolean</b>	'false'   'true'.	True
<b>Char</b>	"" AnyChar "".	'z'
<b>String</b>	"" {{AnyChar\(\Escape ''''\)} [Escape('0' 't' 'n' 'f' 'r' '''' '''' Escape)]} ""	"string\n"
<b>Delimiter</b>	Space   '/*' {AnyChar}\('*' '/') '*/'   '//' {AnyChar\(\n r)} 'int'   'boolean'   'double'   'char'   'string'   'void'   'null'	/* коментар */
<b>Keyword</b>	'using'   'if'   'else'   'while'   'return'   'break'   'continue'	
<b>Special Symbol</b>	';'   '{'   '}'   ','   '='   '*'   '('   ')'   '['   ']'   '='   ' '   '&&'   ' '   '&'   '=='   '!='   '<'   '>'   '<='   '>='   '+'   '-'   '/'   '%'   '++'   '--'   '~'   '!'   '!'	



# Синтактична структура на SimpleC#

## Програми

Program = {UsingClause} 'class' Ident '{' {FieldDecl | MethodDecl} '}'.

```
using ...;  
class Program {  
    ...декларации...  
    ...методи...  
}
```

## Декларации

UsingClause = 'using' Ident ';'.

FieldDecl = Type (Ident | Ident '[' Number ']' ) ';'.

MethodDecl = (Type | 'void') Ident '(' {(Type Ident)\','}' Block.

Block = '{' {VarDecl} {Statement} '}'.

VarDecl = Type Ident ';'.

Type = 'int' | 'boolean' | 'float' | 'char' | 'string' | Ident .

*Разпознават се само едномерни масиви, и то те не могат да бъдат локални променливи*



# Синтактична структура на SimpleC#

## Изречения

Statement = Location '=' Expr ';' | MethodCall ';' | 'if' '(' Expr ')' Statement ['else' Statement] |  
'while' '(' Expr ')' Statement | 'return' Expr ';' | 'break' ';' | 'continue' ';' | Block.  
MethodCall = Ident '(' [Expr {',' Expr}] ')'.  
Location = Ident | Ident '[' Expr ']'.

## Изрази

Expr = AdditiveExpr [( '<' | '<=' | '==' | '!=' | '>=' | '>' ) AdditiveExpr].  
AdditiveExpr = ['+' | '-'] MultiplicativeExpr { ('+' | '-' | '|' | '|') MultiplicativeExpr }.  
MultiplicativeExpr = SimpleExpr { ('\*' | '/' | '%' | '&' | '&&') SimpleExpr }.  
SimpleExpr = Location | MethodCall | Literal | '++' Location | '--' Location | Location '++' |  
Location '--' | '~' Expr | '-' Expr | '!' Expr | '(' Expr ')'.  
Literal = Number | Boolean | Float | Char | String | 'null'.



*Въпроси?*  
*apenev@uni-plovdiv.bg*

