2009年度 情報科学演習D 言語処理

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．

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．ただし，コンパイラの入力は，Pascal風言語で記述されたプログラムを字句解析して得られたトークン列とする．(字句解析のためのプログラムは予め準備してある．)

1. 期間：2009年10月15日〜2010年1月29日（計16回）
2. プログラミング言語：C言語
3. 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）

ただし，言語処理工学Aの講義内容と進捗に合わせて，次の2つの副課題を設定する．
副課題1　構文チェッカの作成
副課題2　意味チェッカの作成
これら副課題として作成するプログラムへの入力は，最終課題として作成するコンパイラへの入力（Pascal風言語で記述されたプログラムのトークン列）と同じである．また，副課題の仕様の主要な部分は，最終課題の仕様の一部と完全に対応している．従って，ある副課題で作成したプログラムの大部分は，その次の副課題，及び，最終課題において再利用可能である．

1.3 実験実施上の注意

1. 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．
2. 数名の大学院生が，TAとして演習の補助を行う．遠慮なく質問すること．但し，質問はenshu-d@image.med.osaka-u.ac.jp宛にメールを送ること．TA，及び，教員のアドレスは以下の通り．

質問・動作確認依頼用		enshu-d@image.med.osaka-u.ac.jp
教員	中本 将彦	nakamoto@image.med.osaka-u.ac.jp
	肥後 芳樹	higo@ist.osaka-u.ac.jp
TA	宮本 敬三	kz-miymt@ist.osaka-u.ac.jp
	兼光 智子	t-kanemt@ist.osaka-u.ac.jp
	西野 稔	m-nisino@ist.osaka-u.ac.jp
	山田 慎也	s-yamada@ist.osaka-u.ac.jp

3. 連絡は主に (ユーザー名)@exp.ics.es.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.)

4. 演習の期間は，副課題で作成するプログラムを，その後の副課題や最終課題において再利用することを前提として設定されている．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副課題のプログラムを作成すること.

5. 演習用ツールキット (`EnshuD-toolkit-2009.tgz`) を 2009 年度 情報科学演習 D ホームページ
(`http://www.image.med.osaka-u.ac.jp/member/nakamoto/enshuD2009/index.html`) からダウンロードし，次のように各自のホームディレクトリに展開すること.

```
%cd ~
%tar xvfz EnshuD-toolkit-2009.tgz
```

展開すると EnshuD 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.

6. 環境変数 PATH の先頭は，`~/EnshuD/bin` とすること．C シェル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．他のシェルを利用している場合にも，同様の設定を行うこと.
(例) `set PATH =( ~/EnshuD/bin /usr/ucb /usr/bin /bin /usr/bin/X11 .)`

7. 各自のホームディレクトリの下に本実験用のディレクトリ “SPC” を作成し，各課題毎に指定されている必要最小限のファイルのみをその中に入れておくこと．リードパーミション (read permission) および実行パーミッション (execute permission) は開けておくこと．SPC，および，その下位ディレクトリ内から他のディレクトリやファイルにシンボリックリンク等を張ることは禁止する.

8. 作成した，あるいは作成中のプログラムのソースコードは他人にコピーされないように，パーミションをかけたディレクトリに置いておくこと．(具体的には，“chmod 700 directory 名” を実行すればよい．) ゲームなど演習に直接関係のないファイルは置かないこと.

9. yacc 等のコード生成ツールを用いてはならない.(構文解析から CASLII 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.)

1.4 レポートの提出方法

レポートは電子ファイル (PDF 形式) で提出すること．ファイルの提出先は以下のディレクトリである.

