情報工学実験 C（ソフトウェア）報告書

コンパイラ）

学生番号: 09429533
提出者: 高島和嗣

提出日: 2020 年 月日（）
締切日: 2020 年 月日（）

1 実験の目的

- yacc，lex というプログラムジェネレータを使ってプログラムを作成する．
- コンパイラを作成することでプログラム言語で書かれたプログラムとアセンブリ言語との対応について深く理解する．
- 木構造の取り扱いを理解し，木構造を用いて C 言語をアセンブリ言語に変換するコード生成のプログラムを作成する．

2 作成した言語の定義

<プログラム> ::= <変数宣言部> <文集合>
<変数宣言部> ::= <宣言文> <変数宣言部> | <宣言文>
<宣言文> ::= define <識別子>; | array <識別子> [<数>];
<文集合> ::= <文> <文集合>| <文>
<文> ::= <代入文> | <ループ文> | <条件分岐文>
<代入文> ::= <識別子> = <算術式>; | <識別子> [<数>] = <算術式>;
<算術式> ::= <算術式> + <項> | <算術式> - <項> | <項>
<項> ::= <項> * <因子> | <項> / <因子> | <因子>
<因子> ::= <変数> | (<算術式>)
<変数> ::= <識別子> | <数> | <識別子> [<数>]
<ループ文> ::= while (<条件式>) { <文集合> }
<条件分岐文> ::= if (<条件式>) { <文集合> }
| if (<条件式>) { <文集合> } {<else 文>} { <文集合> }
| if (<条件式>) { <文集合> } {<else if 文>} { <文集合> }
| if (<条件式>) { <文集合> } {<else if 文>} {<else 文>} { <文集合> }
<else 文> ::= else { <文集合> }
<else if 文> ::= else if (<条件式>) { <文集合> }
| else if (<条件式>) { <文集合> } {<else if 文>}
<条件式> ::= <算術式> == <算術式> | <算術式> < <算術式> | <算術式> > <算術式>
<識別子> ::= <英字> <英数字列> | <英字>
<英数字列> ::= <英数字> <英数字列>| <英数字>

<英数字> ::= <英字> | <数字>
<数> ::= <数字> <数> | <数字>
<英字> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|A|B|C|D|E|F|G|H|I|J|K
|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<数字> ::= 0|1|2|3|4|5|6|7|8|9

3 定義した言語で受理されるプログラムの例

- 代入文

```
define a;
define b;
define c;

a=0;
b=0;
c=0;
```

- 算術式

```
define a;
define b;
define c;
define d;

a=1;
b=2;
c=3;
d=4
a=a+1;
b=a*c-b;
c=d/a+1;
d=c%2;
```

- if 文

```
define a;
define b;
define c;
define d;

a=3;
b=5;

if(a==2){
    b=b+1;
}
```

```
else if(a<4){
    b=10;
}
else{
    b=0;
}
```

- ループ文

```
define a;
define b;
define c;
define i;

a=0;
b=0;
c=0;
i=0;
while(i<6){
    a=b*2+c;
    i=i+1;
}
```

4 コード生成の概略

4.1 メモリの使い方

今回作成したプログラムは先にデータセグメントを確保し，獲得したデータセグメントに変数領域を記述し，次にテキストセグメントを生成した．

4.2 レジスタの使い方

スタックを用いずにレジスタで計算を行い，レジスタを指定するときは t*(*は 0 ～ 9) とし, *が 9 より大きくなった時は警告文を端末に表示するようにした．

4.3 算術式のコード生成の方法

項が変数の場合は lim 命令でレジスタに値を入れ，変数の場合は li 命令で変数のアドレスをレジスタに入れ，lw 命令でそのアドレスに保存されている値を取り出して，別のレジスタに入れ計算を行う．算術式の最初に左辺のアドレスを$t0 レジスタに入れておき，計算後に結果を左辺の変数のアドレスに保存する．

