Wybrane narzędzia do tworzenia analizatorów leksykalnych i składniowych w C/C++

Narzędzia, zastosowanie oraz próby rankingu.

Flex

Flex wywodzi się jak cała rodzina tego typu narzędzi od jednego przodka: Lexa.

Polska strona o Flex : http://man.przez.net/flex.1.html

Struktura lexa:

definicje pomocnicze

%%

reguły

%%

podprogramy pomocnicze

Flex

Przykładowy kod Flexa – skaner rozpoznaje komentarze podczas zliczania linii:

```
%x comment
%%
        int line_num = 1;
„/*„    BEGIN(comment);
<comment>[^*\n]*            /* zjedz wszystko, co nie jest '*' */
<comment>"*"+[^*/\n]*       /* zjedz '*'-ki, po których nie ma '/' */
<comment>\n                 ++line_num;
<comment>"*"+"/„            BEGIN(INITIAL);
```

re2c

Bardzo szybkie narzędzie tworzące skanery 2-3 razy szybsze niż ich odpowiedniki stworzone przez Flex’a. Posiada niestety bardzo ubogą dokumentację co stanowi utrudnienie.

Przykładowy kod re2c:

```
char *scan(char *p){
char *q;
#define YYCTYPE         char
#define YYCURSOR        p
#define YYLIMIT         p
#define YYMARKER        q
#define YYFILL(n)
/*!re2c
[0-9]+                  {return YYCURSOR;}
[\000-\377] {return NULL;}
*/
}
```

LLOOP

LLOOP jest generatorem parserów typu LL(1) stosuje notację BNF

- Umożliwia definiowanie dyrektyw preprocesora np. podmienienie wyrazów przed rozpoczęciem przetwarzania
- Umożliwia wykonywanie poleceń po zakończonym parsingu. Przykładem może być sprawdzenie struktury języka, która może być wykonywana po parsingu danych lub automatycznej korekcji błędów.
- Każdy obiekt sparsowany może być zamieniony do swojej oryginalnej formy jaką miał w strumieniu wejściowym.
- Możliwość stworzenia plików konfiguracyjnych, które ustawiają ścieżki plików wyjściowych.cpp i .obj , plików testowych
- Każdy nieterminal, każdy token, każda dyrektywa preprocesora jest mapowana do klasy. Oddzielne klasy są generowane do dalszego wykorzystywania przez programistę.

LLOOP

Przykładowa klasa zawierająca informacje o tokenie:

```
class SampleToken{
protected:
                unsigned char m_cWsFlag [ 1 ] [ 20 ] ;
                unsigned short m_uNbWs [ 1 ] ;
                SampleTokenFactory * ms_factory;

public:
                friend class SampleTokenFactory;
                static bool is_a ( Symbol * pSymbol )
                static SampleTokenFactory & factory ( )
                static void setFactory ( SampleTokenFactory * pNewFactory )
    SampleToken ( )
    virtual ~SampleToken ( )
    virtual const char * getName ( ) const
    virtual bool parseSymbol ( Parser & parser , Symbol * & rpReturnSymbol , Symbol * pParent = NULL )
    bool parseSymbol ( Parser & parser , SampleToken * & rpReturnSymbol , Symbol * pParent = NULL )
    virtual bool expand ( std::ostream & os )
    virtual bool backtrace ( std::ostream & os )
    virtual bool visit ( Visitor & visitor )
    virtual bool isToken ( ) const
    virtual bool isNonTerminal ( ) const
    virtual void reset ( )
}
```

LLOOP

przykładowy kod zapisany w pliku wejsciowym:

```
import int;
Shape::= 'Point' '(' 'x' '=' int ',' 'y' '=' int [',' Color] ')' [Shape]
| 'Hole' [ Shape ]
| {{ }}
Color::= 'blue'
| 'red'
| 'white'
{{
}}
```

Stroną projektu jest :http://www.ersa-france.com/lloop/

LRgen

LRgen, stworzony przez Parsetec, umożliwia budowę lekserów, parserów, translatorów i kompilatorów dla języków programowania.

