主専攻実験[S-8] 関数プログラミング

課題 4-1

情報科学類 202113564 三村潤之介

課題 4-1

・parse 関数とこれまでに作成した eval 関数を組み合わせなさい。

parse 関数と eval 関数を組み合わせた run は、str 型のデータを受け取って、exp 型のデータを返す関数である。parse 関数によって str から生成・出力された exp を前回までに作成した eval 関数によって評価すれば良い。

すでに Makefile などでは、コンパイルするファイルとして、インタープリタ用の eval.ml ファイルが用意されているため、ここに eval 関数を書き込んだ。この eval 関数は、1 つの exp のみを引数として受け取るものが望ましいため、課題 3 で作成したような、環境も引数とするものではなく、課題 2 で作成したものを採用した。

ただし、eval 関数だけをコピーすると、eval 関数内の exp 表現が読み込めないため、エラーとなってしまう。ここで、課題で作成した exp 定義や value 定義を持ってくるのではなく、syntax.ml の定義を使用させる。こうすることで、main.ml 内で共通した exp データを Syntax.exp として扱えるようになる。

以下に実行例を示す。

Main.run "1+2*3";;

- : Syntax.value = Syntax.IntVal 7

run 関数によって、”1+2*3”という文字列が評価され、7 と出力されていることが分かる。

・syntax.ml に記述された exp 型の式を拡張するためには、lexer.mll と parser.mly をどう変更すればよいか。例として、2 つの数が等しくないことを意味する演算子「<>」を追加してみよ。

exp 表現としては、2 つの exp をもつ「Neq」とし、「<>」がトークン「NEQUAL」として表現されるとした。

まず、syntax.ml にて、exp の定義として、Neq を以下のように追加した。

```
 type exp =
…
 | Neq of exp * exp     (* e <> e *)
```

次に、lexer.mll にトークンを追加した。

```
rule token = parse
…
  | "<>"     { NEQUAL }
```

次に、parser.mly に使うトークンの登録、及び exp 表現への変換を記述した。

```
…
%token NEQUAL   // "<>"
…
exp:
…
  // e1 != e2
  | exp NEQUAL exp
    { Neq ($1, $3)}
```

以上の記述をした上で、コンパイルした。以下に実行例を示す。

Main.parse "1<>2";;

- : Syntax.exp = Syntax.Neq (Syntax.IntLit 1, Syntax.IntLit 2)

parse によって、構文解析が行われ、「1 と 2 が異なる」ということを表す exp 表現が出力されていることが分かる。ただし、eval 関数に「<>」に関する記述をしていないため、この exp 文を評価することはできない。