5　工夫した点

算術式では，項が変数なら変数 a に実数なら変数 b に入れ，a,b を引数とした算術式のコードを生成するようにしていたが，変数と変数，実数と実数の場合に上手くいかず，項が変数，実数の順でも順序がレジスタの引数が逆のコードが生成されてしまった．そこで，regcheck とい名前の関数を実装し，変数と変数，実数と実数の場合でも a,b に適切な引数を入れれるようにし，変数，実数の順の時は順序を入れ替えるようにした．

if 文の実装の際には，抽象構文木に if のみの場合と if と else のみの場合と if と elseif のみの場合と if と elseif と else の場合に分けて実装した．コードの場合によっては条件式を満たし，処理を行った後に適切なラベルにジャンプ出来ないことがあった．そこで if 文の処理であらかじめコードに関係しないラベル$D0 を if 文の処理の最後に設置しておき，条件式を満たしその処理を実行し終わった時は$D0 にジャンプするようにした．

6　ソースプログラムのある場所

/home/users/ecs/09429533/term3-4/compiler/lesson4/ast

7　最終課題を解くために書いたプログラムの概要

- 最終課題 1

```
define i;
define sum;

sum = 0;
i = 1;
while(i < 11) {
   sum = sum + i;
   i = i + 1;
}
```

- 最終課題 2

```
define i;
define fact;

fact = 1;
i = 1;
while(i < 6) {
   fact = fact * i;
   i = i + 1;
}
```

- 最終課題 3

```
 define fizz;
  define buzz;
  define fizzbuzz;
  define others;
  define i;

  fizz = 0;
  buzz = 0;
  fizzbuzz = 0;
  others = 0;
  i = 1;
  while(i < 31){
    if ((i / 15) * 15 == i){
      fizzbuzz = fizzbuzz + 1;
    }
    else if ((i / 3) * 3 == i){
      fizz = fizz + 1;
    }
    else if ((i / 5) * 5 == i){
      buzz = buzz + 1;
    }else{
      others = others + 1;
    }

    i = i + 1;
  }
```

- 最終課題 3 (mod を用いた場合)

```
define fizz;
define buzz;
define fizzbuzz;
define others;
define i;

fizz = 0;
buzz = 0;
fizzbuzz = 0;
others = 0;
i = 1;
while(i < 31){
    if (i % 15 == 0){
        fizzbuzz = fizzbuzz + 1;
    }
    else if (i % 3 == 0){
        fizz = fizz + 1;
```

```
    }
    else if (i % 5 == 0){
        buzz = buzz + 1;
    }else{
        others = others + 1;
    }
    i = i + 1;
}
```

8 最終課題の実行結果

- 最終課題 1

```
Stop at 0xfffffffc
327 instructions

 > print 0x10004000 0x10004001
0x10004000 (268451840) = 0x0000000b (11)
0x10004004 (268451844) = 0x00000037 (55)
0x10004008 (268451848) = 0x00000000 (0)
0x1000400c (268451852) = 0x00000000 (0)
0x10004010 (268451856) = 0x00000000 (0)
0x10004014 (268451860) = 0x00000000 (0)
0x10004018 (268451864) = 0x00000000 (0)
0x1000401c (268451868) = 0x00000000 (0)
0x10004020 (268451872) = 0x00000000 (0)
0x10004024 (268451876) = 0x00000000 (0)
0x10004028 (268451880) = 0x00000000 (0)
0x1000402c (268451884) = 0x00000000 (0)
0x10004030 (268451888) = 0x00000000 (0)
0x10004034 (268451892) = 0x00000000 (0)
0x10004038 (268451896) = 0x00000000 (0)
0x1000403c (268451900) = 0x00000000 (0)
```

- 最終課題 2

```
Stop at 0xfffffffc
182 instructions

 > print 0x10004000 0x10004001
0x10004000 (268451840) = 0x00000006 (6)
0x10004004 (268451844) = 0x00000078 (120)
0x10004008 (268451848) = 0x00000000 (0)
0x1000400c (268451852) = 0x00000000 (0)
0x10004010 (268451856) = 0x00000000 (0)
0x10004014 (268451860) = 0x00000000 (0)
```

```
0x10004018 (268451864) = 0x00000000 (0)
0x1000401c (268451868) = 0x00000000 (0)
0x10004020 (268451872) = 0x00000000 (0)
0x10004024 (268451876) = 0x00000000 (0)
0x10004028 (268451880) = 0x00000000 (0)
0x1000402c (268451884) = 0x00000000 (0)
0x10004030 (268451888) = 0x00000000 (0)
0x10004034 (268451892) = 0x00000000 (0)
0x10004038 (268451896) = 0x00000000 (0)
0x1000403c (268451900) = 0x00000000 (0)
```

