情報工学実験II

－ yacc/lexによる字句・構文解析系の作成－

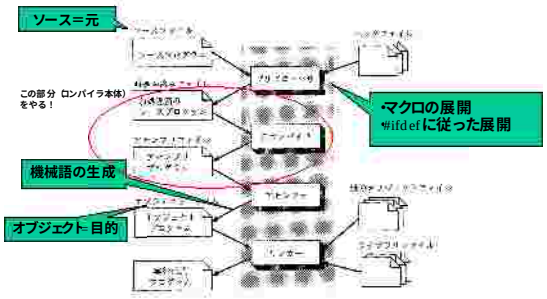
開　講：後期金曜日３－５限
対　象：情報科学科３年
単　位：B群
担　当：情報科学科　田中　賢

この実験の目標

- コンパイラの仕組み、特に字句解析、構文解析を理解する
- 字句・構文解析生成系である、lex、yaccの基本的な使い方を習得する

第１回　コンパイラの仕組み、様々な正規表現の書き方とlexによる字句解析系の生成
第２回　文法とその記述、yaccによる電卓の作成
第３回　簡単なプログラミング言語のインタプリタの作成
第４回　第3回の続き

コンパイラの概要　- Cコンパイラ-



- コンパイラの構造-

コンパイルの４つのステップ
1. 字句解析(lexical analysis)
2. 構文解析(syntax analysis)
3. 意味解析(semantic analysis)
4. コード生成(code generation)

1. 字句解析・・テキストを字句要素に分割

C言語における字句 (lexeme)

識別子(identifier)	変数名や関数名
キーワード(keyword)	int、returnなど用途が決まっている語
定数(constant)	値が一定の数。整数定数、浮動小数点定数、文字定数など
文字列リテラル(string literal)	文字列
演算子(operator)	+、*、=など
区切り記号(separator)	;など

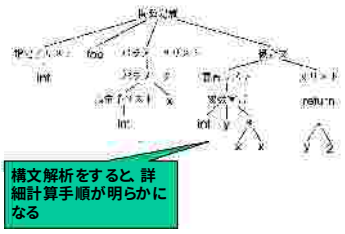
- アセンブリ言語と機械語-

関数の字句解析例）

- コンパイラの構造-

2. 構文解析・・文法のチェックと構文木の生成

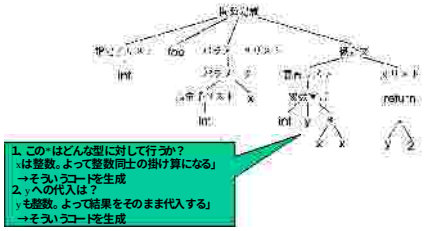
例） 関数fooの構文木



- コンパイラの構造-

3. 意味解析・・字句解析と構文解析以外の全ての解析

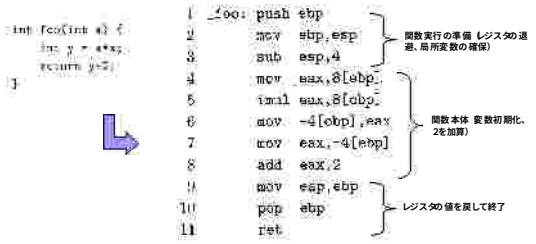
例） 関数fooの意味解析(主として型チェック)



- コンパイラの構造-

4. コード生成・・・オブジェクトコードの生成

例） 関数fooのコード生成



※インタプリタの場合は、オブジェクトコードを生成するかわりに、構文木を順次解釈しながら計算結果を求めていく。

字句解析

字句解析系を作るには・・・

1. 言語の字句要素を正規表現で表現
2. 1. を非決定性有限オートマトン(NFA)に変換
   受理状態のラベルによって字句要素を識別）
3. 2. を1つのNFAに合成
4. 3. を決定性有限オートマトン(DFA)に変換
5. 4. と等価な状態数最小のDFAを求める

