

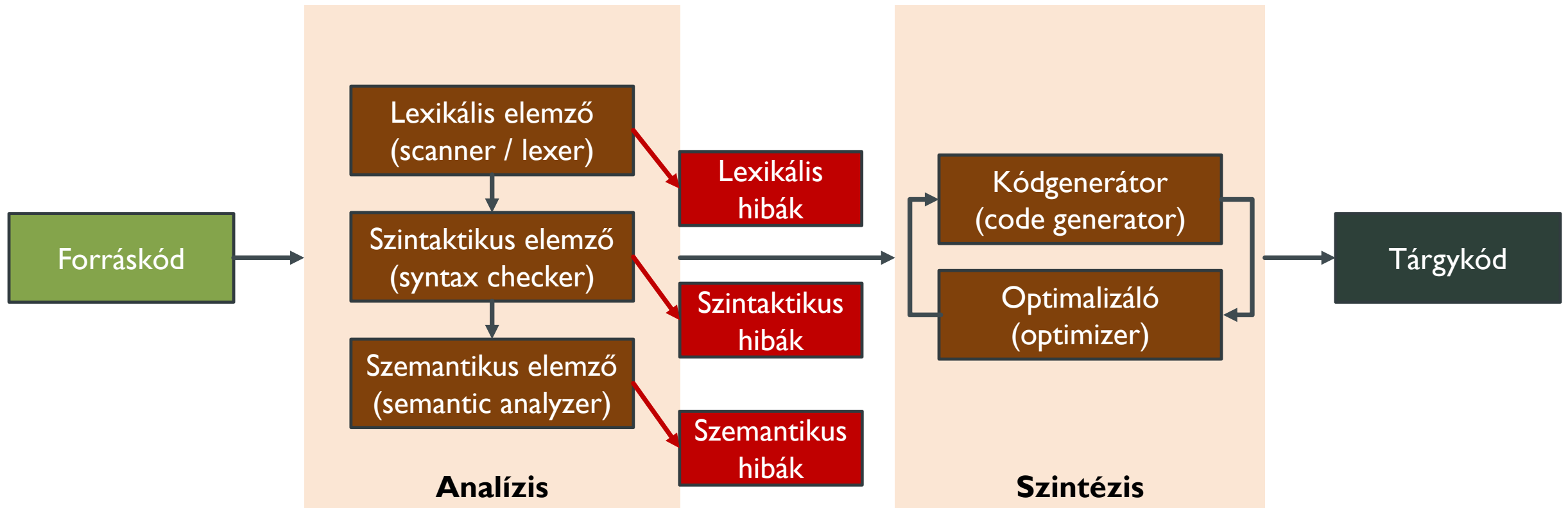


LEXIKÁLIS ELEMZÉS

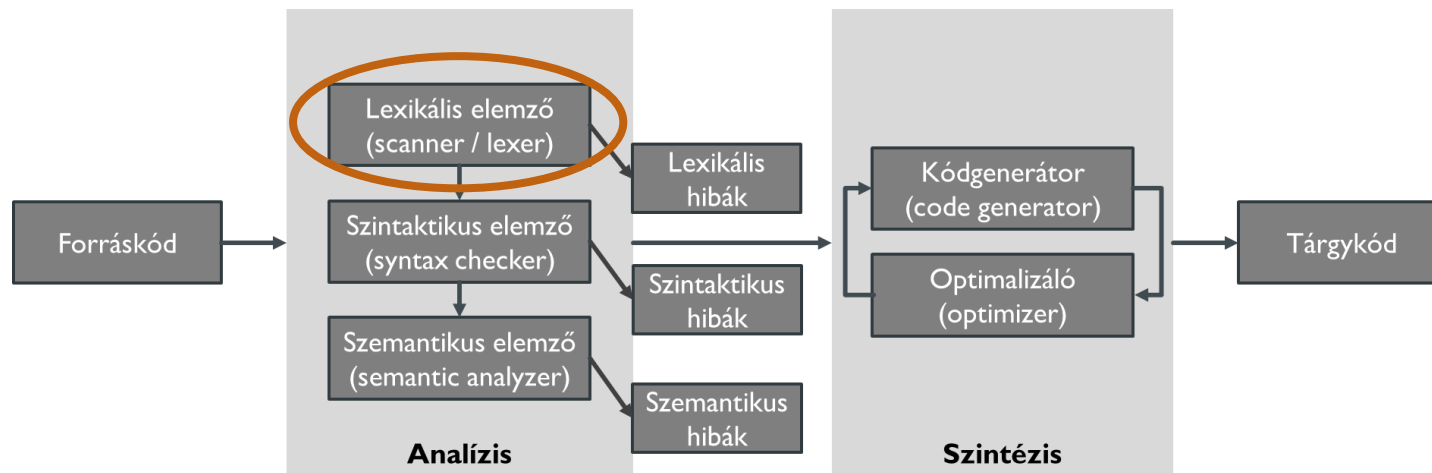
FORMÁLIS NYELVEK ÉS FORDÍTÓPROGRAMOK ALAPJAI

Dévai Gergely
ELTE

A FORDÍTÓPROGRAMOK LOGIKAI FELÉPÍTÉSE



LEXIKÁLIS ELEMZŐ



- *Feladat:*
A forrásszöveg elemi egységekre tagolása
- *Bemenet:*
Karaktersorozat
- *Kimenet:*
Lexikális elemek (tokenek) sorozata + lexikális hibák
- *Eszközök:*
Reguláris kifejezések, véges determinisztikus automaták

`x = x + 1;`

Változó ('x'), Operátor ('='), Változó ('x'), Operátor ('+'), Literál (1), Utasításvég

MIK LEGYENEK A TOKENEK?

x = (x + 2) * 3;

utasítás ?

x = (x + 2) * 3;

kifejezés

utasításvég ?

x = (x + 2) * 3;

bal o. op. (=) jobb o. utasításvég ?

x = (x + 2) * 3;

azonosító (x) op. (=) nyitó azonosító (x) op. (+) literál (2) csukó op. (*) literál (2) utasításvég ?

MIK LEGYENEK A TOKENEK?

x = (x + 2) * 3 ;

utasítás

Aminek a belső
szerkezete fontos,
nem lehet token!

Aminek a formája
nem írható le
reguláris kifejezéssel
(pl. helyes zárójelezés),
nem lehet token!

x = (x + 2) * 3 ;

bal o. op. (=) jobb o. utasításvég

x = (x + 2) * 3 ;

