2016年度 情報科学演習D 指導書

1 演習の概要

1.1 目的

比較的簡単な計算機言語の処理プログラムを設計，作成することにより，言語処理系の基本的な機能を理解する．

- 授業期間：2016 年 10 月 06 日 (木) 〜 2017 年 2 月 2 日 (木)（計 15 回）
- プログラミング言語：Java 言語
- 関連する講義：言語処理工学 A（単位認定は独立に行われる）
- 演習 D ウェブサイト：`http://sdl.ist.osaka-u.ac.jp/~shinsuke/enshud/2016/`

なお，以降では本資料を「指導書」と呼ぶ．

1.2 課題

Pascal 風言語で記述されたプログラムをアセンブラ言語 CASL II で記述されたプログラムに翻訳（変換）するコンパイラを作成しなさい．具体的には，言語処理工学 A の講義内容と進捗に合わせて，次の 4 つの課題を行う．

1. 副課題 1 字句解析器（Lexer）の作成
2. 副課題 2 構文解析器（Parser）の作成
3. 副課題 3 意味解析器（Checker）の作成
4. 最終課題 コンパイラ（Compiler）の作成

1.3 演習環境

情報科学演習 D では統合開発環境 Eclipse の利用を前提としている．演習室のマシンには Eclipse の実行環境がすでにインストールされている．演習室にインストールされているバージョンは，Java が 1.8.0，Eclipse は 4.5.2 である．演習室以外の環境で作業を行っても構わないが，Java 環境の構築に十分留意すること．

演習で用いるリソース群（サンプルデータ等）を演習用ツールキット（`enshud-toolkit-2016.zip`）として配布している．演習 D ウェブサイトからツールキットをダウンロードし，以下の手順に従って展開すること．

- ツールキットをダウンロードし展開する．

  `http://sdl.ist.osaka-u.ac.jp/~shinsuke/enshud/2016/enshud-toolkit-2016.zip`

```
% cd ~
% unzip enshud-toolkit-2016.zip
```

展開すると enshud-toolkit-2016 というディレクトリが作成され，その中に演習に必要なプログラムおよびデータが入っている．

- 環境変数 PATH の先頭を~/enshud-toolkit-2016/bin とする．

  csh を利用している場合：

  ```
  % set PATH=(~/enshud-toolkit-2016/bin:$PATH)
  ```

  bash を利用している場合：

  ```
  $ export PATH=~/enshud-toolkit-2016/bin:$PATH
  ```

  csh を利用している場合，環境変数 PATH の設定は，.cshrc ファイル中で set コマンドによって行うことができる．また，bash の場合は.bashrc ファイル内にて指定する．

- プログラムを作成する際は，プライベートな（他ユーザの読み取り権限のない）ディレクトリを作成し，その中で作業すること．他ユーザからのコピー防止のためである．具体的なコマンドは以下の通り．

  ```
  % chmod 700 enshud-work
  ```

1.4 演習実施上の注意

- 副課題，最終課題に限らず他人のプログラムの一部，または全部をコピーした者に対しては単位認定を行わない．同様に他人のレポートのコピーを行った者に対しても単位認定は行わない．
- 数名の大学院生が，TA として演習の補助を行う．授業中にわからないことがあれば遠慮なく質問すること．また，授業時間外でも聞きたいことがある場合には，⟨enshu-d@fenrir.ics.es.osaka-u.ac.jp⟩ 宛にメールを送ることにより質問をしてもよい．
- こちらからの連絡は主に@ecs.cmc.osaka-u.ac.jp 宛の電子メールにて行うので，演習期間中はチェックしておくこと．あるいは，チェックできるアドレスに転送すること．
- 副課題 3 と最終課題については，それぞれその前に作成したプログラムを再利用することを前提に演習の期間が設定されている．従って，言語処理工学 A のテキスト等を参考にして，再利用を十分考慮しながら演習に取り組むこと．
- yacc 等のコード生成ツールを用いてはならない（字句解析から CASL II プログラム生成までを自ら作成することによってコンパイラとその関連技術の理解を深めることが情報科学演習 D の目的の 1 つであるため）．

1.5 レポートの書き方

レポートを書くにあたって下記の事項に留意すること．

- 課題の目的として，どのようなプログラムを作成するのかについて記述すること．また，取り組んだ発展課題があれば，どのようなことに挑戦したのかも書くこと．

- 所望の動作をするプログラムを作成するためにどのような原理に基づいて処理を行うか（どういう問題をどう解くか），また実装の方針としてどのようなアルゴリズム，データ構造を用いたのかを記述すること．
- 実装した個々の関数の仕様や説明を書く必要はない．
- 良く推敲すること．主語・述語の対応はいいか，そこで述べていることの前提となっていることは既に述べられているかなどをチェックする．
- 図や表を積極的用いてわかりやすい説明を心がけること．図表には番号及びキャプションを付け，本文からは番号で参照すること．
- 以前の副課題と重複する説明は省いて良い．省略する場合は，以前の副課題レポートで説明済みであることがわかるように記述すること．もし再度説明する場合は簡潔に記述すること．図表は流用しても良い．
- 最終課題のレポートでは，下記の項目については必ず記述すること．
  - 変数への値の代入に関するコード生成
  - 式に関するコード生成
  - 手続き呼び出し文に関するコード生成
  - 副課題で作成したプログラムをうまく再利用できたかどうか，およびその理由
- 指導書の仕様の中に曖昧な点，不完全な点があればどのように定めたかを書くこと．
- 最終課題において発展課題を行った場合（あるいは副課題で追加の機能を実装した場合）は章や節を設けて，何をどのように行ったのかを記述すること．
- そのプログラムは正しく動作したか．また，どうやってその確認を行ったのかを記述すること．
- まとめでは，この課題で何を行い，何を達成したのかを簡潔に述べる．
- 感想では，うまくできたと思った部分，うまくできなかった部分（こうしたかったという部分），取りにくかったバグ，便利だったツール，課題の改善案などについて書くこと．今後の参考にするため必須の項目とする．
- 参考にした資料があれば参考文献として挙げること．どの部分を参考にしたのかをレポート内に記述し，引用すること．

  例：式を表現するデータ構造については文献 [1] を参考にした．
- 友人からアドバイスをもらった場合は謝辞で記述すること．
- レポートの基本的な構成は以下の通りである．適宜，節を追加すること（特に 2，3，4 章）．また，必要に応じて構成を変更しても良い．
  - 1. 課題の目的
  - 2. プログラムの動作原理
  - 3. プログラムの実装方針
  - 4. プログラムの動作確認
  - 5. まとめ

