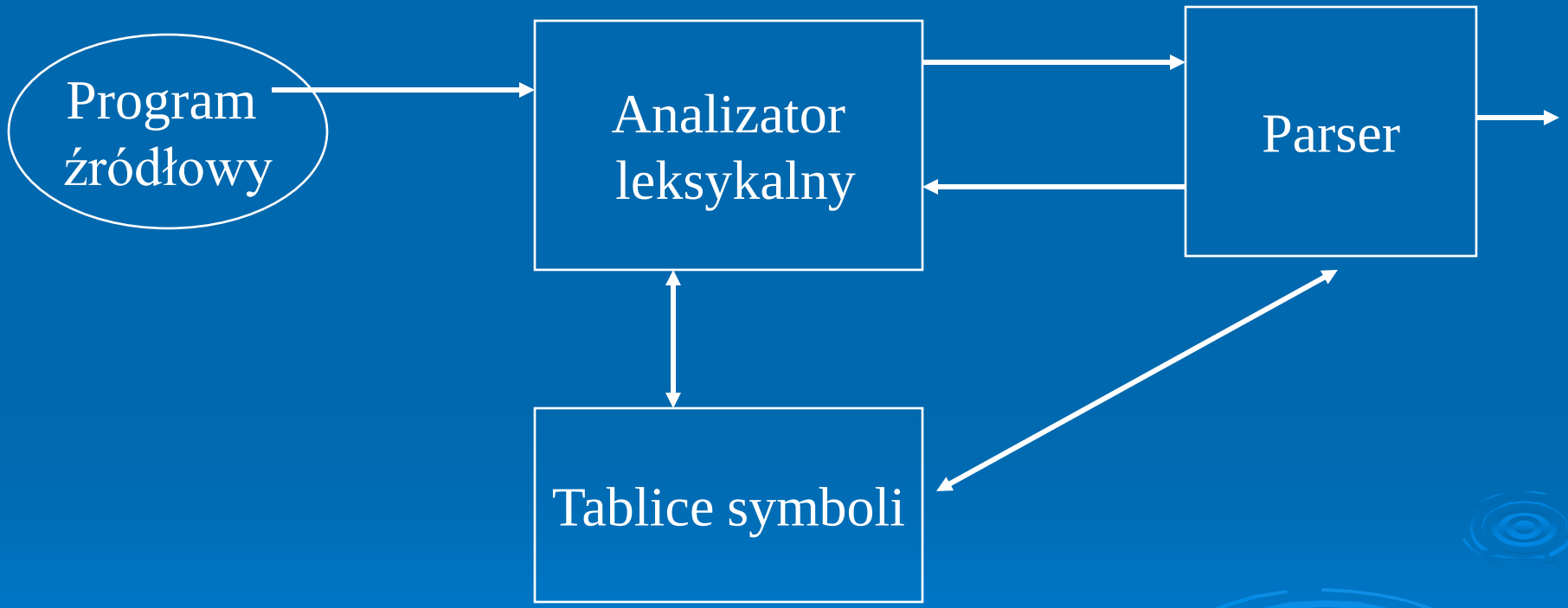
Metody kompilacji
Wykłady 4-5

Analiza Leksykalna

Wstęp

- Analizator leksykalny odczytuje znaki z wejścia, rozpoznaje leksemy i produkuje tokeny.
- Wraz z symbolem terminalnym, który jest używany przez parser, token zawiera dodatkowe informacje w postaci wartości atrybutów.
- Token jest to terminal wraz z dodatkową informacją.

ANALIZA LEKSYKALNA



Analiza Leksykalna

Wstęp

- Sekwencja znaków wejściowych, która zawiera pojedynczy token nazywa się leksemem.
- Tak więc, można powiedzieć, że analizator leksykalny izoluje parser od reprezentacji znakowej symbolu, czyli parser dostaje tokeny, nie znaki.

ANALIZA LEKSYKALNA

- Głównym zadaniem analizatora leksykalnego jest wczytywanie znaków z programu źródłowego, rozpoznawanie leksemów i produkowanie tokenów(znaczników) dla każdego leksemu.

ANALIZA LEKSYKALNA

- Inne zadania:
- Wyeliminowanie komentarzy i „znaków białych”: spacja, znak nowej linii, znak tabulacji.
- Kolejnym zadaniem jest współudział w obsłudze błędów generowanych przez kompilator.

ANALIZA LEKSYKALNA

- Na przykład, analizator leksykalny może śledzić liczbę linijek kodu, liczbę znaków każdej linijki i przekazywać te dane do kompilatora.

