# Relational Operators

# Relational Operators

```
%union {
  long val;
  char * string;
  struct ExprRes * ExprRes;
  struct InstrSeq * InstrSeq;
  //struct BExprRes * BExprRes;
}

%type <string> Id
%type <ExprRes> Factor
%type <ExprRes> Term
%type <ExprRes> Expr
%type <InstrSeq> StmtSeq
%type <InstrSeq> Stmt
%type <ExprRes> BExpr
```

# Relational Operators

```
Stmt    : IF '(' BExpr ')' '{' StmtSeq '}'        {$$ = doIf($3, $6);};
BExpr   : Expr EQ Expr                            {$$ = doEq($1, $3);};
```

# Relational Operators

```
extern struct ExprRes * doEq(struct ExprRes * Res1,  struct ExprRes * Res2) {
  struct ExprRes * Res;
   int reg = AvailTmpReg();
  AppendSeq(Res1->Instrs, Res2->Instrs);
  Res = (struct ExprRes *) malloc(sizeof(struct ExprRes));

  AppendSeq(Res1->Instrs, GenInstr(NULL, "seq", TmpRegName(reg), TmpRegName(Res1->Reg),
                                   TmpRegName(Res2->Reg)));
   Res->Reg = reg;
  Res->Instrs = Res1->Instrs;
  ReleaseTmpReg(Res1->Reg);
    ReleaseTmpReg(Res2->Reg);
  free(Res1);
  free(Res2);
  return Res;
}
```

# Relational Operators

```
extern struct InstrSeq * doIf(struct ExprRes * Res, struct InstrSeq * seq) {
  struct InstrSeq * seq2;
 char * label = GenLabel();
 AppendSeq(Res->Instrs, GenInstr(NULL, "beq", "$zero", TmpRegName(Res->Reg), label));
  seq2 = AppendSeq(Res->Instrs, seq);
  AppendSeq(seq2, GenInstr(label, NULL, NULL, NULL, NULL));
  free(Res);
  return seq2;
}
```