情報科学実験Ｂ

担当：松本 真佑 先生
提出者：秋口　敬
コース：計算機科学コース
学籍番号：09B19002

提出年月日：2022 年 1 月 28 日

1　課題の目的

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せよ.

2　課題達成の方針と設計

2.1　方針

配布された 1 つ 1 つの pas ファイルについて処理が増えていくので、その処理について、
1. 配布 cas ファイルを確認し、各行がどの処理を行っているかを調べる。元のプログラム文と cas の分を一対一で対応付けする。
2.EBNF 記法に従って、1. で一対一対応した cas ファイルを出力する。

2.2　設計

cas ファイル作成のために加えた処理は以下の通りである。

1. 変数のメモリの確保および変数とメモリ番号の対応付け
2.cas ファイルの基盤の作成 (pascal プログラム文の処理以外の部分)
3. 副プログラムの基盤の作成
4.if、while 文の処理
5. 変数の処理
6. 配列の処理
7. 関係演算子の処理
8. 符号の処理
9. 加法演算子の処理
10. 乗法演算子の処理
11.”not”の処理
12. 定数の処理
13. 手続き呼び出しの処理
14. 出力文の時の処理
15. 変数を出力するときの処理
16. 文字列の出力

3　プログラムの実装方針

課題 3 で作成した AST を用いて、EBNF 記法に基づいて順に処理を入れていく。AST を用いるため、EBNF の順番と cas ファイルでの処理の順番が異なる場合でも対応しやすい。

3.1　変数のメモリの確保および変数とメモリ番号の対応付け

課題 3 で副プログラムごとに変数のリストを作成している。配列の要素数も保存しているので、それを利用して各変数にメモリの番地を対応付けさせる。また、変数のメモリ確保の cas 文もここで定義する。
この作業により、副プログラムごとに変数名とその番地が一対一対応しているので、5 節や 18 節での変数の処理が容易となっている。

3.2　cas ファイルの基盤の作成 (pascal プログラム文の処理以外の部分)

配布 cas ファイルの pascal プログラム文の処理以外の部分を出力する。1 節や 17 節で定義した変数のメモリの確保や文字列のメモリの確保の文もここで出力する。

3.3　副プログラムの基盤の作成

副プログラムごとにラベル名を”PROC0”や”PROC1”とつける。その後は、仮パラメータの内容をスタックから取り出すなど、配布 cas ファイルの pascal プログラム文の処理以外の部分を出力する。副プログラム名とラベル名を一対一対応しておき、手続き呼び出しされるとそのラベルに JUMP するようにしておく。また、変数の参照のためにどの副プログラムの処理を行っているかも保存しておく。

3.4　if、while 文の処理

if 文の真のときの処理、偽の時の処理および while 文のループの中の処理にラベルをそれぞれ定義する。”LOOP0”や”ENDIF0”などとし、if 文や while 文が出るたびに数字を増やしていく。if 文の中に if 文があるような場合でも再帰的に処理を行うことによって、異なる if 文同士でラベルが混同することがないようにしている。
if 文は、else 節がないときと else 節があるときで処理を分けている。
else 節がない場合、次のようなコードを生成する。

```
;; 条件文の処理
POP GR1
CPA GR1, =#FFFF
JZE ELSE0
;; 条件文が真の時の処理
ELSE0 NOP
```

条件文が偽の時は ELSE0 のラベルに JUMP するので、条件文が真の時の処理は実行されない。条件文が真の時は ELSE0 に JUMP せず、条件文が真の時の処理が行われる。else 節がある場合、次のようなコードを生成する。

```
;; 条件文の処理
POP GR1
```

```
CPA GR1, =#FFFF
JZE ELSE0
;; 条件文が真の時の処理
JUMP ENDIF0
ELSE0 NOP
;; 条件文が偽の時の処理
ENDIF0 NOP
```

