中間レポート 1

学籍番号 1029-25-7819　　名前 五十嵐　雄

1　はじめに

本実験の実装は Haskell で行っている。不明な点があれば以下に挙げる資料を中心に参照せよ。

- https://www.haskell.org/documentation (リファレンス集)
- https://www.haskell.org/hoogle/ (Haskell 用の検索エンジン)

2　課題 1

2.1　回答コード

```
int array[8];

void initialize(int *array);
void sort(int *array);

int main(){
  int i;
  initialize(array);
  sort(array);
  for(i = 0; i < 8; i = i + 1) print(array[i]);
}

void initialize(int *array){
  int i;
  for(i = 0; i < 8; i = i + 1){
    array[i] = 8 - i;
  }
}
```

```
void sort(int *array){
  int i, j, tmp;
  for(i = 0; i < 7; i = i + 1){
    for(j = i+1; j < 8; j = j + 1){
      if(array[i] > array[j]){
        tmp = array[i];
        array[i] = array[j];
        array[j] = tmp;
      }
    }
  }
}
```

2.2 プログラムの説明

initialize

配列の先頭のポインタを受け取って、配列の先頭から降順の数値で埋める。

sort

配列の先頭のポインタを受け取って、バブルソートを行う。

3 課題 5

3.1 課題の回答

抽象構文木のデータ構造は src/AST.hs 構文解析器は src/Parser.hs をそれぞれ参照せよ。

3.2 AST の設計方針

以下にデータ構造を列挙して説明していく。説明の過程で実験資料の BNF で使われている用語が登場するので適宜参照せよ。また SourcePos はソース名と位置情報を持った組み込みのデータ型であり、全てのノードが持っている。詳細は Text.Parsec.Pos.hs (公式ライブラリ) を参照せよ。

3.2.1 型シノニム

よく現れる型に対してエイリアスを張っている。このような型シノニムを適切に定義することで型注釈がより実際の動作を反映したものとなりプログラムが読みやすくなっていく。未知の型も登場するがこのあと説明するのでまずは先へ進むこと。

```
type Program    = [EDecl]
type Identifier = String
type DeclList   = [(Type, DirectDecl)] -
```

- Program プログラム全体は EDecl(external-declaration) のリストで表現される。
- Identifier 識別子は文字列で表現される。
- DeclList 変数宣言は型と direct-declaration の組によって表現される。

3.2.2 EDecl

external-declaration に対応するデータ構造である。グローバル変数宣言 (declaration)、プロトタイプ宣言 (function-prototype)、関数定義 (function-definition) の 3 つの役割を果たす。以下にそれぞれのノードについて解説する。

```
data EDecl = Decl          SourcePos DeclList
           | FuncPrototype SourcePos Type Identifier [(Type, Identifier)]
           | FuncDef       SourcePos Type Identifier [(Type, Identifier)] Stmt
```

- Decl SourcePos DeclList
  グローバルな変数宣言についてのノードであり具体的な内容は DeclList によって表現される。型宣言は先頭の一つだけだが、そのあとに続く変数は整数とポインタで別々の型を取りうるので、各変数と型はこの時点で一対一に対応させている。

- FuncPrototype SourcePos Type Identifier [(Type, Identifier)]
  プロトタイプ宣言についてのノードである。左から返り値の型、関数名、引数リストである。引数リストは型と識別名だけ分かれば良いためそれらを組にしたリストとしている。

- FuncDef SourcePos Type Identifier [(Type, Identifier)] Stmt
  関数定義についてのノードである。プロトタイプ宣言と同様のデータ構造に加えて関数本体を Stmt として持っている。実際は BNF の定義により compound-stmt だけが来るようになっている。

3.2.3 Type

SmallC の型に対応するデータ型。定義は以下のようである。

```
data Type = CPointer Type
          | CInt
          | CVoid
```

現状の文法では CPointer CInt, CInt, CVoid しか現れないことになっているが、CPointer の引数を自分自身とすることで、char 型やポインタのポインタといった拡張にも自然に対応することができる。

3.2.4 DirectDecl

direct-declaration に対応するノードである。型情報はさらに上のレベルで付加されるのでこのノードには必要ない。コードは以下である。

```
data DirectDecl = Variable SourcePos Identifier
                | Sequence SourcePos Identifier Integer
```

Variable は変数で識別名だけを持っている。Sequence は配列で識別名と配列サイズを持っている。

3.2.5 Stmt

statement に対応するノードである。定義は以下のようになる。

```
data Stmt = EmptyStmt    SourcePos
          | ExprStmt     SourcePos Expr
          | DeclStmt     SourcePos DeclList
          | CompoundStmt SourcePos [Stmt]
          | IfStmt       SourcePos Expr Stmt Stmt
          | WhileStmt    SourcePos Expr Stmt
          | ReturnStmt   SourcePos Expr
```

以下に各コンストラクタを説明していく。

- EmptyStmt SourcePos
  空の文 (;) に対応する。
- ExprStmt SourcePos Expr
  expression の文に対応する。
- DeclStmt SourcePos DeclList
  compound-statement の中に登場する変数宣言文を表現するノード。compound-statement 以外には現れない。
- CompoundStmt SourcePos [Stmt]
  compound-statement に対応するノード。中カッコで囲まれた部分に現れる複数の statement をリスト形式で保持している。
- IfStmt SourcePos Expr Stmt Stmt
  if 文に対応するノード。左から条件節、True 節、False 節を保持している。else のない if 文は False 節を EmptyStmt にすることで実現している。

