

# Обработка на текст. Парсер

проф. д-р инж. Христо Вълчанов

<http://cs.tu-varna.bg>

# Въведение

- Отделянето на компонентите на текст е известно като *parsing*.
- Процес на преобразуване на поток от лексеми (думи, символи) в абстрактно синтактично дърво.

# Граматика на език

- Формално детайлно дефиниране на това каква последователност от символи формира синтактично коректен език.
- Граматиката **G** на даден език обикновено се задава като четворка **G=(N,  $\Sigma$ , P, S)**, където:
  - **$\Sigma$**  е множеството терминални символи на езика;
  - **N** е множеството нетерминални (помощни) символи на езика, непресичащо се със  **$\Sigma$** ;
  - **S** е стартов символ на граматиката;
  - **P** включва правила (продукции) от вида  **$\alpha \rightarrow \beta$** . Лявата и дясната част на граматично правило представляват низове, които могат да съдържат както терминални, така и нетерминални символи. Празен извод (не се генерира продукция) се обозначава чрез символа  **$\epsilon$** .

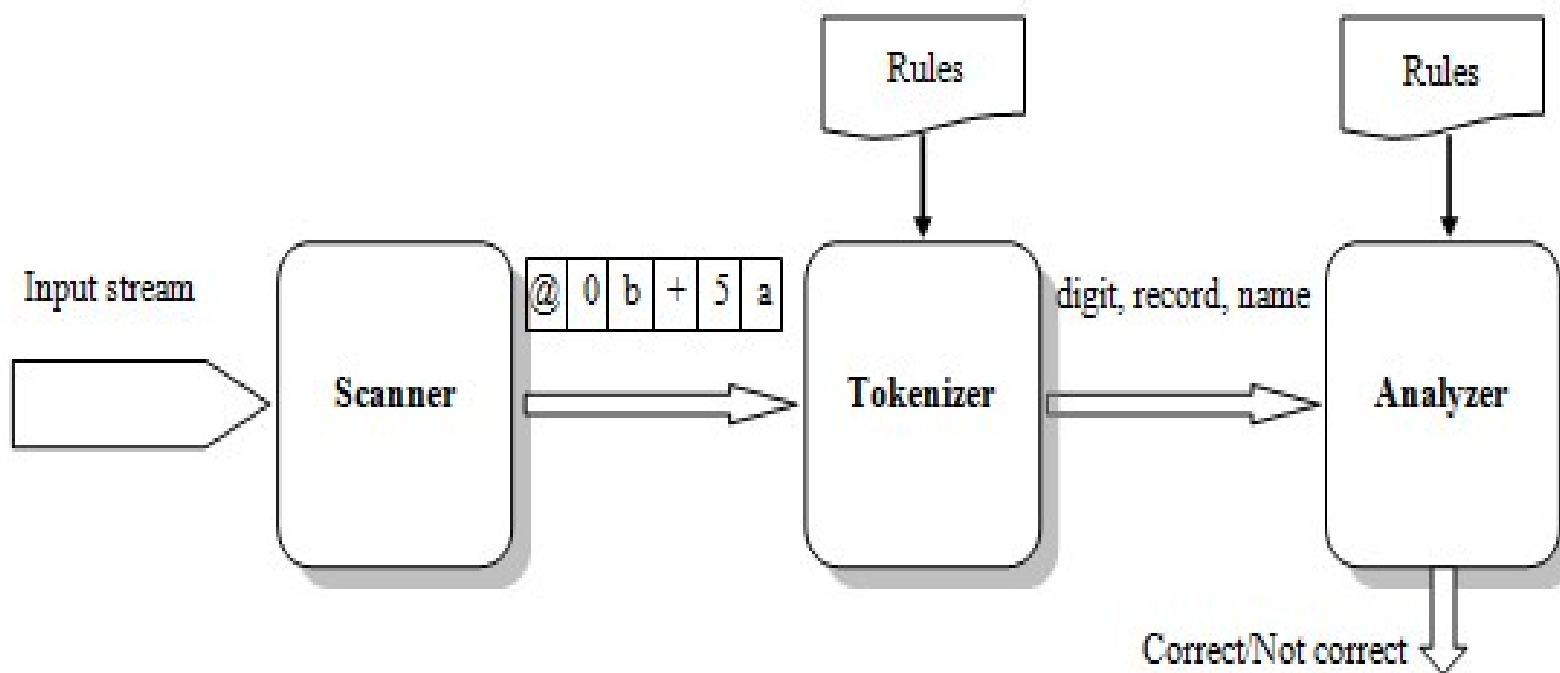
# Дефиниране на граматика

- Бакус-Наурова форма (БНФ).

