Part 2 – documentation

In our bison program we declared tokens which we get from our lexer.

For example:

```
%token <str> INT FLOAT VOID
```

where <str> defines the type of ID which is derived from our %union implementation.

```
%union {
    char *str;
    ParserNode *node;
}
```

We also declared the associativity of selected tokens to represent the correctness of the code we parse, in addition we declared the precedence of the terminals by making the most important one be the one at the bottom of the list.

```
%left <str> COMMA
%right <str> ASSIGN
%left <str> OR
%left <str> AND
%left <str> RELOP
%left <str> ADDOP
%left <str> MULOP
%right <str> LPAREN RPAREN NOT
%left <str> LBRACE RBRACE
```

In our grammar rules we build the tree in the following manner:

```
EXP : EXP ADDOP EXP { concatList($1, makeNode("addop", $2, NULL));
                     concatList($1, $3);
                     $$ = makeNode("EXP", NULL, $1);}
      | EXP MULOP EXP { concatList($1, makeNode("mulop", $2, NULL));
                       concatList($1, $3);
                       $$ = makeNode("EXP", NULL, $1);}
      | LPAREN EXP RPAREN { ParserNode * lparen = makeNode("(", $1, NULL);
                           concatList(lparen, $2);
                           concatList(lparen, makeNode(")", $3, NULL));
                           $$ = makeNode("EXP", NULL, lparen);}
      | LPAREN TYPE RPAREN EXP { ParserNode * lparen = makeNode("(", $1, NULL);
                                concatList(lparen, $2);
                                concatList(lparen, makeNode(")", $3, NULL));
                                concatList(lparen, $4);
                                $$ = makeNode("EXP", NULL, lparen);}
      | ID { $$ = makeNode("EXP", NULL, makeNode("id", $1, NULL));}
      | NUM { $$ = makeNode("EXP", NULL, $1);}
      | CALL { $$ = makeNode("EXP", NULL, $1);}
;
```

$$ and symbols are always node type while the terminals are str type. For terminals we create a corresponding node and then we concatenate the symbols/terminals to each other (they become siblings).

Then we connect the siblings to be the children of the parent node which is the symbols on the left side of the rule.

After parsing the whole code file, we print the tree using the provided helper function.