- 6. 感想
- 謝辞
- 参考文献

- レポートの分量は評価の対象ではない．ページ数が多いほど評価が高くなるわけではない．実装した個々の関数に関する項目等の不必要な記述があり，レポートの構成が悪くなっている場合には，不必要なそれを書かない場合に比べて評価が低くなる場合もある．
- レポートは必ず PDF 形式のものを提出すること．レポートの作成にあたっては必ずしも LaTeX を用いる必要はなく，Word 等を用いても構わない．

1.6 レポートの提出方法

全ての課題について，レポートとソースコードの提出は以下の手順に行うこと．手順に従わない場合，提出を認めないので十分に注意すること．

1. まず，提出ディレクトリに自分のアカウント名のディレクトリを生成する．

   提出ディレクトリ：/home/exp/exppub/expd/2016/s-matsumt （アカウント名が s-matsumt の場合）

2. ソースコードから実行可能な Jar ファイルを生成する．Jar のファイル名は課題名（Lexer, Parser, Checker, Compiler）とすること（Lexer.jar 等）．Eclipse からの Jar の生成方法は別紙参照．

3. Jar とは別に，ソースコード一式を zip で固める．zip のファイル名は課題名（Lexer, Parser, Checker, Compiler）とすること（Lexer.zip 等）．固める対象となるディレクトリは src 以下全てで良い．コマンドは以下の通り．

   ```
   % zip -r Lexer.zip src/
   ```

4. Jar ファイルと zip ファイルを提出ディレクトリに設置し，~~両ファイルのパーミッションを，他のユーザが読み取りできないように変更する．~~各ファイルのパーミッションを以下のように変更する．

   ```
   % chmod 755 Lexer.jar
   % chmod 600 Lexer.zip
   ```

5. 確認依頼システムよりプログラムの動作確認依頼を出す．確認依頼システムは学籍番号とアカウント名が一致していないと失敗するので注意すること．この依頼でもって，TA がプログラムの動作確認を行う．なお，動作確認依頼の締め切りは，当該レポート締め切り日前日の授業終了時間とする．

   確認依頼システム：`http://sdl.ist.osaka-u.ac.jp/~shinsuke/enshud/2016/checkme/`

6. 動作確認の結果が電子メールで@ecs.cmc.osaka-u.ac.jp 宛てに返ってくるので，それを確認する．

   - 動作確認が成功していた場合: CLE[1] を通じてレポート（PDF）を提出する．PDF のファイル名は学籍番号の下 4 桁とフルネーム（ローマ字で姓，名の順）および課題名（LEXER, PARSER, CHECKER, COMPILER）をアンダースコアでつないだものとすること．ファイルのアップロード日時をレポートの提出時刻とみなす．

     例：`1001_MATSUMOTO_SHINSUKE_LEXER.pdf`

[1] `https://cle.koan.osaka-u.ac.jp/`

- 動作確認に失敗していた場合: 再度 Jar の設置と動作確認依頼の送信を行うこと．

なお，TA は演習時間に勤務していることになっているので，それ以外の場合は返答が遅くなる場合がある．原則として動作確認返答の遅れをレポート提出期限超過の理由には認めないので，**早めに動作確認を依頼すること**．

1.7 単位取得条件

1. 60 点以上
2. 欠席 3 回未満（遅刻は 0.5 欠席）

以上の条件を全て満たすこと [2]．ただし，最終課題のレポート提出以降は出席を免除する．なお，**評価として S を得るためには，最低でも 1 つの発展課題を行う必要がある．**発展課題については，6.5 節を参照すること．

1.8 授業時間スケジュール

基本は毎週木曜日の 3 限 4 限である．ただし，以下スケジュールは変更される可能性がある．現在日程は未定であるが，言語処理工学 A との振替えがどこかで 1 回以上発生する．**最新情報は演習 D ウェブサイトに反映するのでウェブサイトを必ず確認すること．**

第 N 回	日時	備考
第 1 回	10 月 06 日 (木)	
———	10 月 13 日 (木)	*言語処理工学 A と振替え
第 2 回	10 月 20 日 (木)	
第 3 回	10 月 27 日 (木)	副課題 1: 確認依頼〆 & レポート〆の前日
———	11 月 03 日 (木)	*祝日
第 4 回	11 月 10 日 (木)	
第 5 回	11 月 17 日 (木)	
第 6 回	11 月 24 日 (木)	副課題 2: 確認依頼〆 & レポート〆の前日
第 7 回	12 月 01 日 (木)	
第 8 回	12 月 08 日 (木)	
第 9 回	12 月 15 日 (木)	
第 10 回	12 月 22 日 (木)	副課題 3: 確認依頼〆 & レポート〆の前日
———	01 月 05 日 (木)	*月曜講義との振替え
第 11 回	01 月 12 日 (木)	
第 12 回	01 月 19 日 (木)	
第 13 回	01 月 20 日 (金)	
第 14 回	01 月 26 日 (木)	
第 15 回	01 月 27 日 (金)	*言語処理工学 A と振替え & 最終課題 確認依頼〆当日
第 16 回	02 月 02 日 (木)	最終課題 レポート〆前日

[2] 各レポートの提出が遅れた場合はその遅延日数に応じて減点される．しかし，欠席回数が 3 回未満であれば（減点も含めた）素点が 60 点以上なら単位習得は可能である．欠席回数 3 回以上の受講生は何れかのレポート提出が少しでも遅れた場合は単位取得はできない．

1.9 レポート提出締切

レポート締切ギリギリに課題を終わらせれば良いという考えでは，レポート締切に間に合わないことがよくある．副課題 1 と副課題 2 は 6 コマ，副課題 3 と最終課題は 8 コマを目安に作業をすること．

対象	日時
副課題 1 確認依頼締切	10 月 27 日 (木) 16:10
副課題 1 レポート締切	10 月 28 日 (金) 18:00
副課題 2 確認依頼締切	11 月 24 日 (木) 16:10
副課題 2 レポート締切	11 月 25 日 (金) 18:00
副課題 3 確認依頼締切	12 月 22 日 (木) 16:10
副課題 3 レポート締切	12 月 23 日 (金) 18:00
最終課題 確認依頼締切	01 月 27 日 (金) 18:00
最終課題 レポート締切	02 月 03 日 (金) 18:00