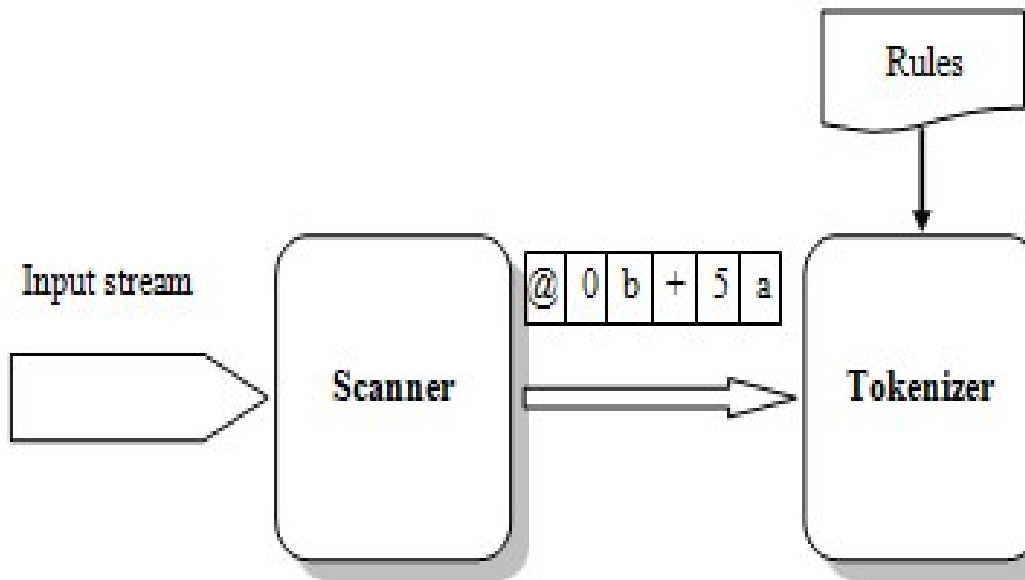
***присвояване ::= променлива “=” израз***

Правилото съдържа нетерминалите *присвояване*, *променлива* и *израз*, и терминалният символ „=“.