条件文が偽の時は ELSE0 のラベルに JUMP するので、条件文が偽の時の処理が行われる。条件文が真の時は ELSE0 に JUMP せず、条件文が真の時の処理が行われる。その後は ENDIF0 に JUMP するため、条件文が偽の時の処理は実行されない。
while 文のコード生成は以下のようになる。

```
LOOP0 NOP
;; 条件文の処理
POP GR1
CPL GR1, =#FFFF
JZE ENDLP0
;; ループの中の処理
JUMP LOOP0
ENDLP0 NOP
```

条件文が真なら、ENDLP0 には JUMP せずにループの中の処理が実行されたのち、LOOP0 に戻る。条件文が偽なら ENDLP0 に JUMP し、ループが終了する。

3.5 変数の処理

対象となる変数について、2 節の変数リストから変数名で検索して変数を保存している番地を得る。その後はその番地を使って変数に対する処理を行えばよい。例えば、VAR+2 番地に対応した変数が代入文の左辺にある場合は以下のような処理を行う。

```
LD GR2, =2
POP GR1
ST GR1, VAR, GR2
```

スタックの一番最新の領域には右辺の式の結果が格納されている。レジスタ 2 に 2 を、レジスタ 1 に右辺の式の結果をポップする。最後に、VAR+GR2 番地にレジスタ 1 の内容を保存している。これは、VAR+2 番地に対応した変数に右辺の式の結果が格納されるのと同義である。

3.6 配列の処理

対象となる配列について、変数リストから変数名で検索して変数を保存しているメモリの先頭の番地を得る。添え字については式の処理を行う。その式の結果+先頭の番地-1 の番地に対象となる配列があるので、その後はその番地を使って配列に対する処理を行えばよい。

3.7　関係演算子の処理

この節からは式の処理について説明する。関係演算子のための新規ラベル (”TRUE0”、”BOTH0”) を用意する。関係演算子が出るたびにこの数字を増やしていく。レジスタ 1、2 に比較する値を入れて比較し、結果が真なら 0000、偽なら FFFF をレジスタ 1 に格納する。

3.8　符号の処理

符号が”-”なら、そのあとの項の処理を行ったのち、符号の処理を行う。スタックの最新の場所に項の処理結果が格納されているので、それを 0 から引けばよい。

3.9　加法演算子の処理

レジスタ 1 に前の項の結果、レジスタ 2 に後の項の結果を入れておく。その後はレジスタ 1、2 に対して ADDA、SUBA、OR を行う。

3.10　乗法演算子の処理

レジスタ 1 に前の項の結果、レジスタ 2 に後の項の結果を入れておく。その後はレジスタ 1、2 に対して MULT、DIV、AND などを行う。

3.11　”not”の処理

GR1 に因子の結果を入れておき、FFFF と XOR をとる。

3.12　定数の処理

整数および boolean 型の時は直接値をスタックする。文字列の時は直接スタックできないので、いったん GR1 に格納したのちにスタックする。

3.13　手続き呼び出しの処理

3 節で手続き名からラベル名を取得しその番地に JUMP するようにする。仮パラメータについては、一つ一つに式の処理をした後に結果をスタックしていく。この処理は仮パラメータの順番とは逆に行っていく。なぜなら、一番先にスタックしたものは一番後に出てくるので、一番先に式の結果を入れたものは最後の仮パラメータとして取り出されるためである。
以下の図では、2 つの仮パラメータに順に 20、40 を入れるときのスタック領域および変数の領域のようすを示している。40、20 の順にスタックしたのち、パラメータ 1 の内容としてポップ、パラメータ 2 の内容としてポップと順に行う。これにより、元のプログラムから正しく仮パラメータを引き継ぐことができる。

