情報工学実験
コンパイラ
再帰下降型構文解析
(2015年度)

内容

- BNFによる言語定義
- 表駆動型オートマトンによる字句解析
- 演算子順位構文解析
- 再帰下降型構文解析
- オブジェクトコード生成
  - 本実験では CPU 実験で作成する CPU 用のアセンブリ言語をオブジェクトコードとする

http://www.edu.cs.okayama-u.ac.jp

スケジュール

4/13　　全体説明・言語定義説明
4/20　　字句解析解説・実装
4/21, 27 字句解析実装
4/28, 5/7　演算子順位構文解析説明・実装
5/11,12　演算子順位構文解析実装
5/18,19　演算子順位構文解析実装
5/25　　再帰下降型構文解析説明・実装
6/1　　再帰下降型構文解析実装
6/8, 15, 22 コード生成説明・実装
6/29　　中間面接（コード生成）
7/6, 7/13　コード生成
7/27, 28, 8/3, 4　面接

再帰下降型構文解析の特徴

- 下向き構文解析
- 言語の定義がそのままプログラムになる

再帰下降型構文解析の例

<文> ::= <if文> | <代入文>
<if文> ::= if 左括弧 <条件式> 右括弧　<文>
<代入文> ::= 変数　代入記号　<算術式>

```
void 文の解析(FILE *fp) {
    token = nextToken(fp);
    if (tokenのタイプ == if) {
        ungetToken();
        if文の解析(fp);
    }
    else if (tokenのタイプ == 変数) {
        ungetToken(fp);
        代入文の解析(fp);
    }
}
```

if文の解析の例

<if文> ::= if 左括弧 <条件式> 右括弧　<文>

```
void if文の解析(FILE *fp) {
    token = nextToken(fp);
    if(tokenのタイプ != if) エラー処理();
    token = nextToken(fp);
    if(tokenのタイプ != 左括弧) エラー処理();
    条件式の解析(fp);
    token = nextToken(fp);
    if(tokenのタイプ != 右括弧) エラー処理();
    文の解析(fp);
}
```

条件式の解析の例

<条件式> ::= <算術式> <論理記号> <算術式>

```
void 条件式の解析(FILE *fp) {
    Oparser(fp);
    論理記号の解析(fp);
    Oparser(fp);
}
```

代入文の解析の例

<文> ::= <if文> | <代入文>
<if文> ::= if 左括弧 <条件式> 右括弧　<文>
<代入文> ::= 変数　代入記号　<算術式>

```
void 代入文の解析(FILE *fp) {
        token = nextToken(fp);
        if (tokenのタイプ != 変数) エラー処理();
        token = nextToken(fp);
        if(tokenのタイプ != 代入記号) エラー処理();
        Oparser(fp);
}
```

左再帰性の除去

- 左再帰性がある場合
  <文集合> ::= <文集合> <文> | <文>
  ```
  void 文集合の解析(FILE *fp) {
        文集合の解析(fp);
        …
  }
  ```
- 左再帰性のない場合
  <文集合> ::= <文> <文集合> | <文>
  ```
  void 文集合の解析(FILE *fp) {
        文の解析(fp);
        文集合の解析(fp);
        …
  }
  ```

確認のための出力

```
void 代入文の解析(FILE *fp) {
        printf(”代入文解析の始まり\n”);
        token = nextToken(fp);
        if (tokenのタイプ != 変数) エラー処理();
        token = nextToken(fp);
        if(tokenのタイプ != 代入記号) エラー処理();
        Oparser(fp);
        printf(”代入文解析の終わり\n”);
}
```

本日の作業内容

- 作業目標
  - まだ演算子順位構文解析の部分が終了していないグループはまずそれを終了させ，TAに確認してもらい，笹倉に報告すること
  - 言語全体の構文解析関数 Parse を作成
    - 算術式は演算子順位構文解析
    - それ以外の部分は再帰下降型構文解析
- まず最初にどの非終端記号に関数を作るかとその関数名を決定
- 誰がどの関数を作るかをグループ内で決定
- 作業報告書でどこまでできたかを報告のこと．のこりは次回の実験の時間に．

注意！

- 以下の文法の構文解析のプログラムを書く必要はない
  - <整数> ::= <数字><整数> | <数字>
    → これはすでに字句解析で解析済み
  - <項> ::= <因子> * <項> | <因子>
    → これはすでに演算子順位文法で解析済み

注意点

- 再帰下降型構文解析のプログラムソースはこれまでとは別なファイルにする.
  例：parse.c
  作成者ごとに別のファイルにしてもよい.

その他の必要な作業

- Makefile の更新
- main.c の変更
  <プログラム>の解析を行う関数を呼ぶように書き換える.
- 入力サンプルプログラムの作成

実験の進め方

1. 作る関数の仕様を決定する
2. 分担してプログラムを書く.
3. プログラムをコンパイルし，実行する.
4. バグがあれば修正する.
5. 正しい答えが出れば終了.

実験をすすめる上での注意

- 入力に使うサンプルプログラムは各グループで定義した文法の内容に合ったものを各グループで用意すること．その際，文法の規則をすべてテストできるようにすること．
- 作ったプログラムが思うように動かないときはまず担当TAに相談すること．