- 第 1 回レポート：/home/exp/exppub/expd/report2009/1st/
- 第 2 回レポート：/home/exp/exppub/expd/report2009/2nd/
- 最終レポート：/home/exp/exppub/expd/report2009/final/

(注)

- 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) をアンダースコアでつないだものとすること．
  例) 1001_NAKAMOTO_Masahiko.pdf
- ファイルのパーミションは 444(リードパーミションのみ) に設定すること．
- 元のファイルのコピーを必ず手元に残しておくこと．
- ファイルの更新日時を，レポート提出時刻とみなす．

1.5 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)
3. 全レポートを遅滞なく提出

以上の条件を全て満たすこと．ただし、最終課題のレポート提出以降は出席を免除する．

1.6 スケジュール

1 回	10/15(木) 3,4 限
2 回	10/22(木) 3,4 限
3 回	10/29(木) 3,4 限
4 回	11/5(木) 3,4 限
副課題 1 レポート締切	11/11(水) 18:00
5 回	11/12(木) 3,4 限
6 回	11/19(木) 3,4 限
7 回	11/26(木) 3,4 限
8 回	12/3(木) 3,4 限
9 回	12/10(木) 3,4 限
副課題 2 レポート締切	12/16(水) 18:00
10 回	12/17(木) 3,4 限
11 回	1/7(木) 3,4 限
12 回	1/14(木) 3,4 限
13 回	1/21(木) 3,4 限
14 回	1/22(金) 4 限
15 回	1/28(木) 3,4 限
16 回	1/29(金) 4 限
最終レポート締切	2/4(木) 18:00

2　副課題 1：構文チェッカの作成

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.

2.1　外部仕様

構文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 scanner(附録 3) によって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.

構文チェッカの実行ファイル名は parser で，実行方法は次の通り.

```
%parser  filename.ts
```

parser の引数は 1 つのみで，parser への入力ファイル名（即ち，構文的に正しいかどうかを判定しようとする 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，parser はエラーメッセージを出力して停止するものとする．parser の出力は標準エラー出力 (stderr) に対して行う.

2.2　Pascal 風言語の構文

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.

附録 2 に Pascal 風言語の構文定義を拡張バッカス・ナウア記法 (Extended Backus-Naur Form (EBNF)) を用いて示す．各構文規則は EBNF 式によって超越名（構文要素の意味を表す日本語名）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．句は超越名，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，かっこは式のまとまり (1 回だけの出現) を示す.

2.3 出力例

scanner の出力，すなわち，parser の入力は，以下の例のように，プログラム中に現れたトークン，そのトークン名（定数 SYMBOL の値），enum 型において実際に割り当てられている整数値（附録 1，附録 3），及び，行番号が 1 行に 1 トークンずつ，出現順に並ぶものとする．

parser は入力されたトークン列に対し，それが構文的に正しい場合に “OK” を出力する．構文的に正しくない場合には，“NG” を出力するとともに，誤りが検出された最初の行番号を出力する．

```
-(Pascal 風プログラム)---------------


{ SAMPLE:OK }
program coverage(output);
var     Sum, V : integer;


procedure printData;
    begin


...


end.
-------------------------------------
                    ↓
-(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
;         SSEMICOLON      37      5
begin     SBEGIN          2       6

...

end       SEND            8       28
.         SDOT            42      28
-------------------------------------
                    ↓
-(出力:トークンリスト)--------------
OK
-------------------------------------
```

```
-(Pascal 風プログラム)---------------


{ SAMPLE:NG }
program coverage(output);
var     Sum, V : integer;


procedure printData;
        Sum := 0;


...


end.
-------------------------------------
                    ↓
-(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
;         SSEMICOLON      37      5
Sum       SIDENTIFIER     43      6
:=        SASSIGN         40      6
0         SCONSTANT       44      6
;         SSEMICOLON      37      6


...


end       SEND            8       27
.         SDOT            42      27
-------------------------------------
                    ↓
-(出力)-----------------------------
NG Line 6
-------------------------------------
```

2.4 構文チェッカのテスト

作成した構文チェッカ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チェッカへの入力データ（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．なお，演習用ツールキットのディレクトリ ~/EnshuD/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 001.pas 〜 013.pas，及び，scanner によって得られたトークン列：ファイル名 001.ts 〜 013.ts）が用意されているので，利用してもかまわない．