Bazuje on na notacji TBNF lecz umożliwia pisanie w notacji EBNF. Kod wynikowy jest kreowany w C++ lub ANSI C.

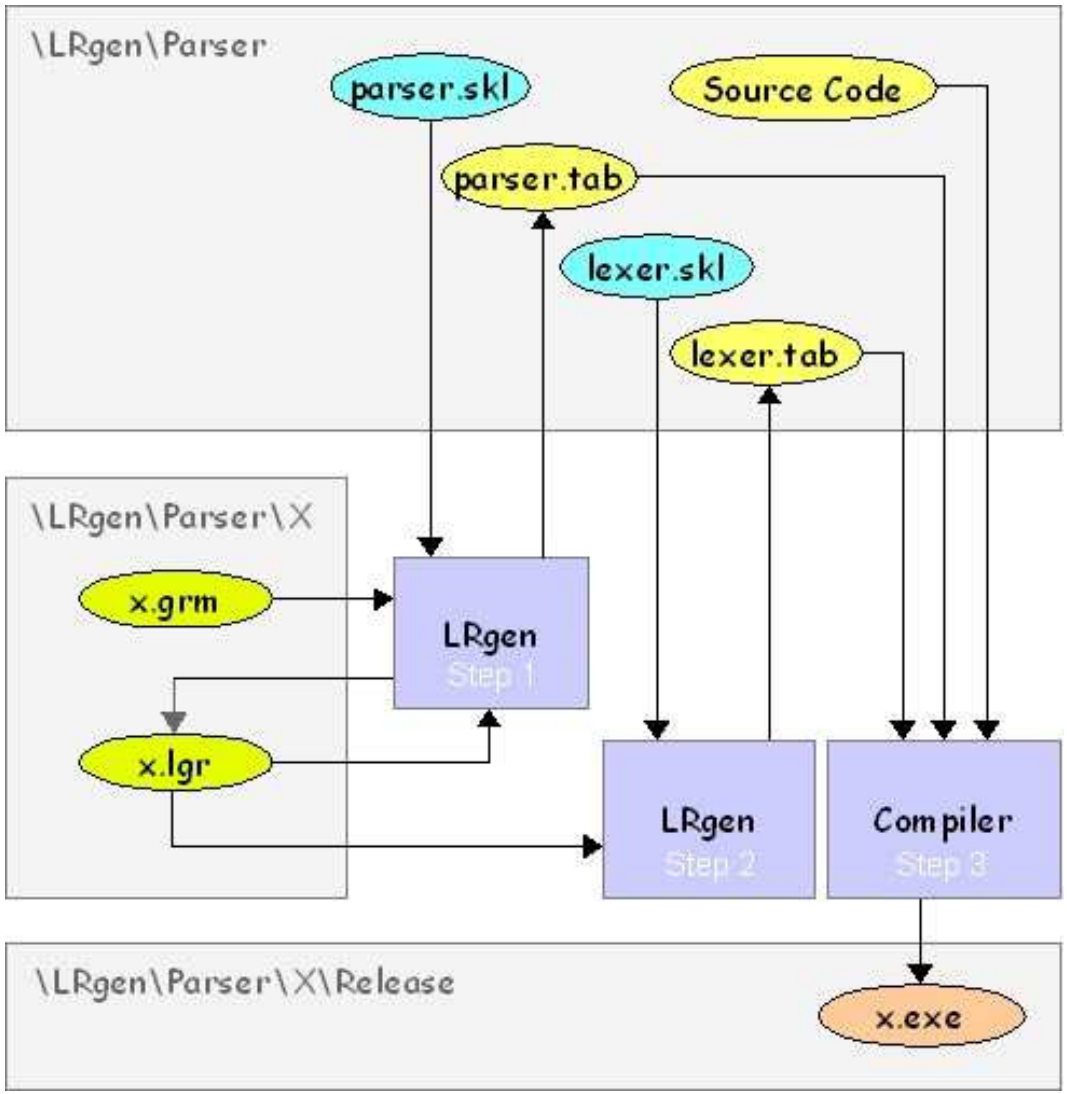
Plusem zastosowania składni TBNF jest potrzeba pisania mniejszej ilości kodu.