ANALIZA LEKSYKALNA

Dlaczego analizator leksykalny jest tworzony osobno

- 1. Prostota projektowania analizatora leksykalnego(w stosunku do projektowania analizatora syntaktycznego) jest najważniejszym czynnikiem.

ANALIZA LEKSYKALNA

Dlaczego analizator leksykalny jest tworzony osobno

2. Zwiększona wydajność kompilatora.

Opracowane są wyspecjalizowane bardzo wydajne techniki analizy leksykalnej.

Ponadto techniki buforowania do wczytywania znaków wejściowych mogą przyspieszyć kompilator znacząco.

ANALIZA LEKSYKALNA

Dlaczego analizator leksykalny jest tworzony osobno

3. Kompilator ma zwiększoną przenośność czyli może być stosowany na różnych platformach.

Specyficzne dla urządzeń wejścia osobliwości mogą być uwzględnione tylko w analizatorze leksykalnym, pozostałe części kompilatora zostają bez zmian.

ANALIZA LEKSYKALNA

Tokeny(Tokens), Wzorce(Patterns), Leksemy(Lexemes)

- Token jest parą składającą się z nazwy symbolicznej i opcjonalnej wartości atrybutu.

ANALIZA LEKSYKALNA

Tokeny, Wzorce, Leksemy

- Wzorzec jest opisem postaci, którą leksem może przyjąć.
- W przypadku słów kluczowych, wzorzec jest po prostu ciągiem znaków, które tworzą słowa kluczowe.

ANALIZA LEKSYKALNA

Tokeny, Wzorce, Leksemy

- Leksem jest ciągiem znaków w programie źródłowym, który pasuje do jakiegoś wzorca.
- Jest to niepodzielny element programu, dlatego jest nazywany również atomem.

ANALIZA LEKSYKALNA
Tokeny, Wzorce, Leksemy

Token	Leksem	Wzorzec
if	if	if
id	abc, n, count,…	litera +cyfra
NUMBER	3.14, 1000	stała numeryczna
;	;	;

ANALIZA LEKSYKALNA

Tokeny, Wzorce, Leksemy

- Dla kodu: printf ("Total = %d\n”, score) ;
  zarówno printf i score są leksemami pasującymi do wzorca dla tokena **id,**
  "Total = %d\n” jest to leksem pasujący do literału(dosłowny tekst).

ANALIZA LEKSYKALNA

Tokeny, Wzorce, Leksemy

- W wielu językach programowania, następujące przypadki obejmują większość tokenów:

  1. Jeden token dla każdego słowa kluczowego.

  Wzorzec dla słowa kluczowego jest taki sam jak słowo kluczowe.

ANALIZA LEKSYKALNA

Tokeny, Wzorce, Leksemy

2. Tokeny dla operatorów: indywidualnie dla każdego operatora lub jeden token dla grupy operatorów(przykładowo operatorów relacji).

ANALIZA LEKSYKALNA

Tokeny, Wzorce, Leksemy

3. Jeden token reprezentujący wszystkie identyfikatory.

4. Jeden lub więcej tokenów reprezentujących stałe, takie jak liczby i literały.

5. Tokeny dla każdego z symboli interpunkcyjnych, takich jak lewy i prawy nawiasy, przecinek, średnik.

ANALIZA LEKSYKALNA

Atrybuty tokenów

- Token posiada opcjonalne atrybuty.
- Najważniejszym przykładem jest token **id**, z którym musimy skojarzyć dużo informacji: typ danych, wymiar tablicy, liczba elementów tablicy, miejsce w programie, w którym zmienna się pojawia po raz pierwszy.

.

ANALIZA LEKSYKALNA

Atrybuty tokenów

- Atrybuty są przechowywane w tablicy symboli.
- Tak więc jednym z atrybutów jest również wskaźnik do wejścia tablicy symboli dla identyfikatora.

ANALIZA LEKSYKALNA

Atrybuty tokenów

Przykład: dla instrukcji przypisania w Fortranie

E = M * C ** 2

mamy następujący wynik produkowany przez analizator leksykalny:

ANALIZA LEKSYKALNA

Atrybuty tokenów

<id, pointer to symbol-table entry for E **>**

< assign_op **>**

<id, pointer to symbol-table entry for M **>**

<mult_op>

<id, pointer to symbol-table entry for C >

<exp-op>

<number , integer value 2 **>**

ANALIZA LEKSYKALNA

Błędy leksykalne (Lexical errors)

- Wykrywanie błędów

  fi(a==f(x)) …
- Raportowanie błędów
- Usuwanie błędów

ANALIZA LEKSYKALNA

Usuwanie błędów

- Załóżmy, że analizator leksykalny podczas rozpoznawania leksemu nie może kontynuować swojego działania, ponieważ żaden ze wzorców nie pasuje.
- Analizator może usunąć kolejne znaki z pozostałego wejścia, aż potrafi dopasować wczytywane znaki do jakiegoś wzorca.