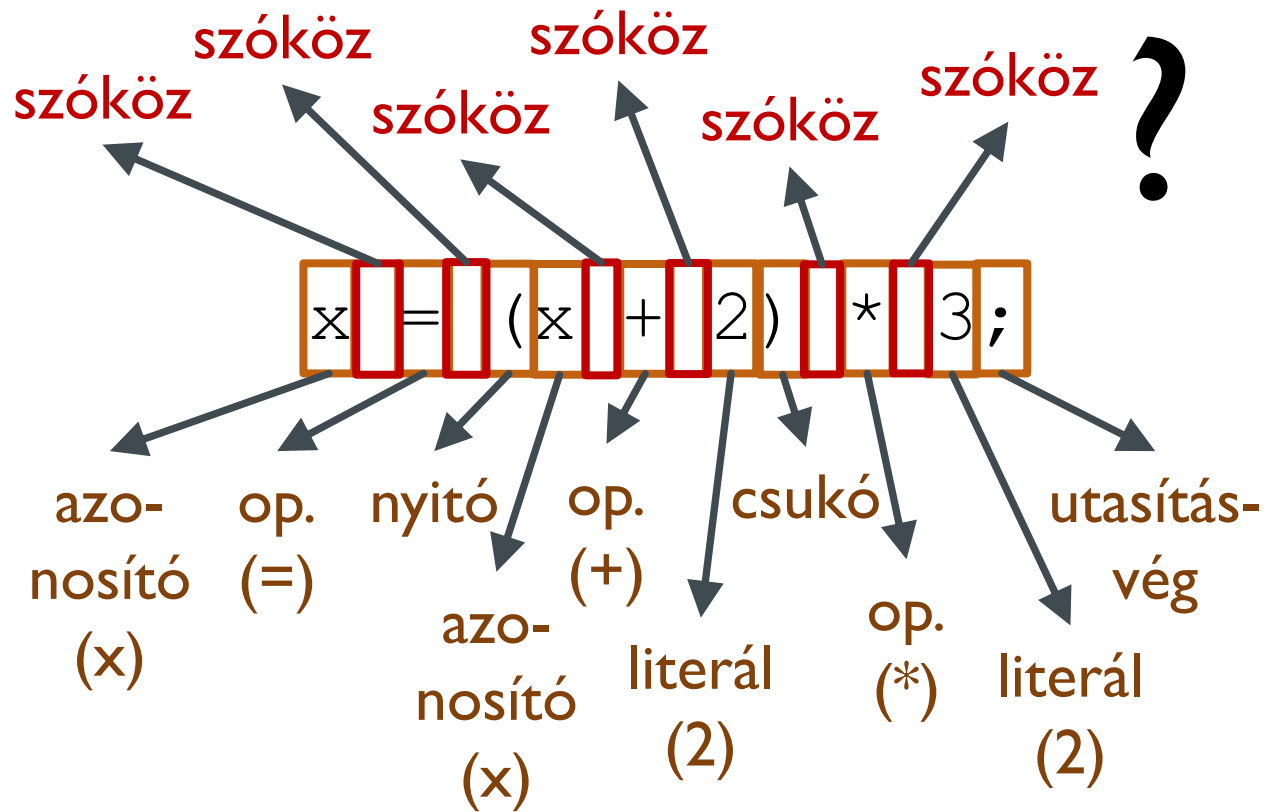
kifejezés

utasításvég

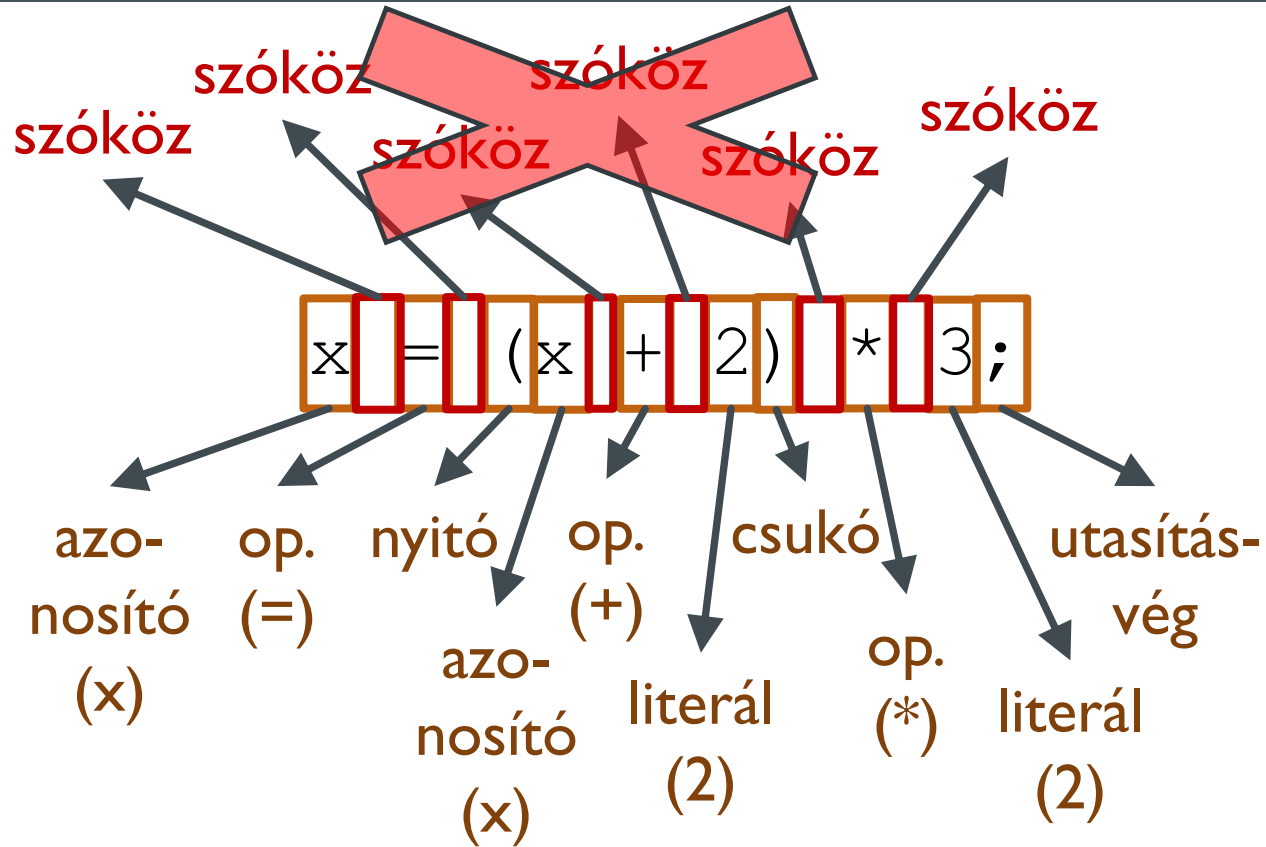
x = (x + 2) * 3 ;

azonosító (x) op. (=) nyitó op. (+) csukó op. (*) utasítás-
vég
azonosító (x) literál (2) literál (2)