- WhileStmt SourcePos Expr Stmt
  while 文対応するノード。左から条件節と本体である。
- returnStmt SourcePos Expr
  return 文に対応するノード。与えられた Expr を保持する

3.2.6 Expr

expression に対応するノードである。BNF 上では結合の強さによって文法要素が入れ子状に表現されているが、データ構造では同じレベルに置いて実装している。正しい構文要素が現れていることは構文解析の実装や意味解析によって担保していくためこの段階では大雑把な定義になっている。実際の定義は以下のようになる。

```
data Expr = AssignExpr   SourcePos Expr Expr
          | UnaryPrim    SourcePos String Expr
          | BinaryPrim   SourcePos String Expr Expr
          | ArrayAccess  SourcePos Expr Expr
          | ApplyFunc    SourcePos String [Expr]
          | MultiExpr    SourcePos [Expr]
          | Constant     SourcePos Integer
          | IdentExpr    SourcePos Identifier
```

以下に各コンストラクタを説明していく。

- AssignExpr SourcePos Expr Expr
  代入文 (assign-expr) に対応するノードである。左の Expr がディスティネーションで右の Expr がソースである。
- UnaryPrim SourcePos String Expr
  単項演算子の適用に対応するノードである。String に演算子、Expr に適用する項を保持している。
- BinaryPrim SourcePos String Expr Expr
  二項演算子の適用に対応するノードである。String に演算子、残りはそれぞれ左オペランドと右オペランドに対応している。
- ArrayAccess SourcePos Expr Expr
  配列アクセスに対応したノードである。左の Expr がソースに対応し、右の Expr が添字に対応する。ソース部分に ArrayAccess が現れても良いので文法上は多重配列アクセスも可能になっている。
- ApplyFunc SourcePos String [Expr]
  関数適用に対応したノードである。String は適用先の関数名で [Expr] が引数リストになっている。

- Constant SourcePos Integer
  定数に対応したノード。単純に符号付き数を保持している。
- IdentExpr SourcePos Identifier
  変数に対応したノード。識別子を文字列で保持している。

3.3 Parser の設計方針

標準ライブラリの Text.Parsec の力により BNF をそのまま書き写すだけでほぼ実装が完了するため、関数を逐一説明するようなことはしない。全体の大まかな流れと工夫が必要であった部分について Parser.hs を参照しながら説明していく。

3.3.1 全体の流れ

- 25 行目　文法の定義
  ライブラリが提供するデータ構造に沿って、予約語と演算子を定義する。
- 31 - 64 行目　lexer
  上で定義したデータ型から lexer セットを生成して、必要になる lexer を取り出す。
- 73 - 140 行目　external-declaration のパーサー
  external-declaration の構文をパースする部分。externalDecl で変数宣言、プロトタイプ宣言、関数宣言の 3 つに分岐して各部分をパースする。DeclList を生成する部分で少し工夫があるので後述する。
- 142 - 196 行目　statement のパーサー
  statement の構文をパースする部分。for 文を while 文に変換している以外は特に特別なことはしていない。
- 198 - 234 行目　expression のパーサー
  expression の構文をパースする部分。logical-OR-exor から unary-expr までの優先度付き演算子で表現されている部分は buildExpressionParser を利用して簡潔に書いている。演算子テーブルは 248 行目から定義されているリストである。

3.3.2 工夫した部分

- 宣言におけるポインタの処理
  例えば $inta, *b$ というような宣言を考える。BNF 上ではまず int と $a, *b$ に分解されてしまうのだが、意味の上では int はそれぞれの変数にかからなければいけない。そこで各変数がポインタであるか否かの情報を回収し、genDecl(88 行目) によって変数ごとに型を生成して DeclList としている。pointerOp(245 行目) によってポインタ演算子をパースし、checkPointer(242 行目) でそれを受け取って型の変換を行っている。

- 演算子の処理
  logical-OR-expr から unary-expr までの構文木は項と演算子で構成されており、構造が非常に似ている。そこで構文木のデータ型を抽象化し演算子と項を受け取る形式にした。Haskell では文字列レベルでのパターンマッチも可能なので後半のコード量は変わらず、かつ Parser のコード量を大きく削減できる。

4 課題 6

パーサーに対するテストが test/ParserSpec.hs にいくらか定義されており、パース元のプログラムとパース結果の抽象構文木の対応を見ることができるのでそれを回答とする。
また文法的に正しくないプログラムを入力した場合はパース結果が $LeftParseError$ となり位置や正しい文法要素が報告されるようになっている。これは Parsec ライブラリの仕様である。

5 課題 7

回答コードは src/ProgramGenerator.hs である。実装については木を素直に辿りながら適切な文字列を生成すればよいだけなので特筆すべき点はない。プログラムの再生成自体を各データ型の Show インスタンスとしてしまうことも考えたが、結果は構文木のままで表示されたほうがデバッグしやすいため採用しなかった。

6 感想

Haskell で Parser は書き慣れていたのでやるだけ感が強かったが、コンビネータなども調べながら楽しくコーディングできた。進捗に余裕が出てきているので最適化を頑張りたい。