また，~/EnshuD/subtests には，構文的に，または意味的に間違ったテストデータも用意されている．このデータも利用しても構わない．各テストデータのエラー内容は下記のとおり．

new07: 9 行目で integer 型と char 型の変数で加算を行っている

new08: 19 行目の while 文の条件式で左辺が integer 型，右辺が boolean 型になっている

new10: 6 行目に syntax エラー，仮パラメータに配列を使ってはいけない

new11: 19 行目で配列変数 a の添え字に true を使っている

test01: 49 行目に syntax エラー，begin が bigen になっている

test02: 35 行目で使用されている変数 po2 が宣言されていない

test03: 11 行目で変数ｐを重複して宣言している

test04: 35 行目で使用されている変数 s は配列変数ではない

test05: 10 行目で宣言されている配列変数の添え字がおかしい

test06: 16 行目で使用されている副プログラム smap は宣言されていない

test09: 8 行目の代入文で左辺と右辺の変数の型が一致していない

test12: 21 行目の if 文の条件式が boolean 型でない

test13: 21 行目の配列変数 dx に添え字が無い

test14: 46 行目に syntax エラー，if 文内の複合文の begin が無い

2.5 プログラミングのヒント

- **トークン列の読み込み**

  scanner の出力するトークン列は，C の関数 fscanf() を用いて容易に読み込みが出来るような形式にしてある．(注．ただし，文字列の場合は注意が必要．)

- **各シンボルに割り当てられる値**

  各シンボルに割り当てられる値は，附録 1 の enum 型の定義に準じている．この enum 型の定義は，scanner の lex のソースリスト scanner.l に含まれているので，コピーして利用してよい (附録 3 参照).

2.6 レポート

レポートには，プログラムのファイル名，プログラムの簡単な説明，テストデータについての説明，テスト結果について記述すること．また，課題の曖昧な部分について，自分で判断したことも記入すること．なお，膨大な枚数になるので，プログラムリスト，テストデータ，テスト結果をレポートに添付する必要はない．従って，提出するレポートは表紙を含めて A4 で 5 ページ以内になると思われる．

なお，確認のため，各自の実験用ディレクトリ SPC 中に PARSE というディレクトリを作成し，実行形式のファイル parser と，プログラムのソースファイル，及び，作成したテストデータのみを置くこと．ただし，不正なコピーを避けるため，ソースファイルは他人が読み書き出来ないようにしておくこと (chmod 600 ファイル名 とすればよい)．

レポート提出期限
11 月 11 日 (水) 18:00

3　副課題 2：意味チェッカの作成

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．

3.1　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 scanner(附録 3) によって字句解析した結果得られたトークン列であり，出力は最初に発見された意味上あるいは構文上の誤りを表すエラーメッセージである．

意味チェッカの実行ファイル名は checker で，実行方法は次の通り．

```
%checker  filename.ts
```

checker の引数は 1 つのみで，checker への入力ファイル名（即ち，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，checker はエラーメッセージを出力して停止するものとする．checker の出力は標準エラー出力 (stderr) に対して行う．

3.2　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．

3.2.1　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．

3.2.2　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： -32767〜32767 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(偽を表す) と true(真を表す) 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．

3.2.3 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．

3.2.4 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．

3.2.5 文字定数

文字定数は char 型の定数であり，引用符 (「’」) ではさまれた一つの文字 (「’」を除く) で表される．

3.2.6 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．

3.2.7 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．

3.2.8 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．

3.2.9 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．

3.2.10 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．

3.2.11 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．

入出力文には宣言済みの手続き 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．

3.2.12 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．

3.2.13　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．

3.2.14　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．

3.3　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．各誤りには出来るだけ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．
6. 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．
7. 演算子と被演算子の間で型の不整合が発生している．
8. 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．
9. 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．
10. 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．
11. 添字の型が integer でない．
12. if 文や while 文の式の型が boolean でない．
13. 代入文の左辺に配列型の変数名が指定されている．

3.4 意味チェッカのテスト

意味チェッカの作成が完了したら，TA による動作確認をうけること．正常な動作が確認された場合のみ，レポートを受け付ける．(従って，この動作確認は単位取得のための必要条件である．) 動作確認の手順については，以下の通りである．

まず，SPC の下に CHECK というディレクトリをつくって実行形式のチェッカ checker を置く．この CHECK と checker のパーミッションは他人が実行できるようにしておくこと．次に，自分で動作することを確認した後，動作確認依頼のメールを enshu-d@image.med.osaka-u.ac.jp 宛に出して TA の確認を受けること．