1.10 スタッフ

今年度は下記のスタッフ（教員および TA）が演習 D を担当する．

- 教員：柗本 真佑，樽谷 優弥
- TA：佐飛 祐介，幸 佑亮，中島 弘貴，山本 将弘

2 Pascal 風言語

Pascal 風言語は，関数やレコード構造などの Pascal 本来の機能のいくつかは取り除かれている．Pascal との主な構文上の違いは次の通り．

1. 扱えるデータ型は標準型と呼ぶ定義済みの型 integer，char, 及び，boolean とその配列型（1 次元）のみである．型定義などによって新たな型を作ることはできない．
2. 文は代入文，手続き呼出し文，入出力文，if 文，while 文のみである．文にラベルをつけることはできない．また，**if 文の実行部は常に複合文（begin で始まり end で終る）でなければならない．**
3. 手続きの宣言を入れ子にすることはできない．また，パラメータの引き渡し方法は，値呼び（call-by-value）のみである．
4. 入出力は標準入出力を対象とする readln 文，及び，writeln 文によってのみ行うことができる．

以下に，Pascal 風言語の構文定義を拡張バッカス・ナウア記法（EBNF: Extended Backus-Naur Form）を用いて示す．各構文規則は EBNF 式によって構文要素を定義する．この式は 1 つの句か，あるいは，いくつかの選択可能な句を縦棒（|）で区切ったものからなり，最後はピリオド（.）で終る．句は構文要素名，引用符（""）で囲んだ文字記号，あるいは，中かっこ，角かっこ，または，かっこで囲んだ別の EBNF 式を要素とし，0 個以上の要素からなる．中かっこ { と } は繰返し（0 回以上の出現）を，角かっこ [と] は選択が任意であること（0 回または 1 回の出現）を示す．

```
プログラム = "program" プログラム名 "(" 名前の並び ")" ";"          (1)
             ブロック
             複合文 ".".
プログラム名 = 名前.                                               (2)
名前の並び = 名前 { "," 名前 }.                                    (3)
ブロック = 変数宣言                                                (4)
           副プログラム宣言群.
変数宣言 = [ "var" 変数宣言の並び ].                               (5)
変数宣言の並び = 変数名の並び ":" 型 ";" { 変数名の並び ":" 型 ";" }.  (6)
変数名の並び = 変数名 { "," 変数名 }.                              (7)
変数名 = 名前.                                                     (8)
型 = 標準型 | 配列型.                                              (9)
標準型 = "integer" | "char" | "boolean".                           (10)
配列型 = "array" "[" 添字の最小値 ".." 添字の最大値 "]" "of" 標準型.  (11)
添字の最小値 = 整数.                                               (12)
添字の最大値 = 整数.                                               (13)
整数 = [ 符号 ] 符号なし整数.                                      (14)
符号 = "+" | "-".                                                  (15)
```

副プログラム宣言群 = { 副プログラム宣言 ";" }. (16)

副プログラム宣言 = 副プログラム頭部 (17)
変数宣言
複合文.

副プログラム頭部 = "procedure" 手続き名 仮パラメータ ";". (18)

手続き名 = 名前. (19)

仮パラメータ = ["(" 仮パラメータの並び ")"]. (20)

仮パラメータの並び = 仮パラメータ名の並び ":" 標準型 (21)
{ ";" 仮パラメータ名の並び ":" 標準型 }.

仮パラメータ名の並び = 仮パラメータ名 { "," 仮パラメータ名 }. (22)

仮パラメータ名 = 名前. (23)

複合文 = "begin" 文の並び "end". (24)

文の並び = 文 { ";" 文 }. (25)

文 = 基本文 | (26)
"if" 式 "then" 複合文 "else" 複合文 |
"if" 式 "then" 複合文 |
"while" 式 "do" 文.

基本文 = 代入文 |手続き呼出し文 |入出力文 |複合文. (27)

代入文 = 左辺 ":=" 式. (28)

左辺 = 変数. (29)

変数 = 純変数 | 添字付き変数. (30)

純変数 = 変数名. (31)

添字付き変数 = 変数名 "[" 添字 "]". (32)

添字 = 式. (33)

手続き呼出し文 = 手続き名 ["(" 式の並び ")"]. (34)

式の並び = 式 { "," 式 }. (35)

式 = 単純式 [関係演算子 単純式]. (36)

単純式 = [符号] 項 { 加法演算子 項 }. (37)

項 = 因子 { 乗法演算子 因子 }. (38)

因子 = 変数 |定数 |"(" 式 ")" |"not" 因子. (39)

関係演算子 = "=" | "<>" | "<" | "<=" | ">"| ">=". (40)

加法演算子 = "+" | "-" | "or". (41)

乗法演算子 = "*" | "/" | "div" | "mod" | "and". (42)

入出力文 = "readln" ["(" 変数の並び ")"] | (43)
"writeln" ["(" 式の並び ")"] .

変数の並び = 変数 { "," 変数 }. (44)

定数 = 符号なし整数 | 文字列 | "false" | "true". (45)

符号なし整数 = 数字 { 数字 }. (46)

文字列 = "'" 文字列要素 { 文字列要素 } "'". (47)

文字列要素 = 引用符 (「'」) と行の終り (改行) 以外の任意の文字. (48)

名前 = 英字 { 英字 | 数字 }. (49)

英字 = "a" | "b" | ⋯ | "z" | "A" | "B" | ⋯ | "Z". (50)

数字 = "0" | "1" | ⋯ | "9". (51)

3　副課題1：字句解析器（Lexer）の作成

Pascal 風言語で記述されたプログラムからトークン列を切り出すプログラムを作成しなさい.

3.1　外部仕様

字句解析器への入力は Pascal 風言語で記述されたプログラムであり，出力は字句解析の結果である．字句解析器のメインクラス名は **Lexer**，Jar ファイル名は **Lexer.jar** とすること．Lexer の実行方法は以下の通り.

```
% java -jar Lexer.jar input.pas output.ts
```

なお，input.pas は入力となるプログラムであり，output.ts は出力の字句解析結果である.

3.2　トークンの種類

字句解析器で切り出すトークンの一覧を下記に示す.

ソースコード中の字句	字句解析器が定義したトークン名	ID	ソースコード中の字句	字句解析器が定義したトークン名	ID
and	SAND	0	writeln	SWRITELN	23
array	SARRAY	1	=	SEQUAL	24
begin	SBEGIN	2	<>	SNOTEQUAL	25
boolean	SBOOLEAN	3	<	SLESS	26
char	SCHAR	4	<=	SLESSEQUAL	27
div /	SDIVD	5	>=	SGREATEQUAL	28
do	SDO	6	>	SGREAT	29
else	SELSE	7	+	SPLUS	30
end	SEND	8	-	SMINUS	31
false	SFALSE	9	*	SSTAR	32
if	SIF	10	(	SLPAREN	33
integer	SINTEGER	11	)	SRPAREN	34
mod	SMOD	12	[	SLBRACKET	35
not	SNOT	13	]	SRBRACKET	36
of	SOF	14	;	SSEMICOLON	37
or	SOR	15	:	SCOLON	38
procedure	SPROCEDURE	16	..	SRANGE	39
program	SPROGRAM	17	:=	SASSIGN	40
readln	SREADLN	18	,	SCOMMA	41
then	STHEN	19	.	SDOT	42
true	STRUE	20	名前	SIDENTIFIER	43
var	SVAR	21	符号なし整数	SCONSTANT	44
while	SWHILE	22	文字列（文字定数）	SSTRING	45

Pascal 風プログラムに含まれるトークンは特殊記号，名前，符号なし整数，及び，文字列に大別され，それぞれ次のように定義される.