SZÓKÖZ, TABULÁTOR, SORVÉGE





SZÓKÖZ, TABULÁTOR, SORVÉGE



- A legtöbb programozási nyelvénél a szóközök, tabok és újsorok **nem alkotnak tokeneket**, csak más tokenek elválasztására valók.
- A lexikális elemzőnek fel kell ismernie ezeket, de nem kell továbbítania a szintaktikus elemző felé.
- Behúzásra érzékeny nyelvek (pl. Python, Haskell):
 - A sorok behúzását kell számontartani
 - Növekvő behúzás: blokk-nyitó token
 - Csökkenő behúzás: blokk-záró token
- Érdekesség: Whitespace programozási nyelv

DEFINÍCIÓ REGULÁRIS KIFEJEZÉSEKKEL

Reguláris kifejezés	Példák	Token típus
<code>while</code>	<i>while</i>	While kulcsszó
<code>[a-zA-Z][a-zA-Z0-9_]*</code>	<i>x, apple, 23, list_length</i>	Azonosító
<code>[+-]?[0-9]+</code>	<i>0, 123, -2, +100</i>	Egész számliterál
<code>[\t\n]+</code>	(szóközök, tabok, sorvégék nem üres sorozata)	
<code>"//".*</code>	<i>// Ez egy megjegyzés</i>	

TOKENIZÁLÁS

```
while x<10 ...
```

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while		
[a-zA-Z][a-zA-Z0-9_]*		
[+-]?[0-9]+		
[\t\n]+		
"<"		

TOKENIZÁLÁS

```
while x<10 ...
```

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while	w	
[a-zA-Z][a-zA-Z0-9_]*	w	w
[+-]?[0-9]+		
[\t\n]+		
"<"		

TOKENIZÁLÁS

```
while x<10 ...
```

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while	w	
[a-zA-Z][a-zA-Z0-9_]*	w	w
[+-]?[0-9]+		
[\t\n]+		
"<"		

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**
A leghosszabban illeszkedő karaktersorozatból képzünk tokenet.
Hiába helyes azonosító szimbólum a „w”, mégis folytatni kell a keresést.

TOKENIZÁLÁS

```
while x<10 ...
```

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**
A leghosszabban illeszkedő karaktersorozatból képzünk tokenet.

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while	wh	
[a-zA-Z][a-zA-Z0-9_]*	wh	wh
[+-]?[0-9]+		
[\t\n]+		
"<"		

TOKENIZÁLÁS

```
while x<10 ...
```

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**
A leghosszabban illeszkedő karaktersorozatból képzünk tokenet.

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while	whi	
<code>[a-zA-Z][a-zA-Z0-9_]*</code>	whi	whi
<code>[+-]?[0-9]+</code>		
<code>[\t\n]+</code>		
<code>"<"</code>		

TOKENIZÁLÁS

```
while x<10 ...
```

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**
A leghosszabban illeszkedő karaktersorozatból képzünk tokenet.

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while	whil	
[a-zA-Z][a-zA-Z0-9_]*	whil	whil
[+-]?[0-9]+		
[\t\n]+		
"<"		

TOKENIZÁLÁS

```
while x<10 ...
```

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while	while	while
[a-zA-Z][a-zA-Z0-9_]*	while	while
[+-]?[0-9]+		
[\t\n]+		
"<"		

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**
A leghosszabban illeszkedő karaktersorozatból képzünk tokenet.
Továbbra is a leghosszabb illeszkedést keressük, meg kell vizsgálni a következő karaktert is.

TOKENIZÁLÁS

```
while x<10 ...
```

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while		while
[a-zA-Z][a-zA-Z0-9_]*		while
[+-]?[0-9]+		
[\t\n]+		
"<"		

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**
A leghosszabban illeszkedő karaktersorozatból képzünk tokenet.

A szóközzel együtt már semmi sem illeszkedik. Vissza kell lépni a legutóbbi illeszkedéshez.

De a „while kulcsszó” és az „azonosító” is illeszkedik. Melyik nyer?

