Zajęcia P1. Analizator leksykalny dla uproszczonego języka C

1 Cel ćwiczeń

Celem ćwiczeń jest stworzenie prostego analizatora leksykalnego dla bardzo uproszczonej wersji języka programowania C. Zadaniem tworzonego analizatora jest:

- rozpoznawanie elementów języka C i określanie ich wartości
- usuwanie białych znaków i komentarzy
- wykrywanie wskazanych dyrektyw
- wykrywanie błędów leksykalnych

2 Czynności wstępne

Po włączeniu komputera należy wybrać system operacyjny Linux i zalogować się jako użytkownik `student`. Należy otworzyć okno konsoli (np. nacisnąć `Alt-F2` i napisać `xterm`), utworzyć własny podkatalog za pomocą polecenia `mkdir` *nazwisko użytkownika*, i podkatalog dla bieżącego ćwiczenia. Ze strony przedmiotu na platformie Moodle dla tematu *Analiza leksykalna* należy pobrać pliki dla języka C. Znajdują się tam następujące pliki:

- `p1c.pdf` — instrukcja (właśnie czytany plik)
- `Makefile` — potrzebny do kompilacji za pomocą polecenia `make`
- `common.h` — plik nagłówkowy zawierający określenie największej długości napisów
- `c.l` — szkielet analizatora leksykalnego, który należy uzupełnić; należy zwrócić uwagę na definicję funkcji `process_token()`, którą należy wykorzystać
- `c.y` — analizator składniowy, którego jedynymi zadaniami jest deklaracja rozpoznawanych elementów końcowych oraz wywołanie analizatora leksykalnego
- `test1.c` — program testowy poprawny w danej gramatyce
- `test2.c` — program testowy z błędami, które powinny zostać wykryte

Po zakończeniu pracy wskazane jest usunięcie utworzonego katalogu wraz z zawartością.

3 Zadania do wykonania

Należy uzupełnić dostarczony szkielet analizatora leksykalnego i pokazać, że działa poprawnie testując go na dostarczonych programach testowych. Analizator powinien wypisywać informacje o rozpoznanych symbolach końcowych w trzech kolumnach:

1. dopasowany tekst
2. rozpoznany symbol
3. wartość symbolu (tylko w sytuacji, gdy symbol rzeczywiście ma wartość)

Do wypisywania tych informacji służy zawarta w dostarczonym szkielecie analizatora funkcja `process_token()`. Funkcja zwraca rozpoznany symbol, dlatego w działaniu dla reguły rozpoznającej symbol powinna znaleźć się instrukcja `return process_token(. . .)` z odpowiednimi parametrami funkcji.

Dostarczony kod należy uzupełnić o następujące elementy **w podanej poniżej kolejności**:

A. wypisanie własnego imienia i nazwiska (w pliku dla programu `bison`)

B. wykrywanie słów kluczowych zdefiniowanych w pliku źródłowym dla programu `bison` (występujących w większości w programach testowych)

C. usuwanie białych znaków

D. usuwanie komentarzy jednowierszowych bez użycia warunków początkowych

E. wykrywanie operatorów wieloznakowych (`<=`, `++`,...) występujących w programach testowych

F. wykrywanie identyfikatorów

G. wykrywanie liczba całkowitych i zmiennoprzecinkowych

H. wykrywanie stałych tekstowych (napisów) bez użycia mechanizmu warunków początkowych

I. wykrywanie stałych znakowych

J. wykrywanie symboli końcowych jednoznakowych: operatorów, interpunkcji

K. wykrywanie dyrektywy dołączania plików

L. wykrywanie napisów z użyciem warunków początkowych

M. usuwanie komentarzy wielowierszowych z użyciem warunków początkowych

N. znajdowanie znaków zamknięcia komentarza przy braku jego rozpoczęcia z użyciem warunków początkowych

O. wykrywanie niezamkniętego komentarza ze wskazaniem wiersza jego rozpoczęcia z użyciem warunków początkowych

4 Ocena

Za każdy element można dostać 1 punkt, czyli razem 15 punktów. Jeżeli ktoś nie zdąży zrobić wszystkich elementów na zajęciach, istnieje możliwość dokończenia analizatora w domu, ale **tylko elementów od K do O** i za każdy element wykonany **w domu przysługuje tylko połowa punktów**. Plik wykonany na zajęciach umieścić **na zajęciach** na platformie Moodle. UWAGA: **Analizator leksykalny potrzebny będzie na następnych zajęciach**. Należy pamiętać, by elementy rozpoznawane w celu przekazania do analizatora składniowego zostały do niego przekazane za pomocą instrukcji `return`. Elementy usuwane, np. komentarze czy białe znaki, nie powinny wywoływać instrukcji `return`.