なお，TA の人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．

動作確認の対象となるエラーを含んだ Pascal 風言語のプログラムとそのトークン列は，演習用ツールキットのディレクトリ ~/EnshuD/subtests のものを使用する．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．

3.5 レポート

レポートには，プログラムのファイル名，プログラムの簡単な説明，また，課題の曖昧な部分について，自分で判断したことも記入すること．なお，膨大な枚数になるので，プログラムリスト，テストデータ，テスト結果をレポートに添付する必要はない．従って，提出するレポートは表紙を含めて A4 で 5 ページ以内になると思われる．但し，確認のためディレクトリ SPC/CHECK に，ソースファイルを置いておくこと．但し，これらのソースファイルは他人が読み書き出来ないようにしておくこと (chmod 600 ファイル名)．

レポート提出期限
12 月 16 日 (水) 18:00

4　最終課題：コンパイラの作成

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.

4.1　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 scanner(附録 3) によって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.

コンパイラの実行ファイル名は spc で，実行方法は次の通り.

```
%spc filename.ts
```

spc の引数は 1 つのみで，spc 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.

spc の通常出力（即ち，CASL II プログラム）は入力ファイル名の拡張子を cas に置き換えたファイル（ディレクトリはカレントディレクトリ）に対して行う．すなわち、filename.ts が与えられた場合、カレントディレクトリに filename.cas を出力する．一方，文法 (構文) 誤りや意味誤りを検出した場合は，エラーメッセージの出力を UNIX の標準エラー出力に対して行う．エラーメッセージの仕様は checker に準じる．エラー出力を行った場合，spc は直ちに処理を中止する.（エラー回復処理を行う必要はない.）

4.2　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 CASLII 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，擬似命令やマクロ命令の他に，次の 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.（注意：用意されている CASLII 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様になっている.）

MULT	:	GR1 $*$ GR2 $\rightarrow$ GR2
DIV	:	GR2 / GR1 $\rightarrow$ 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

4.2.1 サブルーチンライブラリ使用上の注意

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 を使用しないようなコードを出力すること．

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．

4.2.2 最適化について

最終課題が完成し，余裕のあるものは最適化を行なうことを推奨する．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なった内容に基づき最適化を行ない，それをレポートに追記することにより，より高い評価を期待できる．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.）.

4.3 CASL II プログラムの実行

spc が出力した CASL II プログラムはサブルーチンのリンクとアセンブルが行われ、得られた機械語コードを COMET II シミュレータによって実行する．

サブルーチンのリンクは次のように行う．

```
%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 spc 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.（lib.cas は演習用ツールキットに含まれている.）リンカによって lib.cas の内容が program.cas の末尾に追加され、下記のサブルーチン初期化コードがプログラムの一行目と置換される．

```
CASL     START   BEGIN
_OUTBUF  DS      256
_BEGIN   LAD     GR6,0
         LAD     GR7,_OUTBUF
```

アセンブルは CASL II アセンブラ pycasl2 を用いて次のように行う．

```
%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 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．

4.4 コンパイラのテスト

作成したコンパイラのテストは各自で行うこと．ただし，それとは別に，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以下の要領で実施する．

1. **期間**
   2010 年 1 月 14 日〜2 月 4 日

2. **方法**
   演習用ツールキットのテストデータ用ディレクトリ “~/EnshuD/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た時に，各自が担当の TA に連絡して，テストデータの通過率 (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，普通は 100%である) を評価してもらう．

   最終テストの依頼は，enshu-d@image.med.osaka-u.ac.jp 宛にメールを送ること．なお，担当の 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．

3. **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．

4. **準備**
   実験用ディレクトリ SPC に，各自が作成したコンパイラの実行ファイル”spc”を置くこと（TA はディレクトリ SPC に置かれた”spc”を使ってテストを行う）．なお，実験用ディレクトリ SPC，及び，実行ファイル spc はテスト実施者が読めるようにしておくこと．ディレクトリ等のアクセスパーミッションが落ちているなどして，テストが実行できない場合には，実施率は 0 と報告される．

