HOW TO RUN:

1. make
2. ./compiler file.ada
3. ./javaa file.jasm
4. java file

修改的地方：

因為上一份的程式作業沒有完成，所以這次的作業跟上一次的作業有很大部分的不同，
以下更改的部分是我先完成第二份程式作業後實作第三份程式作業時更改到的地方

1. lex.l
   在lex.l增加#define PLINE，用來輸出Java assembly language的註解

```
#define PLINE    {buf2 += yytext;}
...
<COMMENT>\n |
\n                    {
                      LIST;
                      fout << "/*  " << linenum << ":" << buf2 << "  */" << endl;
                      if(Opt_T) cout << linenum << ":" << buf;
                      ++linenum;
                      buf[0] = '\0';
                      buf2 = "";
                      }
```

2.parser.y
(a) 加入string filename和ofstream fout 來讀檔名和輸出Java assembly language

```
string filename;
ofstream fout;
```

(b) 加入bool declaring 來判斷constant是宣告還是已宣告正被assign

```
bool declaring = false;
```

© int main 中讀取檔案

```
    string fin = string(argv[1]);
    filename = fin.substr(0, fin.find_last_of("."));
    fout.open(filename + ".jasm");
```

(d)不同的位置加入code generation，例如：

```
simple_statements:  PRINT
                    {
                        CG_StatePrintBegin();
                    }
                    expression semi
                    {
                        Trace("print expr");
                        CG_StatePrintEnd(true, *$3);
                    }
                    |
                    PRINTLN
                    {
                        CG_StatePrintBegin();
                    }
                    expression semi
                    {
                        Trace("println expr");
                        CG_StatePrintEnd(false, *$3);
                    }
```

(e) 加入codegeneration.hpp 和 codegeneration.cpp，用來產生Java assembly language並輸出到指定檔案

(f) symbol.cpp 在運算的expression中新增一function算出並回傳expression運算的結果

```
Vartype *exprArith(Vartype expr1, Vartype expr2, char opera, int kind, int type)
{
  Vartype* new_expr = new Vartype();
  float val1;
  float val2;
  float sum_real;
  int sum_int;
  switch(expr1.type)
  {
    case TYPE_INT:
      val1 = expr1.var_val.int_val;
      break;
    case TYPE_REAL:
      val1 = expr1.var_val.real_val;
      break;
  }
  switch(expr2.type)
  {
    case TYPE_INT:
      val2 = expr2.var_val.int_val;
      break;
    case TYPE_REAL:
      val2 = expr2.var_val.real_val;
      break;
  }

  switch(opera)
  {
    case '+': // +
    sum_real = val1 + val2;
    break;
    case '-': // -
    sum_real = val1 - val2;
    break;
    case '*': // *
    sum_real = val1 * val2;
    break;
    case '/': // /
    sum_real = val1 / val2;
    break;
  }
    if(expr1.type == TYPE_REAL || expr2.type == TYPE_REAL)
  {
    new_expr->var_val.real_val = sum_real;
  }
  else
  {
    sum_int = sum_real;
    new_expr->var_val.int_val = sum_int;
  }
  //cout << new_expr->var_val.int_val << endl;
  new_expr->var_kind = kind;
  new_expr->type = type;
  new_expr->declar = true;

  return new_expr;
}
```