レポート課題

以下の問題について，この `report.ipynb` を編集し，完成した自分が編集した .ipynb ファイルをGoogle Classroomの該当課題に提出してください．（もしこのノートブックをファイルに保存したいときには，メニューの「file」→「Download」で .ipynb ファイルをダウンロード出来るので，それを提出してください．）

- 適宜コメントを入れてください．
- 提出する際のファイル名は，「`学籍番号.ipynb`」としてください．
- 問題1について，Markdownで書きにくい場合には，ClassroomにPDFファイルを追加して提出してもかまいません．
- 発展課題は任意とします．
- Markdownを書くときには，Wikiにあるぐらいの基本的なマークアップで書いてくれればよいです（もちろん，自分で調べてもっと高度なMarkdownを書いてくれても見られればよいです）．

問題1 - LALR構文解析についてまとめる（第2回の内容から）

Yaccは，***LALR構文解析***を行う構文解析器を出力するということを第2回で説明しました．

LALR構文解析について，以下の用語などを交えて説明してください．（すべてが含まれている必要はないですが，自分でちゃんと理解したうえでまとめること）

- 上向き構文解析
- スタック
- 構文解析表
- Shift / Reduce
- shift/reduce conflict の例
- reduce/reduce conflictの例
- 第1回の問題5（右再帰と左再帰で，どちらがYaccにとって良いか．）

ヒント1: 書籍が沢山出ているので，それを参考にしてください．（僕らのときは，）

> 『情報科学こんせぷつ8 コンパイラの仕組み』渡邊 坦(著) ISBN：978-4-254-12708-9

が教科書に指定されていました．

ヒント2: 今回のノートブックが裏で使っているBisonのページは次のとおり，The Bison Parser Algorithm

問題2 - 電卓の拡張（第1回の内容から）

第1回で作成した電卓を自由に拡張して，以下にコードを載せてください．どんな機能を追加したかをMarkdownにまとめて書いて，コードには適宜コメントを入れてください．

- 授業の練習問題で自分が作成したものを含んでいてもよいです．
- Yaccの**演算子順序**について自分で調べて利用してくれてもよいです．
- 授業で対応していなかった演算・実数・関数に対応するなど，一般の関数電卓にある機能でもよいですし，自分で考えた新たな機能でもよいです（プログラムになってしまうと，次の問題3になってしまうので，あくまで電卓の範囲で）

どんな機能を追加したのか，ここに書く

In [ ]:
```
/* lex f2.lex */
```

In [ ]:
```
/* yacc f2.yacc */
```

In [ ]:
```
/* c f2.c */
```

In [ ]:
```
/* a.out */
自分の電卓の機能をアピールできるような実行例を示してください．
```

問題3 - 自分のプログラミング言語の拡張（第2・3回の内容から）

第2・3回で作成したプログラミング言語を自由に拡張して，以下にコードを載せてください．どんな機能を追加したか（複数推奨）をMarkdownでまとめて書いて，コードには適宜コメントを入れてください．

- 授業の練習問題で自分が作成したものを含んでもよいです．

- インタプリタでもコンパイラでもどちらでもよいです．
  - コンパイラの場合，出力先の言語は授業でやったC・x64アセンブリ言語のどちらかにしてください（出力先を変えてみるのは，以下で任意発展課題としています）
- たとえば，以下のような拡張が考えられます．（もちろん，これ以外の拡張もどんどん入れてください）
  - `if` 文
  - `for` 文
  - `switch` 文
  - サブルーチン（値を返さない）
  - 関数（値を返す）
    - アセンブリ言語でやるとしたら，ラベルなどを付けてジャンプすることになるはず．どうスタックを使うかなどを考えないといけない．
  - 配列（1次元でよい）

どんな機能を追加したのか，ここに書く

In [ ]:
```
/* lex f3.lex */
```

In [ ]:
```
/* yacc f3.yacc */
```

In [ ]:
```
/* c f3.c */
```

In [ ]:
```
/* a.out */
自分のプログラミング言語をアピールできるような実行例を示してください．
```

発展問題1 - コード生成（第3回の内容から）

第3回で，C言語とx64アセンブリ言語に出力するコンパイラを作成しました．それを参考にして，他のプログラミング言語に出力するコンパイラを作成してください．どんなプログラミング言語に出力するかと，必要に応じて実行モデルについてどうしたか（翻訳するときにどう決めたか）をMarkdownにまとめて書いて，コードには適宜コメントを入れてください．

- 自分の知っているどんな言語でもいいです．
  - 言語処理系論で出てきた WebAssemblyについて自分で調べてみて出力できると，おもしろいかもしれない（その場合は呼び出すJavaScriptも書いてほしい）．

どんな言語に出力したのか，ここに書く

In [ ]:
```
/* lex f4.lex */
```

In [ ]:
```
/* yacc f4.yacc */
```

In [ ]:
```
/* c f4.c */
```

In [ ]:
```
/* a.out */
```

発展問題2 - プログラミングではない用途（第3回の内容から）

第3回で，簡単な英文解析プログラムを作成しました．同様にして，プログラミング言語ではない用途でLex・Yaccを使用したものを自由に作成して，以下にコードを載せてください．どんなプログラムかをMarkdownでまとめて書いて，コードには適宜コメントを入れてください．

- 英文の例を単純に拡張するのでもよいです．
- JSONやXMLのようなデータを読み込むプログラムでもよいです．
- コマンドのオプションを読み込むときに使うプログラムでもよいです．
  - たとえば，`ssh` だったら（`% ssh -l ユーザー名`）のようにコマンドを書くけれど，それを解釈するプログラム．間違っていたら`man`コマンドの結果のように使い方を表示すしてみるなど

どんな機能を追加したのか，ここに書く

In [ ]:
```
/* lex f5.lex */
```

In [ ]:
```
/* yacc f5.yacc */
```

In [ ]:
```
/* c f5.c */
```

In [ ]:
```
/* a.out */
```

発展問題3 - 最適化

授業資料で作成したコンパイラは，抽象構文木を作成しただけでした． 抽象構文木の構造をいじってやれば，コンパイラに最適化を実装できます．この課題では，自由に考えて（調べて）自分が作成したコンパイラになんらかの最適化手法を取り入れてください．

- ここまでの問題のどのバージョンのコンパイラでもよいです．
  - 最適化の効果が見やすいので，C言語にコード生成してくれるとありがたい．
- たとえば，
  - 別々の式になっていた足し算を1つにマージするとか（実際問題として実行速度が上がるわけではないが，抽象構文木の形を変える練習として）
  - SSA 形式に変換してみるとか
  - ループ最適化とか

In [ ]:
```
/* lex f6.lex */
```

In [ ]:
```
/* yacc f6.yacc */
```

In [ ]:
```
/* c f6.c */
```

In [ ]:
```
/* a.out */
```