# Структура на парсер



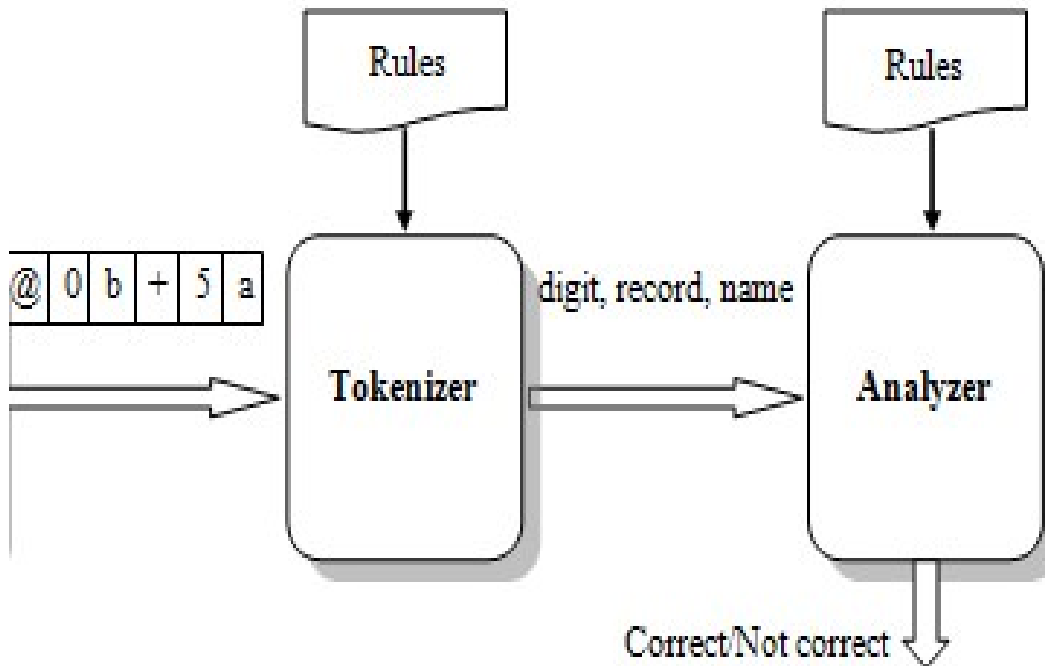
# Модул Scanner



Подготвя входния  
поток за обработка.

Интерфейс:  
**`GetNextChar()`**  
**`Char`**

# Модул Tokenizer



Разпознаване на лексемите (tokens) от входния поток.

Интерфейс:  
***getNextSymbol()***  
***Symbol***

# Типове лексеми

- Лексемите на езика се формират от азбуката му по определени правила (БНФ).

*identifier* ::= *letter* { *letter* | *digit* }

*letter* ::= *A* | *B* | *C* | *E* | *G* | *H* | *J* | *K* | *L* | *M* |  
*N* | *P* | *R* | *S* | *T* | *V* | *W* | *X* | *Y* | *Z*

*digit* ::= *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9*



# Дефиниция на лексеми

```
typedef enum TSymbolType {  
    ident, intconst, plusop, ...  
} TSymbol;
```

```
TSymbol Symbol;
```

# Пример за лексеми

Входна последователност	Лексема	Стойност
olleH	ident	Spelling="Hello"
4321	intconst	Constant=1234
+	plusop	Spelling="" Constant=""

```
char Spelling [MAXLENGTH];  
int Constant;
```

# Формиране на лексеми

```
void GetNextSymbol() {  
    Пропускане_на_незначещи_символи;  
    switch ( Char ) {  
        case 'a': ... case 'z':  
            Анализ_на_идентификатори;  
        case '0': ... case '9':  
            Анализ_на_целочислени_константи;  
        case ':', case '.', case '/', case '<', case '>':  
            Анализ_на_оператори_и_разделители;  
        default: {  
            Symbol = othersy;  
            GetNextChar();  
        }  
    }  
}
```

Променливата *Symbol* приема стойност, съответстваща на типа лексема

# Пример – разпознаване на идентификатор

```
void GetNextSymbol() {
    int digit=0;
    int k=0;
    while ( Char == ' ' ) { GetNextChar(); } // Прескачане празни позиции
    switch (toupper(Char)) {
        // Идентификатор
        case 'A'-'Z': {
            strcpy(Spelling, "          ");
            do {
                if ( k < MAXLENGTH ) {
                    Spelling[k] = toupper(Char);
                    k++;
                }
                GetNextChar();
            } while ((Char >= '0') && (Char <= '9') ||
                    (toupper(Char) >= 'A') && (toupper(Char) <= 'Z'));
            Symbol = ident;
        } break;
    }
```

*identifier ::= letter { letter | digit }*  
*letter ::= A | B | C | E | G | H | J | K | L | M |*  
*N | P | R | S | T | V | W | X | Y | Z*  
*digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9*