TOKENIZÁLÁS

```
while x<10 ...
```

While

kulcsszó

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while		while
<code>[a-zA-Z][a-zA-Z0-9_]*</code>		while
<code>[+-]?[0-9]+</code>		
<code>[\t\n]+</code>		
<code>"<"</code>		

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**
A leghosszabban illeszkedő karaktersorozatból képzünk tokenet.
- **Prioritás elve:**
Ha a leghosszabban illeszkedő karaktersorozat több reguláris kifejezésre is illeszkedhet, a sorrendben korábban álló „nyer”.

TOKENIZÁLÁS

```
while x<10 ...
```

While

kulcsszó

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while		
[a-zA-Z][a-zA-Z0-9_]*		
[+-]?[0-9]+		
[\t\n]+	(szóköz)	(szóköz)
"<"		

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**
A leghosszabban illeszkedő karaktersorozatból képzünk tokenet.
- **Prioritás elve:**
Ha a leghosszabban illeszkedő karaktersorozat több reguláris kifejezésre is illeszkedhet, a sorrendben korábban álló „nyer”.

TOKENIZÁLÁS

while **x** < 10 ...

While

kulcsszó

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while		
[a-zA-Z][a-zA-Z0-9_]*		
[+-]?[0-9]+		
[\t\n]+		(szóköz)
"<"		

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**
A leghosszabban illeszkedő karaktersorozatból képzünk tokenet.
- **Prioritás elve:**
Ha a leghosszabban illeszkedő karaktersorozat több reguláris kifejezésre is illeszkedhet, a sorrendben korábban álló „nyer”.

TOKENIZÁLÁS

```
while x<10 ...
```

While

kulcsszó

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while		
[a-zA-Z][a-zA-Z0-9_]*		
[+-]?[0-9]+		
[\t\n]+		(szóköz)
"<"		

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**
A leghosszabban illeszkedő karaktersorozatból képzünk token.
- **Prioritás elve:**
Ha a leghosszabban illeszkedő karaktersorozat több reguláris kifejezésre is illeszkedhet, a sorrendben korábban álló „nyer”.

A szóközöket felismerjük, de nem lesz belőlük token.

TOKENIZÁLÁS

while **x** < 10 ...

While

kulcsszó

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while		
[a-zA-Z][a-zA-Z0-9_]*	x	x
[+-]?[0-9]+		
[\t\n]+		
"<"		

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**
A leghosszabban illeszkedő karaktersorozatból képzünk tokenet.
- **Prioritás elve:**
Ha a leghosszabban illeszkedő karaktersorozat több reguláris kifejezésre is illeszkedhet, a sorrendben korábban álló „nyer”.

TOKENIZÁLÁS

while x < 10 ...

While

kulcsszó

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while		
[a-zA-Z][a-zA-Z0-9_]*		x
[+-]?[0-9]+		
[\t\n]+		
"<"		

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**
A leghosszabban illeszkedő karaktersorozatból képzünk tokenet.
- **Prioritás elve:**
Ha a leghosszabban illeszkedő karaktersorozat több reguláris kifejezésre is illeszkedhet, a sorrendben korábban álló „nyer”.

TOKENIZÁLÁS

while **x** < 10 ...

While
kulcsszó

Azono-
sító

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while		
<code>[a-zA-Z][a-zA-Z0-9_]*</code>		x
<code>[+-]?[0-9]+</code>		
<code>[\t\n]+</code>		
<code>"<"</code>		

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**
A leghosszabban illeszkedő karaktersorozatból képzünk tokenet.
- **Prioritás elve:**
Ha a leghosszabban illeszkedő karaktersorozat több reguláris kifejezésre is illeszkedhet, a sorrendben korábban álló „nyer”.

TOKENIZÁLÁS

while x < 10 ...

While
kulcsszó

Azono-
sító

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while		
[a-zA-Z][a-zA-Z0-9_]*		
[+-]?[0-9]+		
[\t\n]+		
"<"	<	<

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**
A leghosszabban illeszkedő karaktersorozatból képzünk tokenet.
- **Prioritás elve:**
Ha a leghosszabban illeszkedő karaktersorozat több reguláris kifejezésre is illeszkedhet, a sorrendben korábban álló „nyer”.

TOKENIZÁLÁS

while x < 10 ...

While
kulcsszó

Azono-
sító

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while		
[a-zA-Z][a-zA-Z0-9_]*		
[+-]?[0-9]+		
[\t\n]+		
"<"		<

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**
A leghosszabban illeszkedő karaktersorozatból képzünk tokenet.
- **Prioritás elve:**
Ha a leghosszabban illeszkedő karaktersorozat több reguláris kifejezésre is illeszkedhet, a sorrendben korábban álló „nyer”.