5 Przypomnienie warunków początkowych

- Warunek początkowy aktywny na początku programu: `INITIAL`
- Deklaracja (w pierwszej części): `%x warunek1, warunek2,. . .`
- Dopasowanie w określonym warunku początkowym:
```
<war1> re1      działanie1;
<war1,war2,INITIAL>re2 działanie2;
<*>re3        działanie3
```
- zmiana warunku początkowego: `BEGIN warunek4`
- bieżący warunek początkowy: `YYSTATE`
- sprawdzenie bieżącego warunku po odczytaniu wszystkich danych wejściowych: w funkcji `yywrap`, którą należy zdefiniować i która musi zwrócić wartość 1

6 Dane testowe — plik test1.c

```
1 /*************************************************/
2 /* Program ASCII - wyswietla rozszerzone kody ASCII */
3 /*************************************************/
4 #include <stdio.h>
5 #include "test.h"
6 unsigned char uc; // zmienna sterujaca petli typu char
7 int fromASCII = 128, toASCII = 255;
8 long int x[10];
9 void main( void )
```

```
{
  struct data {
    int rok;
    int miesiac;
    int dzien;
  };
  data poczatek, koniec;
  int i;
  printf( "Rozszerzone kody ASCII\n\n");
  for ( uc = fromASCII; uc <= toASCII; uc1++ ) {
    printf( "%3d:%2c", uc, uc ); printf("\n");
  }
  int x1 = fromASCII + 2 * ( 20 +  toASCII );  /* int */
  double realTest = 12.34e-12 + .56 + 78.; /* double */
  x[0] = 1;
  for (i = 1; i < 10; i++) {
    x[i] = x[i-1] * i * i;
  }
  poczatek.rok = 2018;
  poczatek.miesiac = 10;
  poczatek.dzien = 1;
}
```

7 Dane testowe — plik test2.c

```
/*************************************************/
/* Program ASCII - wyswietla rozszerzone kody ASCII */
/*************************************************/
#include <stdio.h>
#include "test.h"
unsigned char uc; // zmienna sterujaca petli
int fromASCII = 128, toASCII = 255;
void main( void )
{
        printf("\n\n\nRozszerzone kody ASCII\n\n");
        for (uc = fromASCII; uc <= toASCII; uc1++)
        {
                printf("%3d:%2c", uc, uc);
        }
}
int x1 = fromASCII + 2 * ( 20 +  toASCII ); /* te linie /* sluza
* / wylacznie celom testowym ;-) */
double realTest = 12.34 + .56 + 78.;
*/ // nieotwarty komentarz
"Niezamknieta stala tekstowa
/* niezamkniety komentarz
```