特殊記号 = "+" | "-" | "*" | "/" | "=" | "<>" | "<" | "<=" | ">" | ">=" | "(" | ")" | "[" | "]" | ":=" | "." | "," | ".." | ":" | ";" | 綴り記号. (52)

綴り記号 = "program" | "var" | "array" | "of" | "procedure" | "begin" | "end" | (53)

"if" | "then" | "else" | "while" | "do" |
"not" | "or" | "div" | "mod" | "and" |
"char" | "integer" | "boolean" | "readln" | "writeln" |
"true" | "false".

名前 = 英字 { 英字 | 数字 }. (54)

英字 = "a" | "b" | … | "z" | "A" | "B" | … | "Z". (55)

数字 = "0" | "1" | … | "9". (56)

符号なし整数 = 数字 { 数字 }. (57)

文字列 = "'"文字列要素 { 文字列要素 } "'". (58)

文字列要素 = 引用符 (「'」) と行の終り (改行) 以外の任意の文字. (59)

なお，英字の大文字，小文字は区別する．また，綴り記号と同じ綴りの名前は許されないものとする（ここで定義した綴り記号のうち，“char”，“integer”，“boolean”，“readln”，“writeln”，“true”，“false” は本来名前と考えるべきものであるが，本演習ではそれらを綴り記号として扱い，通常の名前とは区別する)．

また，プログラムにはトークン（記号）とは別に，トークン同士を区切るための記号分離子が含まれる．

記号分離子 = スペース | タブ | 注釈 | 行の終り. (60)

注釈 = "{" { 注釈要素 } "}". (61)

注釈要素 = 「}」以外の文字 | 行の終り . (62)

記号分離子は 2 つの隣り合うトークン（記号）の間，もしくは，プログラムの最初のトークン（記号）の前に書くことができる．なお，名前，綴り記号，または，符号なし整数が 2 つ続く時には，それらは 1 つ以上の記号分離子によって区切られているものとする．

3.3 字句解析器のテスト

演習用ツールキットのディレクトリ ~/enshud-toolkit-2016/testdata にテストデータ（構文的に正しい Pascal 風プログラム：ファイル名 normal_001.pas ~ normal_013.pas，及び，字句解析器によって得られたトークン列：ファイル名 normal_001.ts ~ normal_013.ts）が用意されているので，利用してもかまわない．なお，testdata に含まれる.ts ファイルの各行内の区切り文字はタブ（␣␣）となっている．たとえば，normal_001.ts の一行目は，下記のようになっている．

“programY␣␣␣␣SPROGRAMY␣␣␣␣17␣␣␣␣4”

3.4 出力例

Lexer への入力例と出力例を下記に示す．このように，プログラム中に現れたトークン，Lexer で定義したトークン，そのトークンの ID，および行番号が出力される．プログラム中のトークンは，出力ファイルにおいて一行に 1 つずつ，出現順に並ぶ.

入力：Pascal 風プログラム

```
program coverage(output);
var     Sum, V : integer;

procedure printData;
begin

...

end.
```

⇓

出力：トークンリスト

```
program    SPROGRAM        17      2
coverage   SIDENTIFIER     43      2
(          SLPAREN         33      2
output     SIDENTIFIER     43      2
)          SRPAREN         34      2
;          SSEMICOLON      37      2
var        SVAR            21      3
Sum        SIDENTIFIER     43      3
,          SCOMMA          41      3
V          SIDENTIFIER     43      3
:          SCOLON          38      3
integer    SINTEGER        11      3
;          SSEMICOLON      37      3
procedure  SPROCEDURE      16      5
printData  SIDENTIFIER     43      5
;          SSEMICOLON      37      5
begin      SBEGIN          2       6

...

end        SEND            8      28
.          SDOT            42     28
```

3.5 レポート

レポートの書き方および提出方法については，1.5 節および 1.6 節に従うこと．長さは表紙を入れて 5 ページ以内とする．また，プログラムリスト等は膨大な量になるので添付しないこと．

副課題 1 動作確認依頼期限: 2016 年 10 月 27 日 (木) 16:10

副課題 1 レポート提出期限: 2016 年 10 月 28 日 (金) 18:00

4　副課題 2：構文解析器（Parser）の作成

Pascal 風言語で記述されたプログラムから切り出されたトークン列から，プログラムが構文的に正しいかどうか判定するプログラムを作成しなさい．

4.1　外部仕様

構文解析器への入力は Pascal 風言語で記述されたプログラム（Pascal 風プログラム）を字句解析した結果得られたトークン列であり，出力は Pascal 風プログラムが構文的に正しければ “OK”，正しくなければ “NG” と正しくないと判定された最初の行番号とする．構文解析器のメインクラス名は **Parser**，Jar ファイル名は **Parser.jar** とすること．Parser の実行方法は以下の通り．

```
% java -jar Parser.jar input.ts
```

Parser.jar の引数は 1 つのみで，構文解析器への入力ファイル名（副課題 1 の出力結果，すなわち構文的な正しさの判定対象となる Pascal 風プログラムのトークン列を保存しているファイル名）を指定する．入力ファイル名の拡張子は ts とし，そうでないファイルが指定された場合，構文解析器はエラーメッセージを出力して停止する．構文解析器は標準エラー出力（java.lang.System.err）に対して出力を行う．なお，入力 ts ファイルに複数の構文エラーがある場合は，最初のエラーを見つけた時点で標準エラーへの出力を行った後に停止して良い．

4.2　出力例

入力：トークンリスト

```
program   SPROGRAM      17    2
coverage  SIDENTIFIER   43    2
(         SLPAREN       33    2
output    SIDENTIFIER   43    2
)         SRPAREN       34    2
;         SSEMICOLON    37    2
var       SVAR          21    3
Sum       SIDENTIFIER   43    3
,         SCOMMA        41    3
V         SIDENTIFIER   43    3
:         SCOLON        38    3
integer   SINTEGER      11    3
;         SSEMICOLON    37    3

...

end       SEND          8     28
.         SDOT          42    28
```