Analiza Leksykalna

- Analizator leksykalny, który będziemy tworzyć, pozwala na rozpoznawanie liczb, identyfikatorów i „ znaków białych" (spacje, tabulatory i znaki nowej linii) w wyrażeniach.

Analiza Leksykalna

Schemat translacji

expr ->expr + term { print ('+')}
| *expr - term* { print('-') }
| *term*
*term -> term * factor* { print('*') }
| *term / factor* { print (‘/’)}
| *factor*
factor -> (*expr*)
| **num** { print (**num**. *value*) }
| **id** { print (**id**. *lexeme*) }

Analiza Leksykalna

Usuwanie znaków białych

- Większość języków umożliwia dowolną ilość przestrzeni białej między leksemami.

- Komentarze mogą być traktowane jako przestrzeń biała.

Analiza Leksykalna

Usuwanie znaków białych

- Jeśli przestrzeń biała jest eliminowana przez analizator leksykalny, to parser nie będzie musiał brać pod uwagę znaków białych.
- Alternatywą jest uwzględnienie znaków białych w gramatyce ale to znacznie zwiększa złożoność parsera.

Analiza Leksykalna

Pseudokod rozpoznawania i usuwania znaków białych:

for (; ; *peek* = next input character) {
  if (*peek* is a blank or a tab) do nothing;
  else if (*peek* is a newline) *line* = *line*+1;
  else break;
}

Analiza Leksykalna

Czytanie z wyprzedzeniem

- Analizator leksykalny może wymagać odczytywania znaków wejściowych z wyprzedzeniem, zanim zdecyduje jaki ma być leksem właściwy.
- Na przykład, analizator leksykalny dla C lub Javy po rozpoznaniu znaku > musi odczytać następny znak.

Analiza Leksykalna

Czytanie z wyprzedzeniem

- Jeśli następnym znakiem jest =, to znak > jest częścią sekwencji znaków >=, reprezentujących leksem (operator) "większe lub równe".
- Inaczej znak > sam tworzy leksem "większy niż"; w takim przypadku analizator leksykalny odczytał jeden znak za dużo.

Analiza Leksykalna

Czytanie z wyprzedzeniem

- Ogólne podejście do czytania znaków wejścia z wyprzedzeniem jest oparte na zastosowaniu bufora wejściowego, z którego analizator leksykalny może odczytać znak i zapisać go z powrotem.

Analiza Leksykalna

Czytanie z wyprzedzeniem

- Bufory wejściowe mogą być uzasadnione również względem efektywności analizatora ponieważ pobieranie ciągu znaków jest zwykle bardziej wydajne niż odczyt jednego znaku na raz.

Analiza Leksykalna

Czytanie z wyprzedzeniem

- Wskaźnik śledzi część wejścia, która już została przeanalizowana; przejście do poprzedniego znaku jest realizowane przez przesunięcie wskaźnika do tyłu.

Analiza Leksykalna

Czytanie z wyprzedzeniem

- Jeden znak odczytany z wyprzedzeniem zazwyczaj wystarcza, więc prostym rozwiązaniem jest użycie zmiennej, powiedzmy o nazwie *peek*, do przechowywania następnego znaku wejściowego.

Analiza Leksykalna

Czytanie z wyprzedzeniem

- Analizator leksykalny czyta naprzód tylko wtedy, gdy musi.
- Operatora „* „ można użyć aby nie odczytywać następnego znaku. W takich przypadkach wartością zmiennej *peek* jest znak spacji, który może być pominięty gdy analizator jest wywoływany aby znaleźć następny leksem.

Analiza Leksykalna

Rozpoznawanie stałych

- Za każdym razem, gdy w wyrażeniu pojawia się pojedyncza cyfra, rozsądnym wydaje się wczytywanie kolejnych cyfr celem rozpoznania liczby całkowitej ponieważ jest ona sekwencją cyfr.

Analiza Leksykalna

Rozpoznawanie stałych

- Stałe całkowite mogą być reprezentowane przez utworzenie symbolu terminalnego, powiedzmy o nazwie **num** dla każdej stałej, lub wprowadzając składnię stałych całkowitych do gramatyki.

