レポート2(インタプリタ)

1029289895 尾崎翔太

2018/7/5

1　はじめに

ソースコード単位で解説していくのではなく、課題単位で解説していく。また、説明はできるだけその時点における説明にしようと思うので、ソースコードとの整合性が取れない場合がある。例えば、ソースコードにおいてLTExprはCmpExprになっているが、説明ではLTExprが登場する。最後に、課題には存在しなかったが、デバッグのしやすさから減算と、比較のイコールを実装している（ただし、減算は-と第二オペランドの間に空白を入れないときちんと動作しない)。

2　各課題の説明

2.1　Exercise 3.2.1

テストプログラムとその実行は以下である。

```
# if (3 * 4) < 15 then 1 + 2 * 3 else 0;;
val - = 7
```

これによりif文、加算、乗算、比較、その優先順位が正しく動作していることがわかる。また、問題文にあるようにiiを2、iiiを3、ivを4に束縛した環境を大域環境にすると

```
# iv + iii * ii;;
val - = 10
```

となる。

2.2　Exercise 3.2.2

2.2.1　設計方針

read-eval-printループ内でエラーを検出して、エラーの種類に応じてエラーメッセージを表示したあとにread-eval-printループに戻る。

2.2.2 実装の詳細

main.ml の 7〜10 行目に print_error_and_go 関数を定義している。これは string を受け取って、それを表示したあとに read_eval_print 関数を呼び出す。これは、エラー処理をまとめたものである。そして main.ml の 12 行目に try があって、47 行目に with があって、エラーをパターンマッチして、表示するメッセージを print_error_and_go 関数にわたす。エラーの種類は eval で発生するもの、parser で発生するもの、lexer で発生するもの、正体不明の四つに分類した。

2.3 Exercise 3.2.3

2.3.1 設計方針

単純に & & と || を認識できるようにして、それらに対応する文法規則を追加する。評価においては、第一オペランドを評価した時点で値が確定するなら第二オペランドは評価しない。そのため、通常の二項演算子とは異なる関数が必要となる。

2.3.2 実装の詳細

lexer.mll の 38 行目は「& & 」を AND というトークンに、39 行目は「||」を OR をいうトークンに変換する規則である。そして、優先順位を考慮して、LTExpr の上に AndExpr、AndExpr の上に OrExpr というようになっている。評価に関しては、他の二項演算子と違って、オペランドを評価したりしなかったりするので、区別するために BinLogicOp という新たな型を作った。この型のコンストラクタは And と Or の二つである。そして、eval.ml の 64〜77 行目に apply_logic_prim 関数を定義している。これは、BinLogicOp と exval と exp を引数にとって、exval を返す。op が And で arg1 が BoolV false なら exp2 は評価せずに BoolV false を返す。op が And で arg1 が true なら exp2 の評価結果を返す。op が Or の場合も同様である (true と false は逆だが)。なお、オペランドが bool の値をもってなければエラーを吐く。また、eval_exp 関数に BinLogicOp の場合を追加して、第一オペランドを評価してから apply_logic_prim 関数を呼び出している。最後に、apply_logic_prim 関数と eval_exp 関数は相互再帰の形になったので、and でつないでいる。

2.4 Exercise 3.2.4

2.4.1 設計方針

lexer.mll に新たなルールを追加して、再帰的に呼び出せるようにする。ただし、コメントのネストがきちんとできるようにコメントの深さを管理する。

2.4.2 実装の詳細

lexer.mll の 57 行目以降に comment というルールを追加した。このルールは整数を一つ受け取る。この整数はコメントの深さを表していて、この「(∗」 が現れるたびに 1 増え、「∗) 」が現れたびに 1 減る。main で「(∗ 」