- 最終課題 3

```
Stop at 0xfffffffc
2127 instructions

 > print 0x10004000 0x10004001
0x10004000 (268451840) = 0x00000008 (8)
0x10004004 (268451844) = 0x00000004 (4)
0x10004008 (268451848) = 0x00000002 (2)
0x1000400c (268451852) = 0x00000010 (16)
0x10004010 (268451856) = 0x0000001f (31)
0x10004014 (268451860) = 0x00000000 (0)
0x10004018 (268451864) = 0x00000000 (0)
0x1000401c (268451868) = 0x00000000 (0)
0x10004020 (268451872) = 0x00000000 (0)
0x10004024 (268451876) = 0x00000000 (0)
0x10004028 (268451880) = 0x00000000 (0)
0x1000402c (268451884) = 0x00000000 (0)
0x10004030 (268451888) = 0x00000000 (0)
0x10004034 (268451892) = 0x00000000 (0)
0x10004038 (268451896) = 0x00000000 (0)
0x1000403c (268451900) = 0x00000000 (0)
```

- 最終課題 3(mod を用いた場合)

```
Stop at 0xfffffffc
1659 instructions

 > print 0x10004000 0x10004001
0x10004000 (268451840) = 0x00000008 (8)
0x10004004 (268451844) = 0x00000004 (4)
0x10004008 (268451848) = 0x00000002 (2)
0x1000400c (268451852) = 0x00000010 (16)
0x10004010 (268451856) = 0x0000001f (31)
0x10004014 (268451860) = 0x00000000 (0)
0x10004018 (268451864) = 0x00000000 (0)
```

```
0x1000401c (268451868) = 0x00000000 (0)
0x10004020 (268451872) = 0x00000000 (0)
0x10004024 (268451876) = 0x00000000 (0)
0x10004028 (268451880) = 0x00000000 (0)
0x1000402c (268451884) = 0x00000000 (0)
0x10004030 (268451888) = 0x00000000 (0)
0x10004034 (268451892) = 0x00000000 (0)
0x10004038 (268451896) = 0x00000000 (0)
0x1000403c (268451900) = 0x00000000 (0)
```

9 考察

テキストセグメント生成後に，コード末尾にデータセグメントを確保する場合は，テキストセグメントの部分とデータセグメントの部分とで書き込むファイルを分け，最後に 2 つのファイルを合体させる方法をとる．今回作成したプログラムは，最終課題のコンパイルは通ったが，ソケットを用いていないため，算術式の右辺の変数の数は 3 つまでしか計算出来ない．最終課題終了後に確認して気づいたため，プログラムを変更しなかったが，ソケットを用いれば数によらずに計算できるので，今後直すことがあれば算術式にはソケットを用いてようと思った．また，算術式にソケットを用いている理由についても実装してみて改めて深く理解することができた．また，`if` の中に `if` 文を入れたりジャンプするラベルの設定を変える必要がある．今回の実装では，`if` 文の中に `if` 文が入ることを想定せずに作成し，条件式を満たした後にジャンプする処理は `AST_ELSE` と `AST_ELSEIF` の処理の始めに書き込んでいるため，`if` 文の中に `if` 文が入る場合はジャンプの処理が書き込まれない．今回作成したプログラムでは `if` 文の条件を満たした時に実施する処理を `printTree` 関数で探索している途中で，その兄弟である `else if` 文や `else` 文も探索するような抽象構文木を作成しているため，適切な場所でジャンプ処理を行うために今回のように実装したが，`if` 文の中に `if` 文が入るならばジャンプ処理を `IF` の処理の中に書く必要がある．実装するならば，抽象構文木を見直す可能性もあると思い，時間もなかったため今回は最終課題 3 までに対応したプログラムのままである．