Analiza Leksykalna

Rozpoznawanie stałych

- Praca łączenia cyfr w liczbę z reguły należy do zadań analizatora leksykalnego, więc liczby mogą być traktowane jako pojedyncze jednostki(leksemy) w trakcie parsowania i tłumaczenia kodu źródłowego.

Analiza Leksykalna

Rozpoznawanie stałych

- Kiedy pojawia się ciąg cyfr w strumieniu wejściowym, analizator leksykalny przekazuje do parsera token, który składa się z terminala **num** wraz z atrybutem – rozpoznanej liczby całkowitej.

Analiza Leksykalna

Rozpoznawanie stałych

- Dla napisu wejściowego

  31 + 28 + 59

  analizator leksykalny produkuje następujący wynik:

  <num,31> <+> <num,28> <+> <num,59>

Analiza Leksykalna

Pseudokod do rozpoznawania stałych

```
if ( peek holds a digit ) {
    v = 0;
    do {
          v = v * 10 + integer value of digit peek;
          peek = next input character;
    } while ( peek holds a digit ) ;
      return token (num, v);
}
```

Analiza Leksykalna

Rozpoznawanie identyfikatorów i słów kluczowych

- Większość języków używa słów kluczowych, na przykład:

  for, do, if , while, ….

  które są reprezentowane przez ciągi znaków.

Analiza Leksykalna

Rozpoznawanie identyfikatorów i słów kluczowych

- Ciągi znaków są również wykorzystywane do tworzenia nazw zmiennych, tablic, funkcji i t. d.
- Żeby uprościć parser, gramatyki traktują identyfikatory jako terminal, powiedzmy **id,** za każdym razem, gdy identyfikator pojawi się na wejściu.

Analiza Leksykalna

Rozpoznawanie identyfikatorów i słów kluczowych

- Na przykład dla wejścia:

  count = count + increment;

parser dostanie od analizatora leksykalnego następujący ciąg tokenów:

id = id + id.

Dla tokena **id** atrybutem jest leksem reprezentujący identyfikator.

Analiza Leksykalna

Rozpoznawanie identyfikatorów i słów kluczowych

Dla wejścia

count = count + increment;

Analizator leksykalny produkuje:

‹**id**, "count "› ‹=› ‹**id**, "count "› ‹+›

‹**id**, "increment "› ‹ ; ›

Analiza Leksykalna

Rozpoznawanie identyfikatorów i słów kluczowych

- Słowa kluczowe generalnie spełniają zasady tworzenia identyfikatorów, więc jest potrzebny mechanizm do podjęcia decyzji: leksem reprezentuje słowo kluczowe czy identyfikator.

Analiza Leksykalna

Rozpoznawanie identyfikatorów i słów kluczowych

- Problem jest łatwiejszy do rozwiązania, jeśli słowa kluczowe są zastrzeżone: nie mogą one być wykorzystywane jako identyfikatory.

  Ciąg znaków tworzy identyfikator jeżeli nie reprezentuje on słowo kluczowe.

Analiza Leksykalna

Rozpoznawanie identyfikatorów i słów kluczowych

- Analizator leksykalny rozwiązuje ten problem za pomocą tablicy do przechowywania ciągów znaków, czyli symboli.

Analiza Leksykalna

Rozpoznawanie identyfikatorów i słów kluczowych

- 1. Tablica symboli może izolować resztę kompilatora z reprezentacji ciągów, fazy kompilatora mogą korzystać z referencji lub wskaźnika do łańcucha w tabeli.
- Referencje/wskaźniki mogą być bardziej efektywne niż manipulowanie samymi ciągami.

Analiza Leksykalna

Rozpoznawanie identyfikatorów i słów kluczowych

- 2. Tablica symboli może być inicjalizowana słowami zarezerwowanymi.

Analiza Leksykalna

Rozpoznawanie identyfikatorów i słów kluczowych

- Kiedy analizator leksykalny rozpoznaje leksem, który może stanowić identyfikator, najpierw sprawdza, czy leksem jest przechowywany w tablicy symboli.
- Jeżeli tak, to zwraca token przechowywany w tablicy; w przeciwnym razie tworzy i zwraca token, którego pierwszym elementem jest **id**.

Analiza Leksykalna

Rozpoznawanie identyfikatorów i słów kluczowych

Tablica symboli może być zaimplementowana jako tablica haszująca z użyciem klasy o nazwie *Hashtable*.