# Пример – оператор от два символа

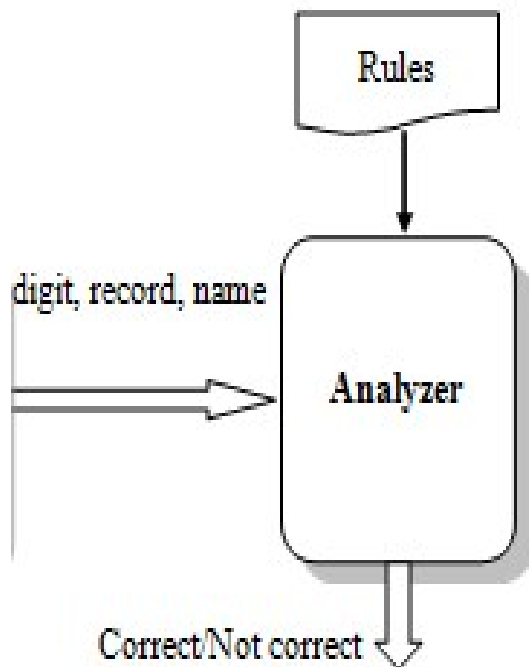
```
case '<': {  
    GetNextChar();  
    if ( Char == '/' ) {  
        Symbol = endTag;  
        GetNextChar();  
    } else {  
        Symbol = startTag;  
        ...  
    }  
} break;
```

*starttag ::= < letter >*

*endtag ::= < / letter >*

*letter ::= A | B | C | E | G | H | J | K | L | M |  
N | P | R | S | T | V | W | X | Y | Z*

# Модул Analyzer



Проверява се дали дадена последователност от лексеми е валидна съгласно синтактичните правилата на дадена граматика.

# Анализатор

- Използва се метод на рекурсивното спускане (LL(1) анализ).
- На нетерминални символи се съпоставят съответните реализиращи ги синтактични функции.
- Обработката на терминални символи в анализатора се състои в сравняването им с очакваните терминални символи според правилата на граматиката.

# Анализ на входния поток

- Прилагат се две базови функции:

```
int accept(TSymbol sym) {
    if (Symbol == sym) {
        GetNextSymbol();
        return 1;
    }
    return 0;
}

int expect(TSymbol sym) {
    if (accept(sym)) return 1;
    error("expect: unexpected symbol");
    return 0;
}
```



# Реализация на правила

Терминални символи – малки букви.

Нетерминални символи – главни букви

Текуща лексема – в променлива *Symbol*

$$A ::= a b B$$

```
void A() {  
    accept( a );  
    accept( b );  
    B();  
}
```

# Реализация на правила

Терминални символи – малки букви.

Нетерминални символи – главни букви

Текуща лексема – в променлива *Symbol*

$$A ::= a \{ b B \}$$

```
void A() {  
    accept( a );  
    while ( Symbol == b ) {  
        accept( b );  
        B();  
    }  
}
```

# Реализация на правила

Терминални символи – малки букви.

Нетерминални символи – главни букви

Текуща лексема – в променлива *Symbol*

$$A ::= a \{ b \ c \ B \ d \ E \}$$

```
void A() {  
    accept( a );  
    while ( Symbol == b ) {  
        accept( b );  
        accept( c );  
        B();  
        accept( d );  
        E();  
    }  
}
```

# Реализация на правила

Терминални символи – малки букви.

Нетерминални символи – главни букви

Текуща лексема – в променлива *Symbol*

$$A ::= b B \mid d D$$

```
void A() {  
    switch ( Symbol ) {  
        case b: {  
            accept( b );  
            B();  
        } break;  
        case d: {  
            accept( d );  
            D();  
        } break;  
    }  
}
```

---

# Реализация на правила

Терминални символи – малки букви.

Нетерминални символи – главни букви

Текуща лексема – в променлива *Symbol*

$$A ::= b B \mid empty$$

```
void A() {  
    if ( Symbol == b ) {  
        accept( b );  
        B();  
    }  
}
```

# Пример за анализ

- Азбуката на езика включва:
  - Главни букви ('A' .. 'Z');
  - Цифри ('0' .. '9');
  - Специални символи ('.', ';', '"').
- 
- Лексемите на езика са:
  - Целочислени константи с максимална стойност 1000000;
  - Низове с максимална дължина 8 символа;
  - Едно-символни оператори.

# БНФ форма

*datafile* ::= *record* { *record* }

*record* ::= *field* { ; *field* } .