⇓

出力：構文の判定結果

```
OK
```

入力：トークンリスト

```
program   SPROGRAM      17    2
coverage  SIDENTIFIER   43    2
(         SLPAREN       33    2
output    SIDENTIFIER   43    2
)         SRPAREN       34    2
;         SSEMICOLON    37    2
var       SVAR          21    3
Sum       SIDENTIFIER   43    3
,         SCOMMA        41    3
V         SIDENTIFIER   43    3
:         SCOLON        38    3
integer   SINTEGER      11    3
;         SSEMICOLON    37    3
procedure SPROCEDURE    16    5
printData SIDENTIFIER   43    5

...

end       SEND          8     27
.         SDOT          42    27
```

⇓

出力：構文の判定結果

```
NG Line 6
```

4.3　構文定義

2 節に Pascal 風言語の構文定義が EBNF の形式で示されている．この情報を参照すること．

4.4 構文解析器のテスト

作成した構文解析器のテストは各自で行うこと．テストは Pascal 風プログラムが構文的に**正しい場合と正しくない場合の双方について**行うこと．テスト用に作成した構文解析器への入力データ（テストデータ）も評価の対象となるので，各自で作成したテストデータについてレポートで説明すること．各自が作成したテストデータにはオリジナリティを明確にするため，作成者や作成日等をコメントとしてつけておくこと．

~/enshud-toolkit-2016/testdata には，構文的（Syntax）に間違ったテストデータ（synerr_01 ～ synerr_03）が用意されている．このデータも利用してもかまわない．各テストデータのエラー内容は下記のとおり．意味的なエラー（semerr_01 ～ semerr_11）については，構文解析器では特定する必要はない．

- synerr_01 (06 行目): 仮パラメータに配列を使ってはいけない．
- synerr_02 (49 行目): begin が bigen になっている．
- synerr_03 (46 行目): if 文内の複合文の begin が無い．

4.5 レポート

レポートの書き方および提出方法については，1.5 節および 1.6 節に従うこと．長さは表紙を入れて 5 ページ以内とする．また，プログラムリスト等は膨大な量になるので添付しないこと．

副課題 2 動作確認依頼期限: 2016 年 11 月 24 日 (木) 16:10

副課題 2 レポート提出期限: 2016 年 11 月 25 日 (金) 18:00

5　副課題 3：意味解析器（Checker）の作成

Pascal 風言語で記述されたプログラムが構文的に正しいかどうか判定すると共に，変数等の宣言や有効範囲の誤り，演算子と被演算子の間や仮パラメータと実パラメータの間などでの型の不整合といった意味上の誤りを発見するプログラムを作成しなさい．

5.1　外部仕様

意味チェッカへの入力は Pascal 風言語で記述されたプログラム（Pascal 風プログラム）を字句解析した結果得られたトークン列であり，出力は Pascal 風プログラムが構文的および意味的に正しければ “OK”，正しくなければ “NG” と正しくないと判定された最初の行番号とする．意味解析器のメインクラスは **Checker**，Jar ファイル名は **Checker.jar** とすること．Checker の実行方法は以下の通り．

```
% java -jar Checker.jar input.ts
```

Checker.jar の引数は 1 つのみで，意味解析器への入力ファイル名（検査しようとする Pascal 風プログラムのトークン列を保存しているファイル名）を指定する．入力ファイル名の拡張子は ts とし，そうでないファイルが指定された場合，意味解析器はエラーメッセージを出力して停止するものとする．意味解析器は標準エラー出力（java.lang.System.err）に対して出力を行う．

5.2　Pascal 風言語の意味定義

Pascal 風言語では，演算子の優先順位や操作の内容，文の意味などは Pascal で定義されたものと同じである．注意すべき相違として，手続きにおけるパラメータの引き渡し方は**値呼び（call-by-value）**のみであることがあげられる．

以下では，Pascal 風言語の意味定義と，意味上の誤りを発見する上でポイントとなる点を簡単に述べる．

5.2.1　名前

名前はプログラム名，変数名，手続き名，及び，仮パラメータ名を表すのに用いる．名前の長さに制限はないが，先頭 8 文字のみ有効とする．また，アルファベットの大文字と小文字は別の名前として扱う．

5.2.2　型

すべての変数および式は型を持ち，型はそれらがとる値の集合を定める．型には標準型と配列型がある．なお，型定義などによって新たな型を作ることはできない．

標準型は integer 型，char 型，boolean 型の 3 つである．

- integer: -32768 ～ 32767 の整数の集合．
- char: 「’」以外の文字の集合（各文字のコードは演習室の処理系と同様と考えて良い）．
- boolean: 定数名 false と true によって表される真理値の集合．ただし，false < true である．

配列型は決まった個数の要素からなる構造であり，全ての要素は同じ標準型を持つものとする．配列型の変数において，個々の要素を参照するには添字を用いる．各要素は添字付き変数の変数名の後に 1 つの添字を角かっこでくくって示す．添字の型は integer 型のみで，変数宣言においてその「最大値」と「最小値」を整数で定義する．

5.2.3 定数

定数は整数定数，文字定数，文字列定数，及び，boolean 定数のみである，定数宣言などによって新たな定数を作ることはできない．

5.2.4 整数定数

整数定数は integer 型の定数であり，通常の 10 進記法で表される．

5.2.5 文字定数

文字定数は char 型の定数であり，引用符（「'」）ではさまれた 1 つの文字（「'」を除く）で表される．

5.2.6 文字列定数

文字列定数は char 型の配列型の定数であり，引用符（「'」）ではさまれた文字（「'」を除く）の並びで表される．引用符は文字列の一部ではなく，単にそれの区切りを表すにすぎない．長さ 1 の文字列定数と文字定数は同じ表現となるが，どちらを表すかは文脈によって決定される．

5.2.7 boolean 定数

boolean 定数は boolean 型の定数であり，定数名 true または false で表される．

5.2.8 変数

変数は「変数宣言」で宣言された名前と型を持ち，宣言された型の値しかとらない．変数宣言はプログラム，及び，副プログラム（手続き）において行うことができる．プログラム（副プログラム）に現れる変数は必ず変数宣言において宣言しておかなければならない．ただし，同じ名前を持つ変数をプログラム中で重複宣言してはならない．また，同じ名前を持つ変数を同一副プログラムにおいて重複宣言してはならない．

なお，あるプログラム（副プログラム）で宣言された変数は，そのプログラム（副プログラム）が駆動される毎に作り出され，駆動が終了する時点で破棄される．変数は作り出された時点では不定である（宣言された型の値を持たない）．