Na przykład:

Hashtable words = **new** *Hashtable();*

Pseudokod do rozpoznawania identyfikatorów i słów kluczowych

if (*peek* holds a letter) {
collect letters or digits into a buffer *b*;
s = string formed from the characters in *b*;
w = token returned by *words.get(s)*;
if (*w* is not **null) return** *w*;
else {
Enter the key-value pair (*s*, ‹ **id**, *s*›) into *words*
return token ‹ **id**, *s* ›; } }

Analiza Leksykalna

- **Pseudokod funkcji *scan*, która zwraca tokeny**

```
Token scan() {
    skip white space;
    handle numbers;
    handle reserved words and identifiers;
    /* if we get here, treat read-ahead character
       peek as a token */
   Token t = new Token(peek);
    peek = blank
    return t;
```

Analizator leksykalny w C

```
int lineno = 1;
int tokenval = NONE;

int lexan ()
{
        int t;
        while (1) {
                t = getchar ();
                if (t == ' ' || t == '\t');
                else if (t == '\n')
                        lineno++;
                else if (isdigit (t))
                {
                        tokenval = t – '0';
                        t = getchar();
                        while (isdigit(t))
                        {
                                tokenval = tokenval*10 + t – '0';
                                t = getchar();
                        }
                        ungetc (t, stdin);
                        return NUM;
                }
                else
                {
                        tokenval = NONE;
                        return t;
                }
        }
}
```

Analizator leksykalny w C

```
int lineno = 1;
int tokenval = NONE;

int lexan ()
{
        int t;
        while (1) {
                t = getchar ();
                if (t == ' ' || t == '\t');
                else if (t == '\n')
                        lineno++;
                else if (isdigit (t))
                {
                        tokenval = t – '0';
                        t = getchar();
                        while (isdigit(t))
                        {
                                tokenval = tokenval*10 + t – '0';
                                t = getchar();
                        }
                        ungetc (t, stdin);
                        return NUM;
                }
                else
                {
                        tokenval = NONE;
                        return t;
                }
        }
}
```

Numer aktualnie analizowanego wiersza

Analizator leksykalny w C

```
int lineno = 1;
int tokenval = NONE;

int lexan ()
{
        int t;
        while (1) {
                t = getchar ();
                if (t == ' ' || t == '\t');
                else if (t == '\n')
                        lineno++;
                else if (isdigit (t))
                {
                        tokenval = t – '0';
                        t = getchar();
                        while (isdigit(t))
                        {
                                tokenval = tokenval*10 + t – '0';
                                t = getchar();
                        }
                        ungetc (t, stdin);
                        return NUM;
                }
                else
                {
                        tokenval = NONE;
                        return t;
                }
        }
}
```

Atrybut symbolu leksykalnego

Analizator leksykalny w C

```
int lineno = 1;
int tokenval = NONE;

int lexan ()
{
        int t;
        while (1) {
                t = getchar ();
                if (t == ' ' || t == '\t');
                else if (t == '\n')
                        lineno++;
                else if (isdigit (t))
                {
                        tokenval = t – '0';
                        t = getchar();
                        while (isdigit(t))
                        {
                                tokenval = tokenval*10 + t – '0';
                                t = getchar();
                        }
                        ungetc (t, stdin);
                        return NUM;
                }
                else
                {
                        tokenval = NONE;
                        return t;
                }
        }
}
```

Pobierz jeden znak

Analizator leksykalny w C

```
int lineno = 1;
int tokenval = NONE;

int lexan ()
{
        int t;
        while (1) {
                t = getchar ();
                if (t == ' ' || t == '\t');
                else if (t == '\n')
                        lineno++;
                else if (isdigit (t))
                {
                        tokenval = t – '0';
                        t = getchar();
                        while (isdigit(t))
                        {
                                tokenval = tokenval*10 + t – '0';
                                t = getchar();
                        }
                        ungetc (t, stdin);
                        return NUM;
                }
                else
                {
                        tokenval = NONE;
                        return t;
                }
        }
}
```

Jeżeli t jest białym znakiem, pomiń znak

Analizator leksykalny w C

```
int lineno = 1;
int tokenval = NONE;

int lexan ()
{
        int t;
        while (1) {
                t = getchar ();
                if (t == ' ' || t == '\t');
                else if (t == '\n')
                        lineno++;
                else if (isdigit (t))
                {
                        tokenval = t – '0';
                        t = getchar();
                        while (isdigit(t))
                        {
                                tokenval = tokenval*10 + t – '0';
                                t = getchar();
                        }
                        ungetc (t, stdin);
                        return NUM;
                }
                else
                {
                        tokenval = NONE;
                        return t;
                }
        }
}
```