3.14　出力文の時の処理

式一つ一つに対して 15、16 節で出力の処理を行い、最後にサブルーチン WRTLN で改行を出力する。

3.15　変数を出力するときの処理

5 節の変数の処理をしたのち、サブルーチン WRTINT で出力する。

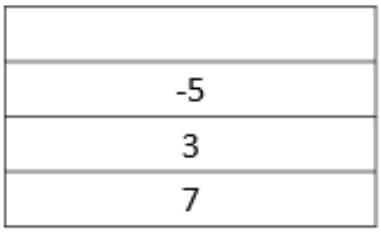
3.16　文字列の出力の処理

文字列の宣言を定義する。この時ラベルを”CHAR0”などとし、定数として文字列を出力するたびにこの数字を増やしていく。次に、サブルーチン WRTSTR を用いるために文字列の長さを取得する。文字列のラベルと長さを用いて、サブルーチン WRTSTR を用いて出力を行う。

3.17　式の処理

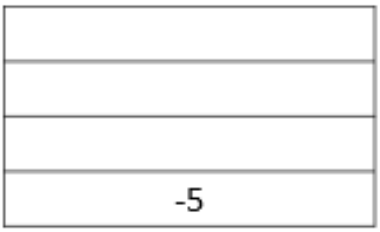
5 節から 12 節で式の処理の細かな部分を説明してきたが、ここでは式の例を用いてスタック領域、レジスタの様子を図に示す。
式が”-5<=3+7”のとき、-5、3、7 がスタックに PUSH されたのち、3 と 7 がレジスタ 1、2 にそれぞれ格納される。3+7 が計算されてレジスタ 1 に結果 (10) が格納される。その結果はスタックに保存される。次に-5 と 10 がレジスタ 1、2 に格納されたあと、比較演算子の処理を行う。結果 (#0000) がレジスタ 1 に格納された後、スタックに保存される。

スタック領域



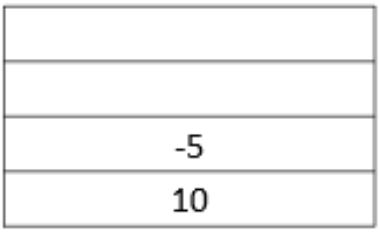
GR1 ?　GR2 ?

スタック領域



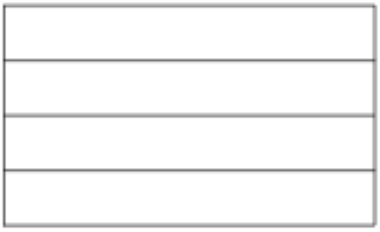
GR1 3　GR2 7

スタック領域



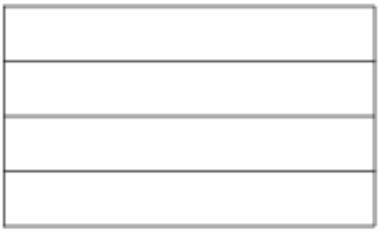
GR1 10　GR2 3

スタック領域



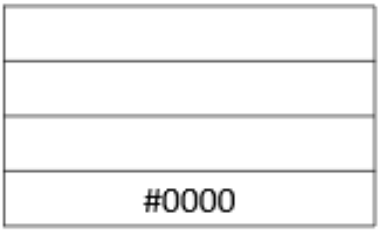
GR1 -5　GR2 10

スタック領域



GR1 #0000　GR2 10

スタック領域



GR1 #0000　GR2 10

4　まとめ

代入文では右辺の処理の後に左辺の処理を行ったり、符号の処理は直後の項の処理の後に行ったりなど、EBNF 記法とは処理が逆になる場合もあったが AST を用いることで楽に処理することができた。なぜなら、代入文の頂点の下に右辺の木と左辺の木を構成しているので、右辺からの処理は容易に行えるからである。
Normal01 から順にテストケース通していく方針をとったが、先の pascal ファイルも確認しながらコーディングを行うことで、書き換えることが少なく済んだ。例えば配列の処理を行うときに、配列の添え字が数字のパターンだけでなく、初めから式であるときも想定して組むなどの工夫を行った。

5　感想、謝辞

今回の課題は、作業自体の難易度は高くなかったが、処理がとても多いと感じた。また、seap で学科の出来具合が一目でわかるので、焦りを感じながら作業を行っていた。