この部分をlexで自動生成可能

字句解析系の動作・・・

プログラムを1文字ずつ読み込んで、DFAが字句要素を受理するたびにそれを書き出す。

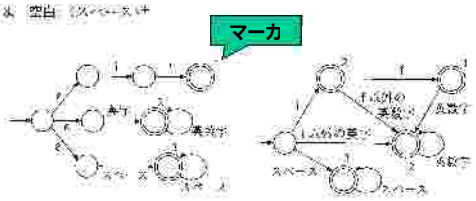
それをそのまま次の構文解析系に渡してやればよい。

- 字句解析プログラム-

【例2.12】 正規表現の例

1. キーワード if
2. 識別子（英字で始まる）
3. 空白（スペース）

※優先順位は1 > 2 > 3



ステップ2の段階　　最終的なDFA

- 字句解析プログラムの自動生成-

lex・・・lexical analyzer

字句要素切り出しプログラム（スキャナ）を生成するツール

正規表現で字句要素を記述したlexソースファイルを作成

↓ ファイル名 scan.l　　lexファイル

lex で字句解析プログラムを生成　% lex scan.l

↓ ファイル名 lex.yy.c

字句解析プログラムをコンパイル　% gcc lex.yy.c -ll

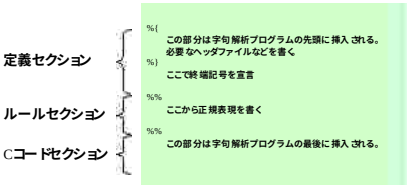
↓

字句解析プログラム完成　　cのソースファイル

lex
–lexical analyser–

・・字句解析プログラムを自動生成するツール

ファイル名は xxxx.l



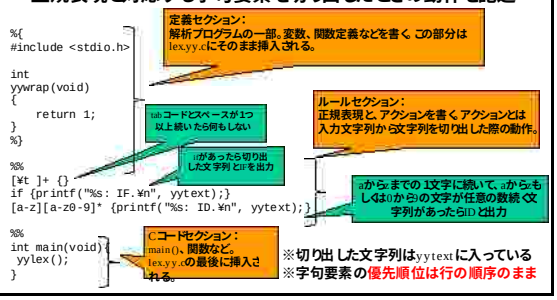
- lexの正規表現-

x	文字'x'にマッチ。
.	改行を除く全ての文字(バイト)。
[xyz]	'文字クラス'; この場合、'x', 'y', 'z' のいずれにもマッチ。
[abj-oZ]	範囲指定を含む'文字クラス'; この場合、'a', 'b'と'j' からo' までの任意のレターと'Z' にマッチ。
[^A-Z]	'否定文字クラス'; クラスに含まれない任意の文字に マッチ。 この場合、'A' から'Z' までの大文字 以外の」文字にマッチ。
[^A-Z¥n]	大文字と改行を 除く全ての文字。
r*	0 もしくはそれ以上の r。r は任意の正規表現。
r+	1 もしくはそれ以上の r。
r?	0 もしくは 1つの r。
r{2,5}	2 つから5つまでの r。
r{2,}	2 つ以上の r。
r{4}	ちょうど4つ の r。
"foo"	文字列foo
"[xyz]¥"foo"	文字列 [xyz]"foo (¥は次項目を参照)
¥X	X が 'a', 'b', 'f', 'n', 'r', 't', 'v' のいずれかの とき、ANSI-C での ¥X の解釈となる。 それ以外の場合、文字 'X' (*' のようなオペレータの 意味を打ち消し、その文字自体を指定する際に使う。
(r)	r にマッチ; ()は 優先順位を変えるために使用。
rs	正規表現 r に正規表現 s が続く連結(concatenation)。
r|s	r もしくは s。

- 字句解析プログラムの生成-

lexの使用例）

正規表現と対応する字句要素を切り出したときの動作を記述



定義セクションのマクロ

例）

```
%{
  ・・・
%}
digit  0-9
alpha a-z
%%
  ・・・
```

のように書くと、ルールの中のマクロをに置き換えてくれる。例えば、

```
{[digit]}* {printf("%s:number.¥n", yytext);}
{[alpha]}+ {printf("%s:words.¥n", yytext);}
```

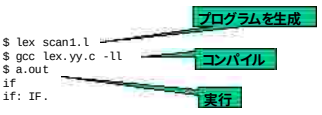
などと書けるようになる。

解析器の主な関数

int yylex() ・・・字句解析器の本体
・・・入力の先頭からパターンにマッチする字句要素を切り出してアクションを実行する
・・・パターンにマッチする最長の文字列を切り出す
・・・複数個の候補がある場合は上位の規則を優先
・・・マッチした文字列はグローバル変数yytextに格納
・・・入力文字列の終わりまで切り出しを繰り返す

int yywrap() ・・・入力がファイルの末尾に達したときに呼ばれる
・・・yywrapが0以外（真）を返せば終了、0(偽)を返せば次の入力ファイルを読みに行く
オプションの%noyywrap で省略可能

