

コンパイラ演習 課題 3

2019年12月12日

課題 3 : 記号表の管理

- ▶ 構文解析時に、基本的な構造を持った記号表を作成するようにする
- ▶ 記号表への操作を行うコードを `parser.y` に追加
(別ファイルで定義してもよい←オススメ)
 - ▶ 変数・手続きの登録 (`insert`)
 - ▶ 変数・手続きの検索 (`lookup`)
 - ▶ 局所変数の削除 (`delete`)
- ▶ `pl0a.p`, `pl0b.p`, `pl0c.p`, `pl0d.p` をソースプログラムとして入力し、結果を確認

記号表

- ▶ プログラム中に出現した識別子を管理する表
 - ▶ 変数, 手続き名など
- ▶ 記号表が保持するデータ
 - ▶ 変数のスタック上でのアドレス
 - ▶ 手続きの先頭アドレス
 - ▶ ...

記号表の役割 1

- ▶ 変数のスタック上の位置を保持

スタック

2	z
1	y
0	x

記号表

x	0
y	1
z	2

```
program EX1;  
var x, y, z;  
...
```

記号表の役割 2

- ▶ 手続きの先頭アドレスを保持

記号表

x	0	glob
y	1	glob
z	2	glob
A	Pa	proc

```
program EX1;  
var x, y, z;  
procedure A;  
...
```

生成されたコードのうち、
手続きAのコードの開始位置

記号表の役割 2

- ▶ 手続きの先頭アドレスを保持

記号表

x	0	glob	
y	1	glob	
z	2	glob	
A		proc	Pa

```
program EX1;  
var x, y, z;  
procedure A;  
...
```

別のカラムを
用意してもよい

記号表の役割 3

- ▶ 大域変数と局所変数, 手続きの
区別を保持

```
program EX1;  
var x, y, z;  
procedure A;  
var x;  
...
```

x	0	glob
y	1	glob
z	2	glob
A	Pa	proc
x	0	loc

```
program EX1;  
var x, y, z;  
procedure A;  
var x;  
...  
end;  
procedure B;  
...
```

x	0	glob
y	1	glob
z	2	glob
A	Pa	proc
B	Pb	proc

課題 3 で実現する記号表に対する操作

- ▶ 変数・手続きが宣言されたら記号表に登録(insert)
 - ▶ 変数・手続き名
 - ▶ 大域変数, 局所変数, または手続きの区別

課題3ではまだ, 変数のスタック上の位置や手続きの開始位置については正確な値を登録しなくてもよい。

- ▶ 変数・手続きが呼び出されたら記号表を検索(lookup)
- ▶ 手続きの終了時点で全ての局所変数を削除(delete)

* 各操作作用の関数を定義する

課題 3 で実現する関数（テスト用出力）

- ▶ 変数・手続きが宣言されたら記号表に登録(insert)
 - ▶ 記号表のすべてのデータ を出力
- ▶ 変数・手続きが呼び出されたら記号表を検索(lookup)
 - ▶ 変数名と大域変数・局所変数・手続きの別を出力
- ▶ 手続きの終了時点で全ての局所変数を削除(delete)
 - ▶ 記号表のすべてのデータ を出力

yaccへのコードの埋め込み (1)

- ▶ 構文中の適切な箇所に、処理（アクション）を実行するCプログラムを記述
 - ▶ 構文規則名 : ボディ **{アクション}** ;
- ▶ 例
 - ▶ 予約語programが認識されたら,
“program is inputted”と出力する.

```
program
    : PROGRAM
      {printf (“program is inputted.¥n”);}
      IDENT SEMICOLON outblock PERIOD
    ;
```

yaccへのコードの埋め込み (2)

▶ 擬似変数の利用

▶ 例：IDENTと認識されたトークンを出力

```
program
    : PROGRAM IDENT SEMICOLON outblock PERIOD
    {printf ("%s¥n", $2);} ;
```

- ▶ \$2：構文規則の右辺の2番目にあるIDENTに対する擬似変数
- ▶ parser.yの宣言部に %token <ident> IDENT とあり、scanner.l で、トークンが IDENT と認識されるとき、
strcpy(yylval.ident, yytext); としているので、
\$2 は`program`の後に現れる識別子を表す。

コードの埋め込み箇所

- ▶ 記号表への登録
 - ▶ 変数・手続きの宣言時
- ▶ 記号表からの局所変数の削除
 - ▶ 手続きの終了時
- ▶ 記号表の検索
 - ▶ 変数・手続きの呼び出し時

ヒント：コードの埋め込み例

▶ 仮定

- ▶ 記号表への登録関数：insert_data
- ▶ 第1引数：変数名
- ▶ 第2引数：大域変数か局所変数かを区別するflag

id_list

```
: IDENT { insert_data( $1, flag ); }
```

```
| id_list COMMA IDENT{ insert_data( $3, flag ); }
```

```
;
```

その他必要な処理

- ▶ 大域変数か局所変数かを識別するコード
 - ▶ 実現例：大域変数か局所変数かを表す変数を用意し、手続きの開始時、終了時に値を操作
 - ▶ 手続きの構文規則
 - ▶ inblockの中で局所変数の宣言と、手続きの定義を記述

proc_decl

: PROCEDURE proc_name SEMICOLON inblock;

PL0Bでの結果

```
program PL0B;
```

```
var n, x; ←
```

```
procedure prime; ←
```

```
var m; ←
```

```
begin
```

```
  m := x div 2; ←
```

```
  while x <> (x div m) * m do ←
```

```
    m := m-1; ←
```

```
  if m = 1 then ←
```

```
    write(x) ←
```

```
end; ←
```

```
begin
```

```
  read(n); ←
```

```
  while 1 < n do ←
```

```
  begin x := n; ←
```

```
    prime; ←
```

```
    n := n-1 ←
```

```
  end
```

```
end.
```

変数n, xを記号表に登録

手続きprimeを記号表に登録

変数mを記号表に登録

変数m, xを検索

変数m, xを検索

変数mを検索

変数mを検索

変数xを検索

局所変数を削除

変数nを検索

変数nを検索

変数x, nを検索

手続きprimeを検索

変数nを検索

プログラム及びレポートの提出

▶ プログラムの提出

▶ プログラムファイル

- ▶ Makefile, parser.y, scanner.l, その他作成した全ファイル

▶ 上記ファイルを kadai3 というディレクトリに保存

- ▶ 以下のコマンドで圧縮し, kadai3.tar.gzを提出
(kadai3 のディレクトリがある階層で)
`tar zcvf kadai3.tar.gz kadai3`

▶ レポートのPDF

- ▶ 作成したプログラムにおける記号表の構造と、各操作関数を挿入する位置について解説
- ▶ ./parser pl0a.p に対する実行結果
(出力内容はスライドp.9参照)