変数には「純変数」と「添字付き変数」がある．純変数は標準型のデータの記憶場所または配列型のデータの先頭の要素の記憶場所を示し，添字付き変数は配列型のデータの 1 つの要素の記憶場所を示す．添字付き変数が表す要素は，添字を表す式の値に対応する要素である．式の値は添字の型として定義された整数の範囲でなければならない．なお，添字付き変数には変数宣言においてその型が配列型と宣言されたものしか用いることはできない．

5.2.9 式

式は評価時点での値を得るための計算規則を表す．式から得られる値は式中の定数，変数の値，演算子によって決まる．

5.2.10 演算子

被演算子を 1 つしか持たない演算子は，符号（“+”，“−”）と論理演算子の “not” である．符号の被演算子の型は integer，‘not” の被演算子の型は boolean である．

被演算子を 2 つ持つ演算子は，被演算子の型と結果の型から算術演算子（“+”，“−”，“∗”，“/”，“div”，“mod”），論理演算子（“and”，“or”），及び，関係演算子（“=”，“<>”，“<”，“<=”，“>”，“>=”）に分類することができる．

	被演算子の型	結果の型
算術演算子	integer	integer
論理演算子	boolean	boolean
関係演算子	標準型	boolean

各演算子の操作内容（演算内容）や演算子間の順位の規則は一般の Pascal の演算子のそれと同じとする．ただし，演算子 “/” は “div” と同じとする．また，関係演算子の 2 つの被演算子の型は同じでなければならない．

5.2.11 文

文はアルゴリズム上の動作を表すものであり，逐次的に実行される「基本文」，条件付で実行される「if 文」，繰り返し実行される「while 文」からなる．ただし，文にラベルをつけることはできない．

基本文には「代入文」，「手続き呼出し文」，「入出力文」がある．

- 代入文は特殊記号 “:=” の右側の式を評価した結果得られた値で左側の変数の値を置き換える．ただし，変数と式は同じ型でなければならない．また，**代入文に配列型の純変数を指定することはできない．**
- 手続き呼出し文は手続き名が表す手続きを駆動する．実パラメータとして式の並びを持つことができ，手続きの駆動時には値呼びでパラメータが引き渡される．
- 入出力文には readln 文と writeln 文がある．
  - readln 文は integer 型，char 型および char 型の配列型のデータを標準入力から読み込むことができる．読み込まれたデータの代入先は変数の並びで指定する．（変数の並びの左から順に読み込まれたデータが代入される）．変数が integer 型の場合，（符号付きの）整数を表す文字列が読み込まれ，整数値に変換された上で変数に代入される．文字列の先頭には何個かの空白があってもよい．変数が char 型の場合，1 文字だけ読み込まれ変数に代入される．変数が char 型の配列型の場合，その配列の大きさと同じ長さの文字列が読み込まれ，配列の最初の要素から順に 1 文字ずつ代入される．ただし，読み込みの途中で改行された場合，改行まで読み込まれ，改行直前の文字まで配列に代入される．指定された変数への代入が全て終ると読み込み途中の行の残りは（改行を含めて）読み捨てられる．また，単に readln と書かれている場合は入力が 1 行読み捨てられる．
  - writeln 文は integer 型，char 型および char 型の配列型の式の値を標準出力に書き出すことができる．書き出す値は式の並びで指定する．標準出力への書き出しは指定された式の並びの左から順に行われる．特に，char 型の配列型の式（具体的には，char 型の配列型の変数および文字列定数）で指定された場合，その配列の最初の要素から順に最後の要素まで 1 文字ずつ書き出される．指定された値を全て書き出し終ると改行が出力される．また，単に writeln と書かれた場合は改行のみ出力される．

5.2.12　手続き

手続きは名前のついた副プログラムであり，手続き呼出し文によって駆動される．

手続きの定義は，プログラム中の「副プログラム宣言」において行う．手続きの場合，手続き名と仮パラメータの並びを定義する．仮パラメータは変数パラメータのみでその型は標準型である．

仮パラメータと実パラメータの対応は，それぞれの並びにおけるパラメータの位置によってつけられる．対応する仮パラメータと実パラメータは同じ型でなければならない．

なお，手続きが仮パラメータの並びを持たなければ，実パラメータの並びを書いてはならない．

手続きの中から呼び出すことができるのは，それ自身，および，それより前に宣言された手続きである．したがって，自分自身を呼び出す再帰的な実行は可能である．また，手続きで参照できる変数はそれ自身で宣言された変数 (ローカル変数) とプログラムで宣言された変数 (グローバル変数) のみである．

5.2.13　プログラム

プログラムは，記号 “program” に続くプログラム名と「名前の並び」,「変数宣言」,「副プログラム宣言群」, 処理内容を表す「複合文」からなる．特に，変数宣言と副プログラム宣言群をまとめて「ブロック」と呼ぶ．

プログラム内部ではプログラム名は特に意味を持たない．また，名前の並びも現時点では意味がない．変数宣言で定義される変数は副プログラム群で宣言される手続きから参照可能ないわゆるグローバル変数である．

5.2.14　宣言の重複

変数名，手続き名，及び，仮パラメータ名の宣言においては，次のような宣言の重複は許されないものとする．

1. プログラムの宣言において，グローバル変数名，手続き名として同一の名前を重複して宣言する．
2. 手続きの宣言において，仮パラメータ名，及び，ローカル変数名として同一の名前を重複して宣言する．または，宣言中の手続き名と同一の名前を仮パラメータ名，または，ローカル変数名として宣言する．

5.3　エラーメッセージの仕様

入力が構文的にも意味的にも正しい場合には，副課題 1 で作成した構文チェッカと同様，”OK”が出力される．入力が正しくない場合には，誤りとなった行の番号が出力され，その後に誤りの内容を表す**英語のメッセージ**が出力される．

例：`Line  12: syntax error`

以下は，典型的な誤りの例であり，これらの誤りは最低限発見されるものとする．各誤りには必ず個別のエラーメッセージを用意すること．

1. 構文に誤りがある（syntax error）.
2. 参照されている変数が宣言されていない.
3. 重複して宣言されている変数がある.
4. 添字付き変数を表す名前が，変数宣言において配列型の変数名として宣言されていない.
5. 配列型変数を添字付きで利用しなければならない場面で，添字無しで利用している.