TOKENIZÁLÁS

while x < 10 ...

While Azono- Kisebb
kulcsszó sító operátor

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while		
[a-zA-Z][a-zA-Z0-9_]*		
[+-]?[0-9]+		
[\t\n]+		
"<"		<

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**
A leghosszabban illeszkedő karaktersorozatból képzünk tokenet.
- **Prioritás elve:**
Ha a leghosszabban illeszkedő karaktersorozat több reguláris kifejezésre is illeszkedhet, a sorrendben korábban álló „nyer”.

TOKENIZÁLÁS

while x < 10 ...

While Azono- Kisebb
kulcsszó sító operátor

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while		
[a-zA-Z][a-zA-Z0-9_]*		
[+-]?[0-9]+		
[\t\n]+		
"<"		

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**
A leghosszabban illeszkedő karaktersorozatból képzünk tokenet.
- **Prioritás elve:**
Ha a leghosszabban illeszkedő karaktersorozat több reguláris kifejezésre is illeszkedhet, a sorrendben korábban álló „nyer”.

TOKENIZÁLÁS

while **x** **<** **10** ...

While Azono- Kisebb
kulcsszó sító operátor

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while		
[a-zA-Z][a-zA-Z0-9_]*		
[+-]?[0-9]+	10	10
[\t\n]+		
"<"		

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**
A leghosszabban illeszkedő karaktersorozatból képzünk tokenet.
- **Prioritás elve:**
Ha a leghosszabban illeszkedő karaktersorozat több reguláris kifejezésre is illeszkedhet, a sorrendben korábban álló „nyer”.

TOKENIZÁLÁS

while x < 10 ...

While Azono- Kisebb
kulcsszó sító operátor

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while		
[a-zA-Z][a-zA-Z0-9_]*		
[+-]?[0-9]+		10
[\t\n]+		
"<"		

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**
A leghosszabban illeszkedő karaktersorozatból képzünk tokenet.
- **Prioritás elve:**
Ha a leghosszabban illeszkedő karaktersorozat több reguláris kifejezésre is illeszkedhet, a sorrendben korábban álló „nyer”.

TOKENIZÁLÁS

while x < 10 ...

While Azono- Kisebb Szám-
kulcsszó sító operátor literál

Reguláris kifejezés	Illeszkedő prefix	Legutóbbi illeszkedés
while		
[a-zA-Z][a-zA-Z0-9_]*		
[+-]?[0-9]+		10
[\t\n]+		
"<"		

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**
A leghosszabban illeszkedő karaktersorozatból képzünk tokenet.
- **Prioritás elve:**
Ha a leghosszabban illeszkedő karaktersorozat több reguláris kifejezésre is illeszkedhet, a sorrendben korábban álló „nyer”.

A LEXIKÁLIS ELEMZÉS ELVEI

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**

A leghosszabban illeszkedő
karaktersorozatból képzünk
tokenet.

- **Prioritás elve:**

Ha a leghosszabban illeszkedő
karaktersorozat több reguláris
kifejezésre is illeszkedhet, a
sorrendben korábban álló „nyer”.

wh ile while whilewhile

A LEXIKÁLIS ELEMZÉS ELVEI

A lexikális elemzés elvei:

- **Leghosszabb illeszkedés elve:**

A leghosszabban illeszkedő karaktersorozatból képzünk tokenet.

- **Prioritás elve:**

Ha a leghosszabban illeszkedő karaktersorozat több reguláris kifejezésre is illeszkedhet, a sorrendben korábban álló „nyer”.