5. **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．

4.5 提出物

4.5.1 ソースファイル

ディレクトリ SPC の下に，ソースファイルを置いておくこと．ただし，他人にアクセスされないように設定しておくこと (chmod 600 ファイル名)．

4.5.2 レポート

報告書は次のような要領で作成すること．長さは 20 ページ程度とする．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．

1. 構成をよく考え，章・節に分け，章題，節題を付ける．
2. 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)．また，紙面のレイアウトにも気を配る．図・表を多く挿入し，図・表に番号および題を付け，本文からは番号で参照する．
3. 最初に，この課題を通して何を追求しようとしたのか，何故それをしようとしたのかを書く．また，プログラム・報告書で評価してほしいことも残さず書いておくこと．これらが示されており，その内容に工学的な意義が認められれば，たとえ他の重要事項が不完全であっても，良い評価が与えられる．
4. 課題の曖昧な点，不完全な点をどのように定めたかを書く．
5. プログラムの説明はトップダウン的に行うのが望ましい (5〜7)．つまり，まず行っている処理の原理について説明をする．ここでは，プログラムで用いているデータ構造や変数を持ち出さず，“ 原理 ” の説明にとどめておく．処理の正しさを，読み手に確信させるのを目的とする．(先に原理の説明があって，それから，その原理をプログラム化するときに，どのようなデータ構造やアルゴリズムを導入したのかを説明すべき．)
6. 次に，データ構造やアルゴリズム，また，呼び出しているサブルーチンの働きを簡単に述べ，それらを用いて，原理との対応がよく分かるように，より詳しく処理の内容を説明する．
7. そして，呼び出されている各サブルーチンについて，上記 5，6 を繰り返すことで，徐々に説明を詳細化していく．(ただし，詳細化し過ぎないように注意すること．コンパイラとしての処理が正しく実現されているということを説明するのが目的であり，すべての関数について説明しないといけないということではない．)
8. 目標としたことがどこまでで き，何ができなかったかを明示し，どのようにすればできるかなどの考察を書く．
9. 演習において友人，知人からアドバイスをもらった場合には，誰にどのような点を教えてもらったかを，謝辞として，参考文献の前に書く（これは礼儀です）．
10. 演習において参考にした本などは，末尾に参考文献の一覧表を示し，本文では文献番号などで参照する．
11. この演習に関して，特に感じたこと，悩まされたプログラムのバグ，システムのバグ，課題の改善案などを必ず書くこと．今後の参考にします．
12. その他特に指定のない項目については,「情報工学実験/プログラミング演習に関する注意」(2009 年度版)，及び,「情報科学科演習室利用に関する規則」(2009 年度版) に従うこと．

レポート提出期限
2 月 4 日 (木) 18:00

附録 1　SYMBOL

```
enum SYMBOL {
        SAND,           SARRAY,         SBEGIN,         SBOOLEAN,
/*      and             array           begin           boolean         */

        SCHAR,          SDIVD,
/*      char            div  /          */

        SDO,            SELSE,          SEND,
/*      do              else            end             */

        SFALSE,
/*      false         */

        SIF,            SINTEGER,       SMOD,           SNOT,
/*      if              integer         mod             not             */

        SOF,            SOR,            SPROCEDURE,     SPROGRAM,
/*      of              or              procedure       program         */

        SREADLN,        STHEN,
/*      readln          then            */

        STRUE,          SVAR,
/*      true            var             */

        SWHILE,         SWRITELN,       SEQUAL,         SNOTEQUAL,
/*      while           writeln         =               <>              */

        SLESS,          SLESSEQUAL,     SGREATEQUAL,    SGREAT,
/*      <               <=              >=              >               */

        SPLUS,          SMINUS,         SSTAR,          SLPAREN,
/*      +               -               *               (               */

        SRPAREN,        SLBRACKET,      SRBRACKET,      SSEMICOLON,
/*      )               [               ]               ;               */

        SCOLON,         SRANGE,         SASSIGN,        SCOMMA,
/*      :               ..              :=              ,               */

        SDOT,           SIDENTIFIER,    SCONSTANT,      SSTRING,
/*      .               名前            符号なし整数    文字列（文字定数）*/

        SLBRACE,        SRBRACE,        SSQUOTE,        SPARA,
/*      {               }               '               パラメータ      */

        SDIGIT,         SALPHA,         SEOF,           SNULL,
/*      数字            アルファベット  ファイルの終了  その他          */
```

```
          SERR
/*      エラー                                                          */
      };
          /* コンパイラ内で用いられるトークンです．ほとんどのシンボルは，*/
          /* スキャナから渡されます．ただし，以下のものは，スキャナ内部  */
          /* だけで用いられるものです．  */
          /*      SERR, SALPHA, SDIGIT, SLBRACE, SRBRACE, SSQUOTE */
          /* また，SNULL は，未定義やまだ定まっていないことを示すための */
          /* もので，スキャナの出力にはなりません．SPARA は，パラメータ */
          /* であることを示すためのもので，シンボルテーブルで用います．  */
          /* これも，スキャナの出力にはなりません．  */
```