Program jest programem komercyjnym.
W swej najprostszej, bezpłatnej wersji korzysta z parsera SLR(1)

W płatnej posiada już parser LALR(1) oraz udostępnia więcej funkcji

LRgen

Diagram przebiegu tworzenia gramatyki:



LRgen

Przykładowy parser prostych wyrażeń arytmetycznych:

```
/* Input Tokens. */
<error> => error()
<integer> => lookup()
/* Operator precedence. */
[ '+' '-' ] <<
[ '*' '/' ] <<
/* Rules. */
Goal -> Exp
Exp -> Primary
-> Exp '+' Exp +> add ..* ("ADD\n")
-> Exp '-' Exp +> sub ..* ("SUB\n")
-> Exp '*' Exp +> mul ..* ("MUL\n")
-> Exp '/' Exp +> div ..* ("DIV\n")
Primary ->  <integer> +> int $1 ..* ("LOAD %s\n")
```

Strona firmy Parsetec : http://www.parsetec.com/lrgen/index.html

TextTransformer

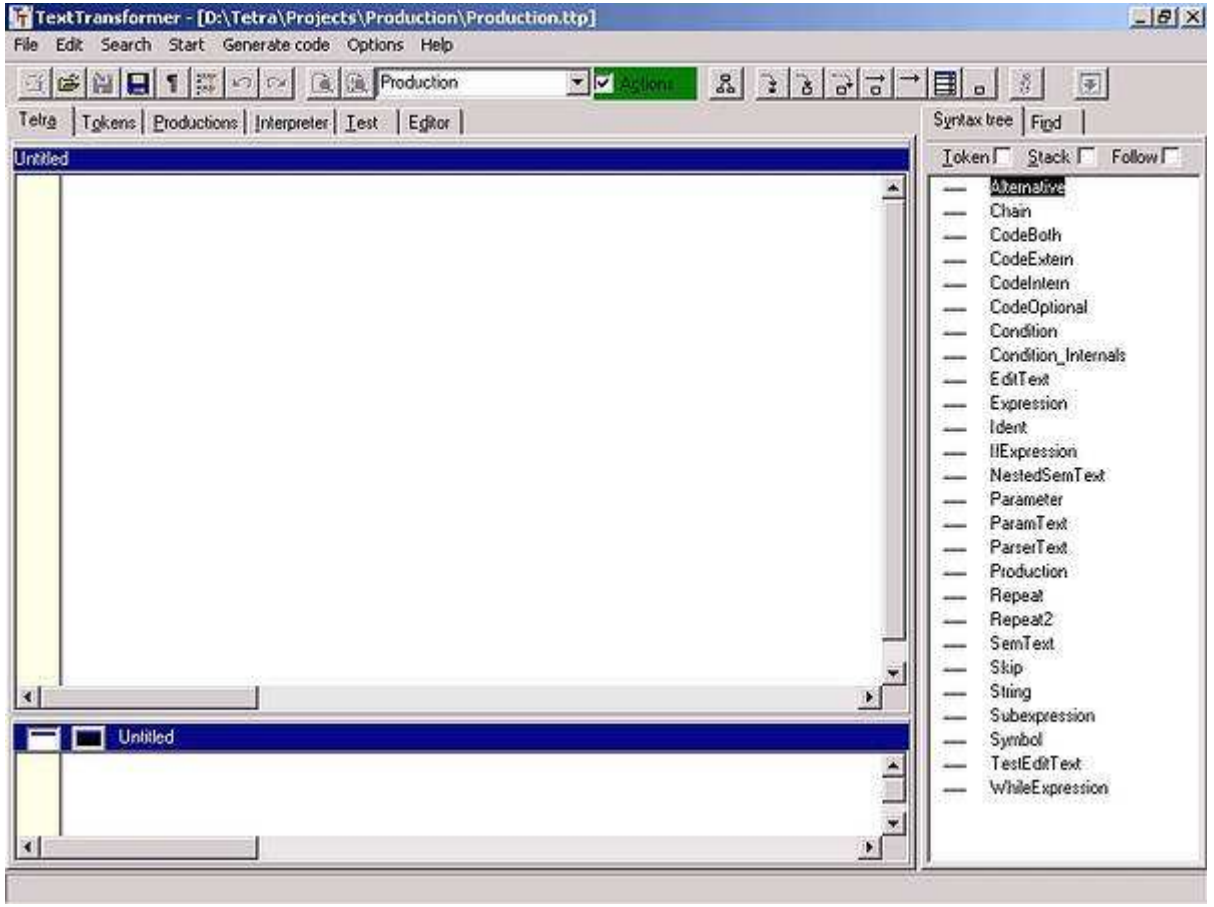
Graficzne środowisko jest cechą charakteryzującą ten wszechstronny program. Zawiera w sobie kombinację generatora parserów, prosty interpreter C++ oraz debuger.