wh|ile|while|whilewhile

Azono-
sító

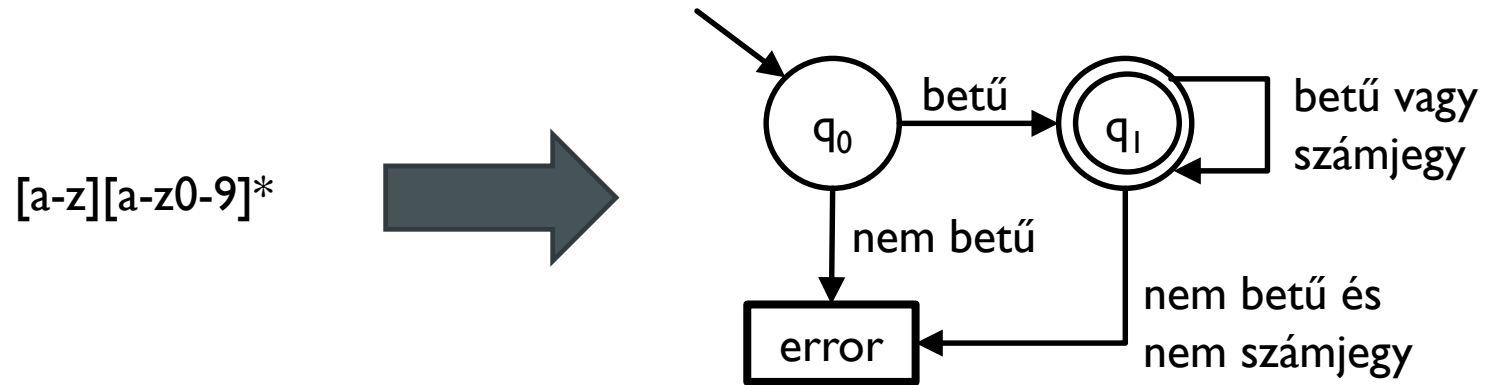
Azono-
sító

While
kulcsszó

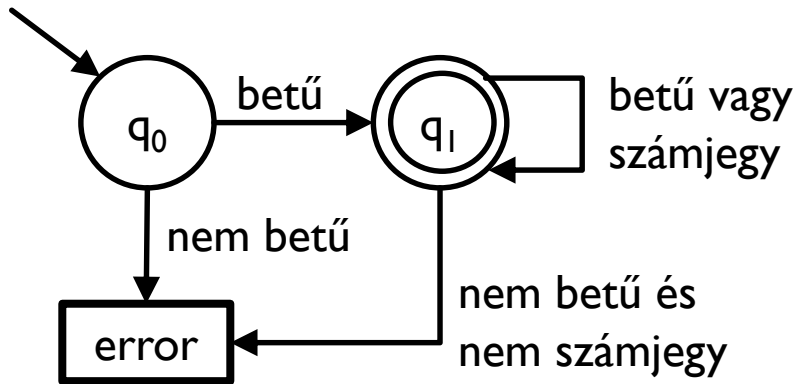
Azono-
sító

IMPLEMENTÁCIÓ

- A reguláris kifejezések átalakíthatók véges determinisztikus automatává.

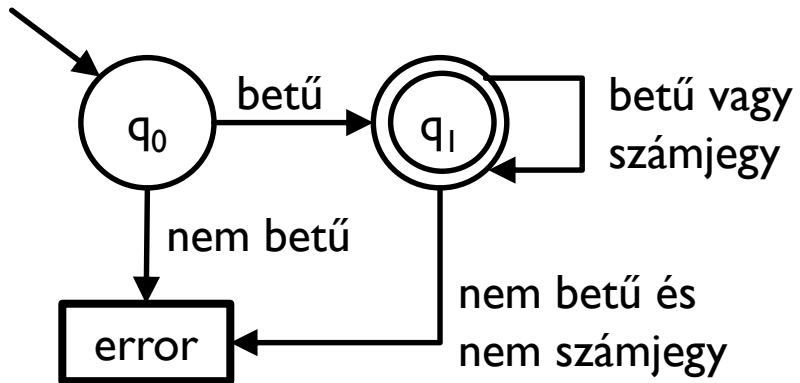


VDA IMPLEMENTÁCIÓJA ELÁGAZÁSOKKAL



```
void vda::process(char next) {  
    if(state == q0) {  
        if(next == 'a' || ...)   
            state = q1;  
        else  
            state = error;  
    } else if(state == q1) {  
        if(next == 'a' || ... || next == '0' || ...)   
            state = q1;  
        else  
            state = error;  
    }  
}
```