附録 2　EBNF による Pascal 風言語の構文定義

プログラム = “program” プログラム名 “(” 名前の並び “)” “;”
　ブロック
　複合文 “.”.
プログラム名 = 名前.
名前の並び = 名前 { “,” 名前 }.
ブロック = 変数宣言
　副プログラム宣言群.
変数宣言 = [“var” 変数宣言の並び].
変数宣言の並び = 変数名の並び “:” 型 “;” { 変数名の並び “:” 型 “;” }.
変数名の並び = 変数名 { “,” 変数名 }.
変数名 = 名前.
型 = 標準型 | 配列型.
標準型 = “integer” | “char” | “boolean”.
配列型 = “array” “[” 添字の最小値 “..” 添字の最大値 “]” “of” 標準型.
添字の最小値 = 整数.
添字の最大値 = 整数.
整数 = [符号] 符号なし整数.
符号 = “+” | “-”.
副プログラム宣言群 = { 副プログラム宣言 “;” }.
副プログラム宣言 = 副プログラム頭部
　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 “;”.
手続き名 = 名前.
仮パラメータ = [“(” 仮パラメータの並び “)”].
仮パラメータの並び = 仮パラメータ名の並び “:” 標準型 { “;” 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 }.
仮パラメータ名 = 名前.
複合文 = “begin” 文の並び “end”.
文の並び = 文 { “;” 文 }.
文 = 基本文 |
　“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.
代入文 = 左辺 “:=” 式.
左辺 = 変数.

変数 = 純変数 | 添字付き変数.
純変数 = 変数名.
添字付き変数 = 変数名 “[” 添字 “]”.
添字 = 式.
手続き呼出し文 = 手続き名 [“(” 式の並び “)”].
式の並び = 式 { “,” 式 }.
式 = 単純式 [関係演算子 単純式].
単純式 = [符号] 項 { 加法演算子 項 }.
項 = 因子 { 乗法演算子 因子 }.
因子 = 変数 |
定数 |
“(” 式 “)” |
“not” 因子.
関係演算子 = “=” | “<>” | “<” | “<=” | “>” | “>=”.
加法演算子 = “+” | “−” | “or”.
乗法演算子 = “∗” | “/” | “div” | “mod” | “and”.
符号 = “+” | “-”.
入出力文 = “readln” [“(” 変数の並び “)”] |
“writeln” [“(” 式の並び “)”] .
変数の並び = 変数 { “,” 変数 }.
定数 = 符号なし整数 | 文字列 | “false” | “true”.
符号なし整数 = 数字 { 数字 }.
文字列 = “ ’ ” 文字列要素 { 文字列要素 } “ ’ ”.
文字列要素 = 引用符 (「’」) と行の終り (改行) 以外の任意の文字.
名前 = 英字 { 英字 | 数字 }.
英字 = “a” | “b” | ··· | “z” | “A” | “B” | ··· | “Z”.
数字 = “0” | “1” | ··· | “9”.