Dzięki środowisku graficznemu programista może skupić się na tworzeniu gramatyki, a nie zgłębianiu zawiłości konstrukcji parsera czy skanera. Debuger umożliwia testowanie gramatyki bez potrzeby kompilacji kodu.

Program jest programem komercyjnym. Dopiero w wersji płatnej istnieje możliwość stworzenia kodu wynikowego w C++. Posiada kreatory oraz ułatwienia, które pomagają i przyspieszają tworzenie kodu.

TextTransformer

Okno graficzne programu:



TextTransformer

Narzędzie to nadaje się głownie do parsowania tekstów, chociaż działa on również jako zwykły generator parserów.

Strona główna projektu :http://texttransformer.com/

Na stronie http://www.texttransformer.org/ można znaleźć kilka projektów m.in. :

•Text2HTML umożliwia wygenerowanie prostej strony www ze zwykłego pliku Ascii

•ZahI2Nombre przykład konwersji liczb niemieckich na ich odpowiedniki w języku francuskim

AnaGram Parser Generator

- Home page: http://www.parsifalsoft.com/
- Generator parserów
- Środowisko interaktywne narzędzia, pozwala na debugowanie, wyświetlanie, edytowanie oraz testowanie gramatyk oraz parserów
- Zachowana funkcjonalność YACC, jednak znacznie wygodniejszy w użyciu, pozawala dopasować parser do własnych potrzeb
- Reguły napisane sa w C/C++, output również, istotnym faktem jest to, że AnaGram nie potrzebuje żadnych bibliotek runtime

AnaGram Parser Generator c.d.

- Ogromną zaletą AnaGram jest możliwość sprawdzania gramatyki jeszcze przed napisaniem akcji. Pozwalają na to File Trace oraz Grammar Trace. Przy ich pomocy można zobaczyć w jaki sposób parser zinterpretuje podany na wejście plik lub można wpisać (mamy okno z dostępnymi tokenami) bezpośrednio jakiś ciąg tokenów i zobaczyć czy nasza gramatyka je akceptuje.

- Poważną wadą narzędzia jest fakt, że wymaga ono licencji, która jest dość droga (aprox. 500$)

AnaGram Parser Generator c.d.

- Przykładowy parser (kalkulator) napisany z wykorzystaniem AnaGram’u, przykład jest szczegółowo opisany:

  http://www.parsifalsoft.com/examples/ffcex.html

Gold

- Home page: http://www.devincook.com/goldparser/
- Darmowy system do tworzenia parserów
- Posiada interaktywne środowisko pozwalające tworzyć gramatyki, testować je, anlizować oraz tworzyć na ich podstawie tablice parsera
- Głównym założeniem Gold'a jest współpraca z wiekszością języków programowania. Gold tworzy szkielet dla silnika, który jest osobnym fragmentem i może być rozwijany na większości platform. Według autorów ta cecha jest główną przewagą nad konkurencją.
- Notacja użyta do opisu gramatyki jest łatwa do zrozumienia. Korzysta ze standardowych formatów.