8 Wyjście analizatora leksykalnego dla test1.c

```
Autor: Imię i Nazwisko
yytext              Typ symbolu       Wartość symbolu znakowo

Przetwarzanie dyrektywy #include <stdio.h>
Przetwarzanie dyrektywy #include "test.h"
unsigned            KW_UNSIGNED
char                KW_CHAR
uc                  IDENT             uc
;                   ;
int                 KW_INT
fromASCII           IDENT             fromASCII
=                   =
```

```
128                     INTEGER_CONST   128
,                       ,
toASCII                 IDENT           toASCII
=                       =
255                     INTEGER_CONST   255
;                       ;
long                    KW_LONG
int                     KW_INT
x                       IDENT           x
[                       [
10                      INTEGER_CONST   10
]                       ]
;                       ;
void                    KW_VOID
main                    IDENT           main
(                       (
void                    KW_VOID
)                       )
{                       {
struct                  IDENT           struct
data                    IDENT           data
{                       {
int                     KW_INT
rok                     IDENT           rok
;                       ;
int                     KW_INT
miesiac                 IDENT           miesiac
;                       ;
int                     KW_INT
dzien                   IDENT           dzien
;                       ;
}                       }
;                       ;
data                    IDENT           data
poczatek                IDENT           poczatek
,                       ,
koniec                  IDENT           koniec
;                       ;
int                     KW_INT
i                       IDENT           i
;                       ;
printf                  IDENT           printf
(                       (
"Rozszerzone kody AS    STRING_CONST    "Rozszerzone kody ASCII\n\n"
)                       )
;                       ;
for                     KW_FOR
(                       (
uc                      IDENT           uc
=                       =
fromASCII               IDENT           fromASCII
;                       ;
uc                      IDENT           uc
<=                      LE
toASCII                 IDENT           toASCII
;                       ;
uc1                     IDENT           uc1
++                      INC
)                       )
{                       {
printf                  IDENT           printf
(                       (
"%3d:%2c"               STRING_CONST    "%3d:%2c"
,                       ,
uc                      IDENT           uc
,                       ,
```

```
uc                  IDENT           uc
)                   )
;                   ;
printf              IDENT           printf
(                   (
"\n"                STRING_CONST    "\n"
)                   )
;                   ;
}                   }
int                 KW_INT
x1                  IDENT           x1
=                   =
fromASCII           IDENT           fromASCII
+                   +
2                   INTEGER_CONST   2
*                   *
(                   (
20                  INTEGER_CONST   20
+                   +
toASCII             IDENT           toASCII
)                   )
;                   ;
double              KW_DOUBLE
realTest            IDENT           realTest
=                   =
12.34e-12           FLOAT_CONST     12.34e-12
+                   +
.56                 FLOAT_CONST     .56
+                   +
78.                 FLOAT_CONST     78.
;                   ;
x                   IDENT           x
[                   [
0                   INTEGER_CONST   0
]                   ]
=                   =
1                   INTEGER_CONST   1
;                   ;
for                 KW_FOR
(                   (
i                   IDENT           i
=                   =
1                   INTEGER_CONST   1
;                   ;
i                   IDENT           i
<                   <
10                  INTEGER_CONST   10
;                   ;
i                   IDENT           i
++                  INC
)                   )
{                   {
x                   IDENT           x
[                   [
i                   IDENT           i
]                   ]
=                   =
x                   IDENT           x
[                   [
i                   IDENT           i
-                   -
1                   INTEGER_CONST   1
]                   ]
*                   *
i                   IDENT           i
*                   *
```

```
i                   IDENT           i
;                   ;
}                   }
poczatek            IDENT           poczatek
.                   .
rok                 IDENT           rok
=                   =
2018                INTEGER_CONST   2018
;                   ;
poczatek            IDENT           poczatek
.                   .
miesiac             IDENT           miesiac
=                   =
10                  INTEGER_CONST   10
;                   ;
poczatek            IDENT           poczatek
.                   .
dzien               IDENT           dzien
=                   =
1                   INTEGER_CONST   1
;                   ;
}                   }
```

9 Wyjście analizatora leksykalnego dla test2.c

```
Autor: Imię i Nazwisko
yytext              Typ symbolu     Wartość symbolu znakowo

Przetwarzanie dyrektywy #include <stdio.h>
Przetwarzanie dyrektywy #include "test.h"
unsigned            KW_UNSIGNED
char                KW_CHAR
uc                  IDENT           uc
;                   ;
int                 KW_INT
fromASCII           IDENT           fromASCII
=                   =
128                 INTEGER_CONST   128
,                   ,
toASCII             IDENT           toASCII
=                   =
255                 INTEGER_CONST   255
;                   ;
void                KW_VOID
main                IDENT           main
(                   (
void                KW_VOID
)                   )
{                   {
printf              IDENT           printf
(                   (
"\n\n\nRozszerzone kSTRING_CONST     "\n\n\nRozszerzone kody ASCII\n\n"
)                   )
;                   ;
for                 KW_FOR
(                   (
uc                  IDENT           uc
=                   =
fromASCII           IDENT           fromASCII
;                   ;
uc                  IDENT           uc
<=                  LE
toASCII             IDENT           toASCII
;                   ;
```

```
uc1                 IDENT           uc1
++                  INC
)                   )
{                   {
printf              IDENT           printf
(                   (
"%3d:%2c"           STRING_CONST    "%3d:%2c"
,                   ,
uc                  IDENT           uc
,                   ,
uc                  IDENT           uc
)                   )
;                   ;
}                   }
}                   }
int                 KW_INT
x1                  IDENT           x1
=                   =
fromASCII           IDENT           fromASCII
+                   +
2                   INTEGER_CONST   2
*                   *
(                   (
20                  INTEGER_CONST   20
+                   +
toASCII             IDENT           toASCII
)                   )
;                   ;
double              KW_DOUBLE
realTest            IDENT           realTest
=                   =
12.34               FLOAT_CONST     12.34
+                   +
.56                 FLOAT_CONST     .56
+                   +
78.                 FLOAT_CONST     78.
;                   ;
Nieoczekiwane zamknięcie komentarza w wierszu 19
Niezamknięty napis otwarty w wierszu 20
Brak zamknięcia komentarza otwartego w wierszu 21
```