6. 配列型の宣言において添字の最小値が添字の最大値より大きな整数となっている．
7. 手続き名が，副プログラム宣言において宣言されていない．
8. 演算子と被演算子の間で型の不整合が発生している．
9. 関係演算子に対し，型の異なる被演算子が用いられている．
10. 代入文の左辺と右辺の式の間で型の不整合が発生している．
11. 手続きの仮パラメータと実パラメータの間で型の不整合が発生している．
12. 添字の型が integer でない．
13. if 文や while 文の式の型が boolean でない．
14. 代入文の左辺に配列型の変数名が指定されている．

エラーメッセージで使用すると思われる典型的な英単語を下記に示す．
添字 (index)，宣言する (declare, define)，演算子 (operator)，被演算子 (operand)，重複した (duplicated)，変数 (variable)，引数 (parameter)，文 (statement)，式 (expression)，条件式 (conditional expression, conditional predicate)，型（type)，副プログラム（procedure）

5.4 意味解析器のテスト

~/enshud-toolkit-2016/testdata には，意味的 (Semantic) に間違ったテストデータが用意されている (semerr_01 ~ semerr_11)．各テストデータのエラー内容は下記のとおり．これら全てのプログラムに正しくエラー出力をすることが出来れば，動作確認はパスしたものとする．当然ながら，構文的なエラー（副課題 2 の内容）も正しく検知すること．

- semerr_01 (35 行目): 使用されている変数 po2 が宣言されていない．
- semerr_02 (11 行目): 変数 p を重複して宣言している．
- semerr_03 (35 行目): 使用されている変数 s は配列変数ではない．
- semerr_04 (10 行目): 言されている配列変数の添字がおかしい．
- semerr_05 (16 行目): 用されている副プログラム smap は宣言されていない．
- semerr_06 (09 行目): 演算子 “+” の被演算子が char 型になっている．
- semerr_07 (19 行目): 関係演算子 “=” において，2 つの被演算子の型が異なる（integer 型と boolean 型)．
- semerr_08 (08 行目): 代入文において，左辺と右辺の式の間で型の不整合が発生している．
- semerr_09 (19 行目): 配列変数 a の添字に true を使っている．
- semerr_10 (21 行目): if 文の条件式が boolean 型でない．
- semerr_11 (21 行目): 代入文の左辺において，配列型の変数名が指定されている [3]．

[3] この場面では「配列変数 dx は添字付き利用されなければならないが添字が無い」という解釈もできるが，「配列型の左辺において配列型の変数名が指定されている」という旨のエラーを出力するようにプログラムを作成することが望ましい．

5.5 レポート

レポートの書き方および提出方法については，1.5 節および 1.6 節に従うこと．長さは表紙を入れて 5 ページ以内とする．また，プログラムリスト等は膨大な量になるので添付しないこと．

副課題 3 動作確認依頼期限: 2016 年 12 月 22 日 (木) 16:10

副課題 3 レポート提出期限: 2016 年 12 月 23 日 (金) 18:00

6　最終課題：コンパイラ（Compiler）の作成

Pascal 風言語で記述されたプログラムをアセンブラ言語 CASL II で記述されたプログラムに翻訳 (変換) するコンパイラを作成しなさい.

6.1　コンパイラの外部仕様

コンパイラへの入力は Pascal 風言語で記述されたプログラム（Pascal 風プログラム）を字句解析した結果得られたトークン列であり，出力はそれをアセンブラ言語 CASL II で記述したプログラム（CASL II プログラム）である．コンパイラのメインクラスは **Compiler**，Jar ファイル名は **Compiler.jar** とすること．Compiler の実行方法は以下の通り.

```
% java -jar Compiler.jar input.ts output.cas
```

Compiler.jar の引数は 2 つである．第一引数では，コンパイラへの入力ファイル名（即ち，翻訳しようとする Pascal 風プログラムのトークン列を保存したファイル名）を指定する．入力ファイル名の拡張子は ts とし，そうでないファイル名が指定された場合，エラーメッセージを出力して停止するものとする.

第二引数では，出力先のファイル名を指定する．出力ファイル名の拡張子は cas とし，そうでないファイル名が指定された場合，エラーメッセージを出力して停止するものとする.

構文的もしくは意味的な誤りを検出した場合は，エラーメッセージを標準エラー出力（java.lang.System.err）に対して出力する．エラーメッセージの仕様は，副課題 3 の意味解析器に準じる．エラー出力を行った場合は，コンパイラは直ちに処理を中止する．なお，第一引数で指定されたファイルが構文的もしくは意味的な誤りを含んでいた場合は，第二引数で指定したファイルを生成してはならない.

6.2　アセンブラ言語 CASL II の仕様

本演習で用いるアセンブラ言語処理系の仕様を別途配布の資料に示す．なお，本演習においては CASL II の入出力装置として UNIX の標準入出力，及び，標準エラー出力が割り当てられているものとする.

本演習で用いる CASL II は 30 種類の機械語命令，4 種類の擬似命令 START，END，DS，DC，及び，2 種類のマクロ命令 IN，OUT から成る．また，これら命令の他に，10 種類のサブルーチンがライブラリとして用意されている．ライブラリの詳細は 6.3 節を確認すること.

6.2.1　実装時に注意すべき仕様

本演習で用いる CASL II は一部の仕様が拡張されているので，下記の点に注意して実装すること.

- 汎用レジスタ GR8 が増設されている．GR8 はスタックポインタ（SP）を兼ねており，GR8 の値を変更することは SP の値を変更することと等価である．また，RPUSH, RPOP 命令の操作対象にもなっている.
- DC, DS 命令は RET 命令と END 命令の間にのみ置くことができる．START 命令と RET 命令の間に置かれた場合はエラーになる.
- ラベルにはスコープが存在する．START 命令と END 命令の間がスコープとなり，その範囲に置かれたラベルが参照可能である．例えば，下記のプログラムでは TARGET1, TARGET2 がそれぞれ異なるスコープで定義されているので，JUMP 命令で参照することができずエラーとなる.

JUMP に失敗する例

```
CASL     START   BEGIN
BEGIN    LAD     GR1,1
         JUMP    TARGET1   ;; 別スコープの TARGET1 に JUMP しようとして失敗する
TARGET2  LAD     GR1,3
         CALL    SUB
         RET
         END               ;; スコープの有効範囲はここまで
SUB      START             ;; ここからは別スコープ
         JUMP    TARGET2   ;; この JUMP も同様に失敗する
TARGET1  LAD     GR1,6
         RET
         END
```

6.3 サブルーチンライブラリ

乗除算やファイルの入出力等の典型的な処理を，サブルーチンライブラリ（~/enshud-toolkit-2016/bin/lib.cas）として用意している．定義されているサブルーチンは以下の通り [4]．