VDA IMPLEMENTÁCIÓJA TÁBLÁZATTAL

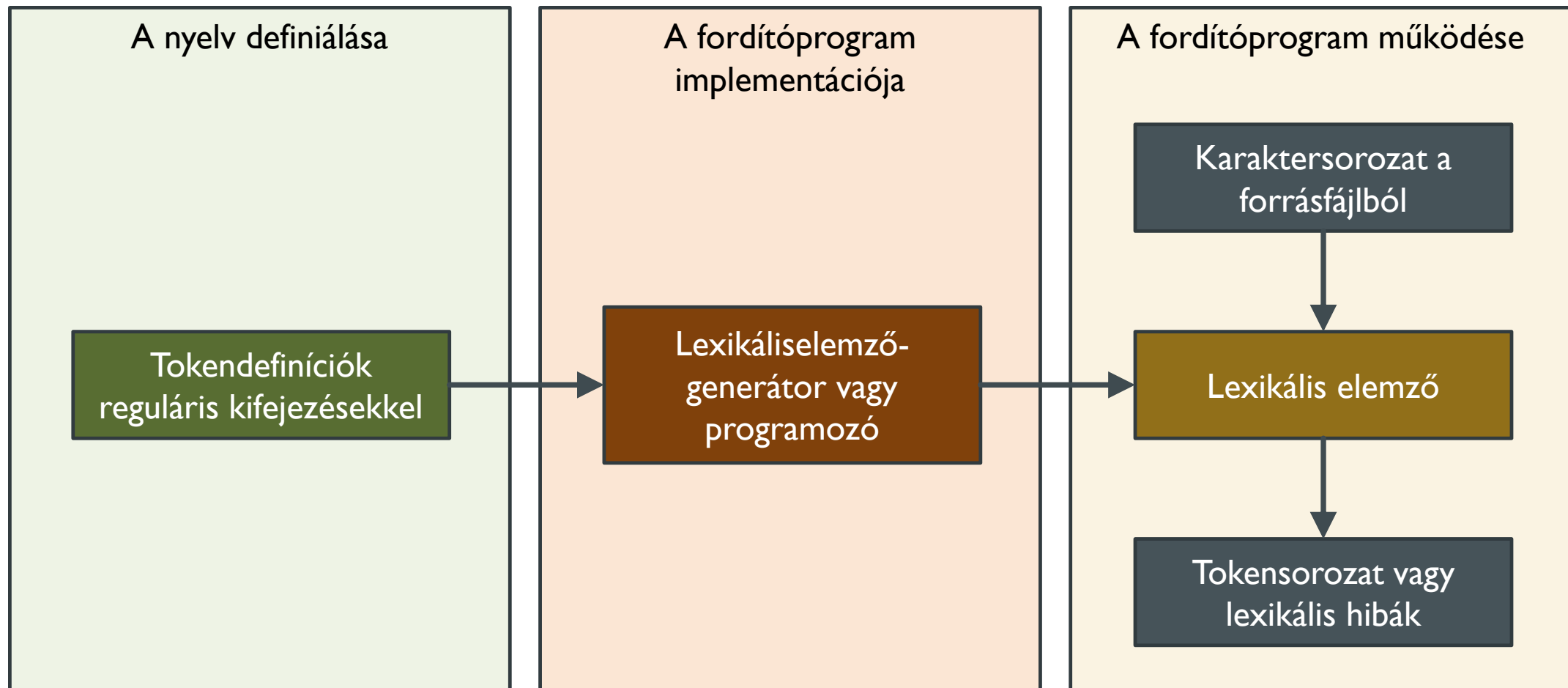


vda::transitions =

	q_0	q_1
'a'	q_1	q_1
...		
'0'	error	q_1
...		
other	error	error

```
void vda::process(char next) {  
    if(state != error)  
        state = transitions[state][next];  
}
```

LEXIKÁLIS ELEMZŐ LÉTREHOZÁSA



TOKENEKHEZ CSATOLT INFORMÁCIÓK

- A felismert tokenekhez a lexikális elemző kiegészítő információkat csatol.
- *Minden tokenhez*: pozíció (első karakter sor és oszlop, utolsó karakter sor és oszlop)
 - Hibaüzenetekhez, refaktoráláshoz szükséges
- *Azonosítókhoz*: az azonosító szövege
 - Szemantikus elemzéshez szükséges
- *Literálokhoz*: a literál értéke
 - Kódgeneráláshoz, kódoptimalizációhoz szükséges

LEXIKÁLIS HIBÁK

- Lexikális hiba esetén hibajelzést ad a fordító, és folytatja az elemzést.
- Illegális karakter: A nyelv ábécéjébe nem tartozó karakter az inputszövegben.
 - Az addig felépített token kiadása, ha volt illeszkedés
 - Az illegális karaktert követő karakterrel folytatódik az elemzés
- Lezáratlan sztring:
 - A sor végén derül ki
 - A következő sorban folytatódik az elemzés
- Lezáratlan többsoros megjegyzés:
 - A fájl végén derül ki
 - Nincs további elemzés

```
if(str == "apple) {  
    cout << str;  
}  
...
```

```
/* I just comment out this 'if',  
because the compiler does not accept it...  
if(str == "apple) {  
    cout << str;  
}  
...
```