- lexと字句解析プログラムの実行例-




```
$ lex scan1.l
$ gcc lex.yy.c -ll
$ a.out
if
if: IF.

a
a: ID.

abc
abc: ID.

a9
a9: ID.

$
```

練習１　正規表現の練習）
以下の正規表現を作成し、様々な記号列を解析してみよ。
条件に一致しない記号列には、その旨メッセージを出力するようにしてみよ）

1)年齢　例）9、35、105
2)年齢　例）9 years old
3)郵便番号　例）112-3425
4)氏名　例）Ken Tanaka
5)/usr/binの直下にある英字のファイル名
ただし、UNIXのファイル名の長さは5文字までとする。
例)/usr/bin/ls
6)/usrで始まる任意の英字からなるディレクトリ名
ただし、UNIXのファイル名の長さは255文字までとする。
例)/usr/bin/TeX/

それぞれ異なる演算要素を含む正規表現を5つ作成し同様に解析してみよ。

課題１ lexの練習(要レポート)
算術式を解析する以下のlexソースからスキャナを作成し、様々な式を解析してみよ。

```
%%
[¥t ]+ {}
"+"|"-"|"*"|"/"|"=" {printf("%s (operator) ", yytext);}
";" {printf("%s (separator) ", yytext);}
[a-z][a-z0-9]* {printf("%s (identifier) ", yytext);}
[0-9¥.]+ {printf("%s (number) ", yytext);}
. {printf("%s (don't know) ", yytext);}
%%
```

レポートにはlexのソースと実行結果をのせること。

課題２ C言語スキャナ 要レポート)

1)C言語のキーワードを見つけ、"found keyword "というメッセージとともにそれを出力するプログラムを作成せよ。適当なC言語のプログラムから、以下のように実行するようにせよ。

```
% a.out < hoge.c
found keyword into
found keyword double
```
以下続く

参考） C言語のkeyword

auto break case char const continue default do double else enum extern float for goto if inline int long register restrict return short signed sizeof static struct switch typedef union unsigned void volatile while _Bool _Complex _Imaginary

2) 1)を元にして、さらに整数を見つけるように拡張せよ。整数とは0から9までの数の列で、符号は考慮しない。整数を見つけた際は、"Integer"と出力せよ。

3)さらに少数を見つけるように拡張せよ。少数とは、数の列の後に小数点が、さらに数の列が続くものとする。見つけた際は"Number"と表示せよ。

4)#で始まる行を読み飛ばすように拡張せよ。読み飛ばしていることを確認するためにまず"#line"と出力するプログラムを作り、動作を確認したらその表示を削除せよ。
※レポートには表示を削除したものをのせればよい。

5)識別子を認識するようにせよ。ただし、識別子とは変数名や関数名に対応する字句要素で、「アンダースコア"_"もしくは英字アルファベット(大文字と小文字)のいずれかのあとに、アンダースコア"_"、英字アルファベット、数字のいずれかが0個以上続くものである。見つけた場合は、"IDENT"と出力せよ。

6)各キーワードを見つけたら、その行番号と字面を表示するようにせよ。最後に、キーワードごとの総数をまとめて出力するようにせよ。

レポートにはlexのソースとCのソース、実行結果をのせること。Cのプログラムは指定された字句要素が含まれている適当なプログラムを各自用意してよい。

6)**の考え方**

```
%{
#include <stdio.h>

int
yywrap(void)
{
    return 1;
}
%}

%%

%%
main(){
        …
        yylex();
        result();
}

int result(){
        …
}
```

あらかじめ必要なグローバル変数を宣言して、

行数を数え、キーワードごとのカウンタをインクリメントしていく。

main()と、結果を出力する適当な関数を定義し、mainの中からcallする。

※行数はどうやったらカウントできるだろうか。

TIPS

エラー
premature EOF・・・%}**が見つからない。定義部を左詰めで書くようにする。**

レポート
DOS(SJIS)**への変換・・・**nkf –s –Lw unix**ファイル名** > DOS**ファイル名**

参考URL

lexはAT&Tで開発されたUNIXツール。本実験ではlinuxを用いるので、lex互換ツールであるFlexを用いている。実習環境ではlexはflexへのリンクになっている。Flexの日本語マニュアルは以下。

http://www.asahi-net.or.jp/~wg5k-ickw/html/online/flex-2.5.4/flex_toc.html