Jeżeli t jest znakiem nowej linii, zwiększ numer aktualnie analizowanego wiersza

Analizator leksykalny w C

```
int lineno = 1;
int tokenval = NONE;

int lexan ()
{
        int t;
        while (1) {
                t = getchar ();
                if (t == ' ' || t == '\t');
                else if (t == '\n')
                        lineno++;
                else if (isdigit (t))
                {
                        tokenval = t – '0';
                        t = getchar();
                        while (isdigit(t))
                        {
                                tokenval = tokenval*10 + t – '0';
                                t = getchar();
                        }
                        ungetc (t, stdin);
                        return NUM;
                }
                else
                {
                        tokenval = NONE;
                        return t;
                }
        }
}
```

Jeżeli t jest liczbą, zamień napis na wartość typu całkowitego

Analizator leksykalny w C

```
int lineno = 1;
int tokenval = NONE;

int lexan ()
{
        int t;
        while (1) {
                t = getchar ();
                if (t == ' ' || t == '\t');
                else if (t == '\n')
                        lineno++;
                else if (isdigit (t))
                {
                        tokenval = t – '0';
                        t = getchar();
                        while (isdigit(t))
                        {
                                tokenval = tokenval*10 + t – '0';
                                t = getchar();
                        }
                        ungetc (t, stdin);
                        return NUM;
                }
                else
                {
                        tokenval = NONE;
                        return t;
                }
        }
}
```

Wczytany znak jest nieznanym symbolem

Tablica symboli

Tablica symboli jest wykorzystywana do przechowywania zmiennych oraz słów kluczowych.

Jest inicjalizowana poprzez wstawienie do niej słów kluczowych:

Tablica symboli

```
int insert(const char* s, int t); /* zwraca indeks w tablicy symboli dla
nowego leksemu s i tokenu t */
int lookup(const char* s); /* zwraca indeks wpisu dla leksemu s lub 0
gdy nie znaleziono */
```

```
insert("div", DIV);
insert("mod", MOD);
```

Analizator leksykalny z tablicą symboli

```
int lexan () {
        int t;
        while (1) {
                t = getchar ();
                if (t == ' ' || t == '\t');
                else if (t == '\n')
                        lineno++;
                else if (isdigit (t)) {
                        ungetc (t, stdin);
                        scanf ("%d", &tokenval);
                        return NUM;
                } else if (isalpha (t)) {
                        int p, b = 0;
                        while (isalnum (t)) {
                                lexbuf[b] = t;
                                t = getchar ();
                                b++;
                                if (b >= BSIZE) error ("compiler error");
                        }
                        lexbuf[b] = EOS;
                        if (t != EOF) ungetc (t, stdin);
                        p = lookup (lexbuf);
                        if (p == 0) p = insert (lexbuf, ID);
                        tokenval = p;
                        return symtable[p].token;
                } else if (t == EOF) return DONE;
                else {
                        tokenval = NONE;
                        return t;
                }
        }
}
```

Analizator leksykalny z tablicą symboli

```
int lexan () {
        int t;
        while (1) {
                t = getchar ();
                if (t == ' ' || t == '\t');
                else if (t == '\n')
                        lineno++;
                else if (isdigit (t)) {
                        ungetc (t, stdin);
                        scanf ("%d", &tokenval);
                        return NUM;
                } else if (isalpha (t)) {
                        int p, b = 0;
                        while (isalnum (t)) {
                                lexbuf[b] = t;
                                t = getchar ();
                                b++;
                                if (b >= BSIZE) error ("compiler error");
                        }
                        lexbuf[b] = EOS;
                        if (t != EOF) ungetc (t, stdin);
                        p = lookup (lexbuf);
                        if (p == 0) p = insert (lexbuf, ID);
                        tokenval = p;
                        return symtable[p].token;
                } else if (t == EOF) return DONE;
                else {
                        tokenval = NONE;
                        return t;
                }
        }
}
```

Jeżeli t jest identyfikatorem

Analizator leksykalny z tablicą symboli

```
int lexan () {
        int t;
        while (1) {
                t = getchar ();
                if (t == ' ' || t == '\t');
                else if (t == '\n')
                        lineno++;
                else if (isdigit (t)) {
                        ungetc (t, stdin);
                        scanf ("%d", &tokenval);
                        return NUM;
                } else if (isalpha (t)) {
                        int p, b = 0;
                        while (isalnum (t)) {
                                lexbuf[b] = t;
                                t = getchar ();
                                b++;
                                if (b >= BSIZE) error ("compiler error");
                        }
                        lexbuf[b] = EOS;
                        if (t != EOF) ungetc (t, stdin);
                        p = lookup (lexbuf);
                        if (p == 0) p = insert (lexbuf, ID);
                        tokenval = p;
                        return symtable[p].token;
                } else if (t == EOF) return DONE;
                else {
                        tokenval = NONE;
                        return t;
                }
        }
}
```

Jeżeli t jest znakiem końca pliku

ANALIZA LEKSYKALNA

Ciągi znaków i języki (Strings and Languages)

- Alfabet jest to dowolny skończony zbiór symboli.