附録 3　　字句解析系 scanner

1. プログラム

字句解析系の lex プログラム scanner.l は演習用ツールキットの`scanner_src` の中に置かれている．各自で lex, cc 等を用いてコンパイルし，実行ファイルを作ること．

lex は字句解析系を作るためのツールであり，lex 言語で書かれたプログラムを C 言語のプログラムに変換する．従って，具体的なコンパイルの手順は以下のようになる．

```
%cd ~/EnshuD/scanner_src
%lex scanner.l
%cc -o scanner lex.yy.c -ll  (-ll は lex 用のライブラリをリンクするのに必要)
%mv scanner ../bin           (パスを通している~/EnshuD/bin に移動させる)
```

なお，scanner.l 中，enum 型でトークンを定義している．この部分は課題のプログラムで必要となるので，コピーして利用すること．

2. 機能

Pascal 風言語で記述されたプログラムからトークンを切り出し，各トークンについて出現順に，(1) トークン，(2) そのトークン名（定数 SYMBOL の値），(3)enum 型において実際に割り当てられている整数値（附録 1），及び，(4) 行番号，を 1 行に 1 トークンずつ出力する．

3. 外部仕様

実行方法は次の通り．

```
%scanner filename.pas
```

scanner の引数は 1 つのみで，scanner への入力ファイル名（即ち，翻訳しようとする Pascal 風プログラムのファイル名）を指定する．入力ファイル名の拡張子は pas で，それ以外のファイルが指定された場合，エラーを出力しプログラムは停止する．

scanner の通常出力（即ち，トークン列）は入力ファイル名の拡張子を ts に置き換えたファイル（ディレクトリはカレントディレクトリ）に対して行う．

4. Pascal 風プログラムのトークン

Pascal 風プログラムに含まれるトークンは,「附録 1　SYMBOL」で宣言されている定数 SYMBOL の各値と同じとする．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.

綴り記号　＝　“program”｜“var”｜“array”｜“of”｜
“procedure”｜“begin”｜“end”｜
“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 }.
英字　＝　“a”｜“b”｜…｜“z”｜“A”｜“B”｜…｜“Z”.
数字　＝　“0”｜“1”｜…｜“9”.
符号なし整数　＝　数字 { 数字 }.
文字列　＝　“ ’ ” 文字列要素 { 文字列要素 } “ ’ ”.
文字列要素　＝　引用符 (「’」) と行の終り (改行) 以外の任意の文字.

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．)

プログラムにはトークン（記号）とは別に，トークンどうしを区切るための記号分離子が含まれる．記号分離子は空白，タブ，注釈，及び，プログラムのテキスト表現における行の終りである．なお，注釈は次のように定義される．

注釈　＝　“{” { 注釈要素 } “}”.
注釈要素　＝　「}」以外の文字｜行の終り.

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．

5. 注意

トークン列の例については副課題 1 の説明を参考にすること．

動作確認依頼について

TA は忙しい中を (場合によっては演習時間外にも) 動作確認のために時間を割いてくれている．動作確認依頼の際に不備があると，TA との間に余分なやりとりを増やすことになるので，以下を参考にして，メールを送る前に十分確認すること．

メールを送る前にすべきこと

- プログラムの動作確認
  - 指導書にある仕様を満たしているか?
  - 出力メッセージの内容，出力先は適切か?
- TA が動作確認を実行できる状態になっているか確認
  - 指定された場所にプログラムを置いているか?
  - パーミッションが適切に設定されているか?

  (毎年，多くの人がこれらの確認を怠っている)

メールに書くべき事項

- Subject
- 所属，学籍番号，氏名
- 動作確認対象のプログラムを置いている場所 (アカウント名はここに含まれる)
- TA に対してして欲しいこと
  - 「お願いします」や「すいません」しか書かない人がよくいる．
  - 自動返答ではなく人に読んでもらうメールであることをよく考えること．

動作確認依頼メールの例

```
[ヘッダ]
Subject: 情報科学演習 D 意味チェッカ動作確認依頼
From: xxxxx@exp.ics.es.osaka-u.ac.jp
To:   enshu-d@image.med.osaka-u.ac.jp
Cc:   xxxxx@exp.ics.es.osaka-u.ac.jp

[本文]
所属: 計算機科学コース
氏名: xxxx xxxx
学籍番号: 90xxxxxx

意味チェッカの動作確認をお願いいたします.
意味チェッカは，下記の場所に置いております.
/home/exp/exp0/xxxxxx/(以下省略)
%また，上記ディレクトリの内容を添付しております.
よろしくお願いします.

(署名欄はあってもなくてもいい)
```