Gold c.d.

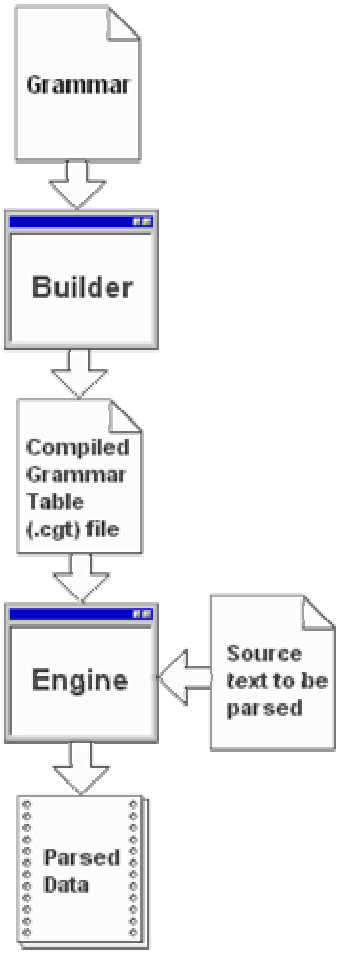
- Porównanie narzędzi pod względem ilości obsługiwanych języków

Free Parsing Systems					
Language	GOLD	YACC / Bison	ANTLR	Grammatica	Spirit
ANSI C	✓	✓			
C++	✓	✓	✓	✓	✓
C#	✓		✓	✓	
Delphi 5 & 6	✓				
Java	✓		✓		
Python	✓				
Visual Basic 6	✓				
Visual Basic .NET	✓				
All Other .NET Languages	✓				
All Other ActiveX Languages	✓				

Gold c.d.

- Rysunek przedstawia schemat działania Gold’a
- Przykład gramatyki opisujący blok „if-then-else”

  http://www.devincook.com/goldparser/doc/meta-language/index.htm



Lemon Parser Generator

- Home page: http://www.hwaci.com/sw/lemon/
- Generator paserów dla C/C++. Tłumaczy plik z gramatyką (*.y) (Context Free Grammar) na język C. W taki sposób powstaje pareser dla wejściowej gramatyki
- Funkcjonalność taka jak w przypadku YACC lub Bison jednak nie można powiedzieć, że jest to kolejny kolon

Lemon Parser Generator c.d.

- Zmieniona została składnie gramatyki. Ma to na celu zapobieganie błędom powstającym podczas jej tworzenia. (Główna zmiana to zrezygnowanie z symboli $$, $1 itd. Powiązano symboliczną nazwę z wyrażeniem w gramatyce)
- Użyto bardziej wyrafinowanego silnika parsera (szybsze działanie)

Spirit

- Home page: http://spirit.sourceforge.net/index.html
- Obiektowo zorientowany generator paserów
- Umożliwia pisanie gramatyki w C++
- Szczególnie przydatny gdy chcemy napisać „mocro-paser” na własne potrzeby (np. parsowanie command lines). Dla tak małych zadań użycie YACC’a lub Bisona mija się z celem.
- Spirit jest narzędziem skalowalnym, oznacza to, że ma wielkie możliwości rozbudowy. Nadaje się więc nie tylko do „mocro-parserów” ale także do zdecydowanie wiekszych projektów.

Spirit c.d.

- Całość systemu ma budowę warstwową. Oznacza to, że tworząc mało skomplikowany parser nie musimy uczyć się obsługi całego narzędzia, a jedynie jego fragmentu. Chcąc napisać coś większego poznajemy kolejne warstwy.
- Do zalet można zaliczyć wiele pre-definiowanych parserów

  (np. floating-point number – real_p, whitespace – space_p)
- Istotnym elementem jest też bardzo bogata i dobrze napisana dokumentacja

Coco/R for C++

- Home page: http://www.ssw.uni-linz.ac.at/Research/Projects/Coco/
- Tworzy na podstawie gramatyki skaner i parser
- W zależności od wersji obsługują różne języki (np. C++)