

Stefan Sokołowski

JĘZYKI FORMALNE I METODY KOMPILACJI

wykład 14

Inst. Informatyki Stosowanej, ANS
Elbląg, 2025/2026

Wykład 14, str. 1

Analiza leksykalna

Dotąd prymitywna:

```
#define YYSTYPE int
. . . . .
YYSTYPE yylex (void) {
    int zn;
    do { zn = getchar(); } while (isspace(zn));
    if (zn == EOF) return 0;
    else return zn;
}
```

stanowiąca część parsera.

Można (warto!) wyodrębnić ją w osobny moduł.

— np. po to, żeby leksemy mogły być wieloznakowe. . .

~~(n){s=s+n; n=n-1;}~~

while (n != 0){s += n; n--;} }

Analiza leksykalna

LEX



FLEX (= Fast LEX)

www.cs.princeton.edu/~appel/modern/c/software/flex

wyszukuje wzorce **regularne** i wykonuje akcje;
w najprostszym przypadku zastępuje

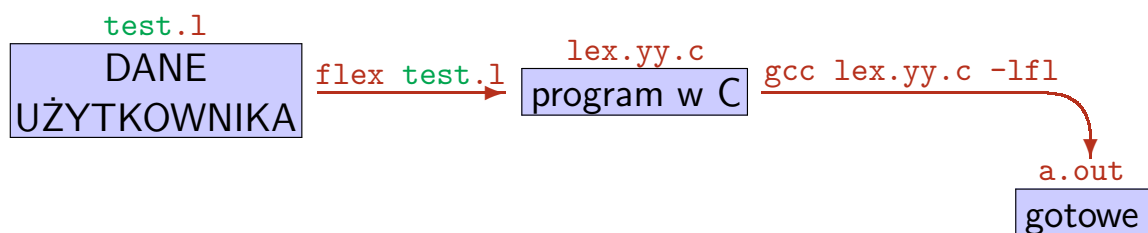
Dane użytkownika
— co robić z tekstem

```
definicje
%%
reguły
%%
kod użytkownika
```

Najważniejsza część: **reguły** składające się z

- **wzorca** — jego napotkanie w tekście wejściowym ma spowodować jakąś akcję;
- **akcji** — czynności, które należy podjąć po rozpoznaniu wzorca w tekście wejściowym (opisane w C).

Analiza leksykalna



Przykład:

Jeśli dane użytkownika:

```
test.l
[0-9]+      printf("LICZBA");
[+\-]      printf("OP");
[A-Z][a-z]+ printf("KTOŚ");
```

to

tekst <pre>kto wie, ile jest (15 - 8) + 6 ? Jasio wie, Szymek nie wie.</pre>	./a.out < tekst	wynik <pre>kto wie, ile jest (LICZBA OP LICZBA) OP LICZBA ? KTOŚ wie, KTOŚ nie wie.</pre>
---	----------------------------------	--

Działanie FLEXa

- Akcje FLEXa *mogą* (ale nie muszą) polegać na zastępowaniu fragmentów tekstu innymi fragmentami. Mogą to być dowolne komendy języka C.
- Wzorce muszą być *wyrażeniami regularnymi*. FLEX rozumie wiele rozmaitych rodzajów wyrażeń regularnych.
- Część definicyjna (przed regułami) może służyć na przykład do tego, żeby definiować typy i funkcje potrzebne w akcjach reguł, podeklarować zmienne pomocnicze i wykonać inicjalizację.
- Część kodu użytkownika (za regułami) może służyć na przykład do tego, żeby porządnie zakończyć działanie analizatora.
- Kod w języku C dostarczony przez użytkownika jest przez FLEXa umieszczany bez zmian w gotowym analizatorze `lex.yy.c`. Jeśli są w nim błędy, to zostaną one zasygnalizowane dopiero w czasie kompilacji translatorem `gcc`.

BISON z FLEXem

Translator utworzony przez Bisona czyta tekst wejściowy (w języku źródłowym) leksem za leksemem — każde wywołanie funkcji

```
YYSTYPE yylex (void)
```

(typ `YYSTYPE` sami definiujemy) dostarcza mu jeden leksem.

Jeśli mamy leksemy więcej niż 1-znakowe, to najwygodniej zdefiniować tę funkcję przez Flexa.

BISON z FLEXem

- w danych do Bisona `plik.y` deklarujemy każdy taki leksem:
`%token PETLA`
- tam, gdzie leksem ma wystąpić w gramatyce, piszemy tą zadeklarowaną nazwę:
`PETLA '(' zmienna ')' '{' komendy '}'`
- przetwarzamy dane do Bisona komendą:
`bison -d plik.y`
 — to tworzy program w C `plik.tab.c` oraz plik nagłówkowy `plik.tab.h`
- we wstępnej części danych do Flexa `plik.1` odwołujemy się do pliku nagłówkowego:
`#include "plik.tab.h"`
 podajemy reguły rozpoznawania wieloznakowych leksemów:
`"WHILE" { return PETLA; }`
 i **pomijamy** `main` dla tego pliku
- przetwarzamy dane do Flexa komendą:
`flex plik.1`
 — to tworzy program w C `lex.yy.c`
- oba programy w C kompilujemy łącznie, razem z opcją biblioteki Flexa:
`gcc plik.tab.c lex.yy.c -lfl`
 — to tworzy program w C `a.out`, stanowiący żądany kompilator

BISON z FLEXem