*field* ::= *integer* | *string*

*integer* ::= *digit* { *digit* }

*string* ::= " *text* "

*text* ::= *letter* { *letter* | *digit* }

*letter* ::= A | B | C | E | G | H | J | K | L | M |

N | P | R | S | T | V | W | X | Y | Z

*digit* ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

# Данни

*1;"Иван";62100;"СИТ";5.*

*2;"Георги";62217;"СИТ";2.*



# Пример за анализ

Терминални символи: “.”, “.” и “.”

Дефиниция на лексеми:

```
typedef enum TSymbolType {  
    intconst, text,  
    semicolon, period, quotas, othersy  
} TSymbol;
```

# Пример за анализ

```
void GetNextSymbol() {
    int digit=0;
    int k=0;

    // Прескачане празни позиции
    while ( Char == ' ' ) { GetNextChar(); }

    switch (toupper(Char)) {
        // Текст
        case 'A'-'Z': {
            strcpy(Spelling, "          ");
            do {
                if ( k < MAXLENGTH ) {
                    Spelling[k] = toupper(Char);
                    k++;
                }
                GetNextChar();
            } while ((Char >= '0') && (Char <= '9') ||
                    (toupper(Char) >= 'A') && (toupper(Char) <= 'Z'));
            Symbol = text;
        }
    } break;
```

*text ::= letter { letter | digit }*

*letter ::= A | B | C | E | G | H | J | K | L | M |  
N | P | R | S | T | V | W | X | Y | Z*

*digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9*

# Пример за анализ

```
// Целочислена константа
```

```
case '0'-'9': {
```

```
    Constant = 0;
```

```
    do {
```

```
        digit = (unsigned int)(Char) - (unsigned int)('0');
```

```
        if ( (Constant < ( MAXINTEGER / 10 ) )
```

```
            || ( (Constant == (MAXINTEGER / 10 ) )
```

```
                && ( digit = (MAXINTEGER % 10 ) ) ) ) {
```

```
                    Constant = ( 10 * Constant) + digit;
```

```
        } else {
```

```
            error("error: Int constant too large");
```

```
            Constant = 0;
```

```
        }
```

```
        GetNextChar();
```

```
    } while ((Char >= '0') && (Char = '9'));
```

```
    Symbol = intconst;
```

```
} break;
```

*integer ::= digit { digit }*

*digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9*

# Пример за анализ

```
// Едно-символен оператор
case '.': {
    Symbol = period;
    GetNextChar();
} break;
case ';': {
    Symbol = semicolon;
    GetNextChar();
} break;
case '"': {
    Symbol = quotas;
    GetNextChar();
} break;
// Други недопустими символи
default: {
    Symbol = othersy;
    GetNextChar();
} break;
} // switch
}
```

*record* ::= *field* { ; *field* } .  
*string* ::= " *text* "

# Реализация

```
void DataFile(void) {
    Record();
    while ((Symbol==intconst) || (Symbol==quotas)) {
        GetNextSymbol();
        Record();
    }
}

void Record() {
    Field();
    while (Symbol==semicolon) {
        accept(semicolon);
        Field();
    }
    expect(period);
}

...
```

*datafile ::= record { record }*  
*record ::= field { ; field } .*  
*field ::= integer | string*  
*integer ::= digit { digit }*  
*string ::= " text "*  
*text ::= letter { letter | digit }*  
*letter ::= A | B | C | E | G | H | J | K | L | M |*  
*N | P | R | S | T | V | W | X | Y | Z*  
*digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9*

# Реализация

```
void Field() (
    if (accept(intconst)) {
        ;
    } else if (accept(quotas)) {
        expect(text);
        expect(quotas);
    } else {
        error("field: expects intconst or string");
    }
}
```

```
datafile    ::= record { record }
record      ::= field { ; field } .
Field       ::= integer | string
integer     ::= digit { digit }
string      ::= " text "
text ::= letter { letter | digit }
letter ::= A | B | C | E | G | H | J | K | L | M |
          N | P | R | S | T | V | W | X | Y | Z
digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

# Стартиране

```
int main () {  
    GetNextSymbol();  
    DataFile();  
}
```

# Въпроси?