Typowe przykłady symboli to są litery, cyfry i znaki interpunkcyjne.

Zbiór {0,1} jest alfabetem binarnym.

Znaki tablicy kodów ASCII tworzą ważny przykład alfabetu.

ANALIZA LEKSYKALNA

Ciągi znaków i języki

- Napis (ciąg znaków, łańcuch) nad pewnym alfabetem jest to skończony ciąg symboli z tego alfabetu.

- Długość napisu s, zapisywana jako | s |, jest to liczba wystąpień symboli w napisie s.

ANALIZA LEKSYKALNA

Ciągi znaków i języki

- Na przykład, napis banan ma długość 5.
- Pusty ciąg znaków, oznaczany jako ε, jest ciągiem o zerowej długości.

Bardzo szeroka definicja języka:

Język jest to zbiór napisów nad pewnym ustalonym alfabetem.

ANALIZA LEKSYKALNA

Ciągi znaków i języki

- Przykład:

  dla alfabetu L = {A, . . . , Z},

  zbiór {A, B, C, BF,…,ABZ,…}

  jest językiem zdefiniowanym przez alfabet L.

- Należy zauważyć, że definicja "języka" nie wymaga, aby każdy napis posiadał jakiś sens.

ANALIZA LEKSYKALNA

Pojęcia związane z napisami

Prefiks(przedrostek) napisu s jest to dowolny napis uzyskany przez usunięcie zero lub więcej symboli z końca s.

Na przykład, ban, banana, and ε są prefiksami napisu banana.

ANALIZA LEKSYKALNA

Pojęcia związane z napisami

- Sufiks(przyrostek) napisu s jest to dowolny napis uzyskany przez usunięcie zero lub więcej symboli z początku s.

Na przykład, nana, banana, and ε są sufiksami napisu banana.

ANALIZA LEKSYKALNA

Pojęcia związane z napisami

- Podnapis(podciąg spójny) napisu s jest uzyskiwany przez usuwanie jakichkolwiek przedrostków i przyrostków z s.

Na przykład, banana, nan i ε są podnapisami napisu banana.

ANALIZA LEKSYKALNA

Pojęcia związane z napisami

- Przedrostki, przyrostki i podnapisy napisu *s* są właściwe jeśli nie są napisami pustymi lub nie są równe *s*.

ANALIZA LEKSYKALNA

Pojęcia związane z napisami

- Podciąg napisu *s* jest to dowolny ciąg utworzony przez usunięcie zero lub więcej nie koniecznie kolejnych znaków z *s*.

- Na przykład, baan jest podciągiem napisu banana.

ANALIZA LEKSYKALNA

Pojęcia związane z napisami

- Złączenie (concatenation) napisów x i y jest to napis xy utworzony przez dodanie y na koniec do x.

  Na przykład jeśli x = dog i y = house, to

  xy = doghouse.

- Dla napisu pustego są sprawiedliwe równości:

$$\varepsilon s = s\varepsilon = s.$$

ANALIZA LEKSYKALNA

Pojęcia związane z napisami

- Podnoszenie napisu do potęgi definiujemy w następujący sposób:

$$s^0 = \varepsilon$$

$$s^i = s^{i-1}s \quad \text{dla} \quad i > 0$$

- Przykłady:

$$s^1 = s, \quad s^2 = ss, \quad s^3 = sss.$$

ANALIZA LEKSYKALNA

Operacje na językach

- Suma(Union)

$$L \cup M = \{s \mid s \in L \text{ or } s \in M\}$$

- Złączenie (Concatenation)

$$LM = \{xy \mid x \in L \text{ and } y \in M\}$$

- Potęgowanie (Exponentiation)

$$L^0 = \{\varepsilon\}; \quad L^i = L^{i-1}L$$

ANALIZA LEKSYKALNA

Operacje na językach

- Domknięcie Kleen'a (Kleene closure)

  $L^* = \cup_{i=0,\ldots,\infty} L^i$

- Domknięcie dodatnie (Positive closure)

  $L^+ = \cup_{i=1,\ldots,\infty} L^i$

ANALIZA LEKSYKALNA

Wyrażenia regularne (Regular Expressions)

- Podstawowe wyrażenia regularne:

  ε jest wyrażeniem regularnym oznaczającym język $\{\varepsilon\}$

  $a \in \Sigma$ jest wyrażeniem regularnym oznaczającym język $\{a\}$

ANALIZA LEKSYKALNA

Wyrażenia regularne

- Jeśli r i s są wyrażeniami regularnymi oznaczającymi języki $L(r)$ i $M(s)$ to
  - $r \mid s$ jest wyrażeniem regularnym oznaczającym język $L(r) \cup M(s)$
  - rs jest wyrażeniem regularnym oznaczającym język $L(r)M(s)$

ANALIZA LEKSYKALNA

Wyrażenia regularne

r^* jest wyrażeniem regularnym oznaczającym język $L(r)^*$ ·

(r) jest wyrażeniem regularnym oznaczającym język $L(r)$.

ANALIZA LEKSYKALNA

Wyrażenia regularne

- Czyli wyrażenia regularne są budowane z mniejszych wyrażeń regularnych w sposób rekurencyjny.

ANALIZA LEKSYKALNA

Wyrażenia regularne

- Wyrażenia regularne mogą zawierać zbędne pary nawiasów.
- Możemy usunąć niektóre pary nawiasów, jeśli przyjmiemy następującą konwencję:

ANALIZA LEKSYKALNA

Wyrażenia regularne

- a) Operator jednoargumentowy * (operator domknięcia) ma najwyższy priorytet i jest łączny lewostronnie,

- b) Złączenie ma drugi najwyższy priorytet i jest łączne lewostronnie.

ANALIZA LEKSYKALNA

Wyrażenia regularne

- **Przykład** : $\Sigma = \{a, b\}$.

  1. Wyrażenie regularne **a|b** oznacza język $\{a, b\}$.
  2. (**a|b**) (**a|b**) oznacza {*aa*, *ab*, *ba*, *bb*), czyli język, którego elementami są napisy o długości 2.

ANALIZA LEKSYKALNA

Wyrażenia regularne

- 3. **a*** oznacza język, którego elementy są złożeniem zero lub więcej symboli *a*:

  $\{\varepsilon, a, aa, aaa, \ldots\}$.

ANALIZA LEKSYKALNA

Wyrażenia regularne

- 4. (**a|b**)* oznacza język, którego elementy są złożeniem zero lub więcej symboli *a* lub *b* :

  {*ε, a, b, aa, ab, ba, bb, aaa, . . .*}.

  5. **a|a*b** oznacza język

  {*a, b, ab, aab,aaab,. . .*}

ANALIZA LEKSYKALNA

Wyrażenia regularne

- 4. (**a|b**)* oznacza język, którego elementy są złożeniem zero lub więcej symboli *a* lub *b* :

  {*ε, a, b, aa, ab, ba, bb, aaa, . . .*}.

  5. **a|a*b** oznacza język

  {*a, b, ab, aab,aaab,. . .*}

Przykład

Rozważmy alfabet $\sum = \{ a \}$

Jakie jest wyrażenie regularne, które oznacza język, którego każde słowo jest o długości nieparzystej

???

Przykład

Rozważmy alfabet $\sum = \{ a \}$

a (aa)*

jest wyrażeniem regularnym, które oznacza język, którego każde słowo jest o długości nieparzystej

Przykład

Rozważmy alfabet $\sum = \{ a, b \}$

Jakie jest wyrażenie regularne, które oznacza język, którego każde słowo zaczyna się od litery b

???

Przykład

Rozważmy alfabet Σ = { a, b }

b (a | b)*

jest wyrażeniem regularnym, które oznacza język, którego każde słowo zaczyna się od litery b

Przykład

Rozważmy alfabet $\Sigma = \{ a, b \}$

Jakie jest wyrażenie regularne, które oznacza język, którego każde słowo musi zaczynać się na literę a, a kończy się na literę b

????

Przykład

Rozważmy alfabet $\Sigma = \{ a, b \}$

b (a | b)* a jest wyrażeniem regularnym, które oznacza język, którego każde słowo musi zaczynać się na literę b, a kończy się na literę a

Przykład

Rozważmy alfabet $\sum$ = { a, b, c }

Jakie jest wyrażenie regularne, które oznacza język:

L = { a, c, ab, cb, abb, cbb, abbb,
cbbb, abbbb, cbbbb . . . } ?

Przykład

Rozważmy alfabet $\sum$ = { a, b, c }

((a | c) b*) oznacza język:

L = { a, c, ab, cb, abb, cbb, abbb,
cbbb, abbbb, cbbbb . . . }

Dziękuję za uwagę