MULT	:	GR1 $*$ GR2 $\rightarrow$ GR2
DIV	:	GR1 / GR2 $\rightarrow$ 商は GR2, 余りは GR1
RDINT	:	入力装置から読み込んだ整数 $\rightarrow$ (GR2)
RDCH	:	入力装置から読み込んだ文字 $\rightarrow$ (GR2)
RDSTR	:	入力装置から読み込んだ文字列（読み込む文字数は GR1 で指定） $\rightarrow$ (GR2)
RDLN	:	入力装置からの文字を改行まで読み飛ばす
WRTINT	:	GR2 を整数として出力装置に書き出す
WRTCH	:	GR2 を文字として出力装置に書き出す
WRTSTR	:	(GR2) から長さ GR1 の文字列を出力装置に書き出す
WRTLN	:	改行を出力装置に書き出す

本サブルーチンライブラリは GR6 および GR7 を使用する．サブルーチンライブラリは，256 語長のバッファ（DS 領域）を必要とする．また，GR6 は 0，GR7 はそのバッファの先頭番地を指すように初期化されていなければならない．例えば，出力する CASL コードは下記のようになっていなければならない．

サブルーチンライブラリを利用した CASL コード

```
CASL     START   BEGIN
BEGIN    LAD     GR6, 0      ;; GR6 の初期化
         LAD     GR7, LIBBUF ;; GR7 の初期化
         :
         :                   ;; 処理本文
         :
         RET
LIBBUF   DS      256         ;; サブルーチンライブラリ用のバッファ領域
         :
         :
         END
```

上記の初期化を除き，サブルーチン以外から GR6, GR7 の値が更新されると誤動作する可能性があるため，GR6, GR7 を使用しないコードを出力すること．サブルーチンのリンクは次のように行う．

[4] 用意されている CASL II のインタプリタでは，WRTINT, WRTCH, WRTSTR によって出力装置に書き出されたデータは，WRTLN によって改行が行われるまでは出力されない仕様になっていることに注意すること．

```
% link.pl program.cas
```

link.pl は演習用ツールキットに含まれているリンカであり，program.cas はコンパイラが出力した CASL II プログラムのファイル名である．リンカの実行にはサブルーチンライブラリの lib.cas を link.pl と同じフォルダに置かなければならない．リンカによって lib.cas の内容が program.cas の末尾に追加される．

なお，**このサブルーチンライブラリの使用は必須ではない**．最終レポートでは，サブルーチンライブラリを用いたか否かを明記の上，用いなかった場合についてはその理由についても述べること．**サブルーチンを用いず，自分なりの工夫をした場合（サブルーチンライブラリの改変も含）は，より高い評価を期待できる**．どのような工夫を行ったのかを必ず最終レポートに明記すること．

6.4 CASL II プログラムの実行

CASL II プログラムへのアセンブル（変換）は次のように行う．

```
% pycasl2 program.cas program.com
```

アセンブルが成功すると program.com というファイルが出力される．これが COMET II シミュレータへの入力ファイルになる．CASL II アセンブラの使い方の詳細については配付資料を参照すること．

COMET II シミュレータのファイル名は pycomet2 で，実行方法は次の通り．

```
% pycomet2 -r program.com
```

program.com は CASL II アセンブラによって得られた機械語コードである．COMET II シミュレータの使い方の詳細については配付資料を参照すること．

6.5 発展的な内容への取り組み

最終課題が完成し，余裕があるものは発展的な内容へ取り組むことを推奨する．言語処理工学 A で学習した内容や，自身でインターネット等を用いて調査を行った内容に基づき発展的な内容をコンパイラに付加することにより，より高い評価を期待できる．以下に発展的な内容の例を挙げる．もちろん他のことについて行なっても構わない．

最適化: 生成する CASL II のコード中に無駄な命令をできるだけ無くすことである．最適化を行なった場合は，「どのような最適化を行なったか」のみを記述するだけでは不十分であり，「その最適化がどの適度効果があったのか」についても記述することが望ましい．例えば，覗き穴最適化を行なった場合は，それにより命令の数がどの程度削減されたのかを評価すべきである（本演習で用いるアセンブラ処理系には，実行ステップ数を記録，出力する機能がある．）．

構造体の導入: 本演習で取り扱う Pascal 風言語で，構造体を定義できるように言語仕様を拡張する．EBNF をどのように変更したのか，構造体を引数として受け渡す時にどのように実現したのか等について述べること．

効率的なテスト: 作成したプログラムが意図したとおりに動作しているのかを確かめるのは大変な作業である．例えば JUnit のような既存のフレームワークを用いて作成したプログラムのテストを効率良く行えるような環境を構築することはソフトウェアの生産性を大きく向上することができる．プログラムのテスト方法を工夫した場合にはどのように工夫したのかについて述べること．なお，既存のフレームワークを用いた場合には，どのフレームワークをどのように用いたのかについても言及すること．

6.6 コンパイラのテスト

本参集課題では，作成されたコンパイラの品質（無欠陥性）を最低限保証するため，TA による最終テストを実施する．そのため，動作確認依頼の期限が早めに設定されているので十分に注意すること．最終テストは以下の要領で実施する．

動作確認依頼の手順

1. 演習用ツールキットのテストデータ用ディレクトリ ~/enshud-toolkit-2016/testdata にあるテストデータ全てに対して正しい結果が得られることを確認する．
2. コンパイラの Jar と zip を生成して提出ディレクトリに設置し，パーミッションを変更する（1.6 節の手順 2〜手順 3 を参照）．
3. 確認依頼システム [5] を通じて TA に連絡し，テストデータの通過率を評価してもらう．通過率は，用意されたテストデータのうち何パーセントに対して正しい結果が得られたかを表す割合である．普通は 100%である．なお，TA からテストの方法について別途連絡があった場合には，それに従うこと．

合格条件

テストデータ用ディレクトリに用意されている全てのテストデータに対して，コンパイラが正常に動作し，コンパイルされた CASL II プログラムの実行結果も正しい場合，最終テストに合格したものとする．

その他

- 最終テストに合格していない人のレポートは受付けない．
- 他人のプログラムの一部もしくは全体をコピーしてテストに合格したということが判明した場合は，単位認定は行わない．
- 担当の TA にテストの依頼をしたのにもかかわらず返答が長期間ない場合は，教員までその旨を伝え指示に従うこと．

受付期限

最終回の一つ前の授業終了時間を受付期限とする．受付期限を過ぎた動作確認依頼には対応しないので注意すること．

最終課題 動作確認依頼期限: 2017 年 01 月 26 日 (木) 16:10

6.7 レポート

レポートの書き方および提出方法については，1.5 節および 1.6 節に従うこと．長さは表紙を入れて 15 ページ以内とする．また，プログラムリスト等は膨大な量になるので添付しないこと．

最終課題 レポート提出期限: 2017 年 02 月 03 日 (金) 18:00

[5] http://sdl.ist.osaka-u.ac.jp/~shinsuke/enshud/2016/checkme/