Flexes.l

```
/* Flex tokens */
%option noyywrap nodefault yylineno
%{
#include <string.h>
#include "flexes.h"
#include "flexes.tab.h"
%}

%%
[ \t\n]+                      ; /* Whitespace, ignore */
"%".*                         ; /* comment, ignore */
"question"                    return TQUESTION;
"rule"                        return TRULE;
"action"                      return TACTION;

"input"                       return TINPUT;
"if"                          return TIF;
"and"                         return TAND;
"or"                          return TOR;
"then"                        return TTHEN;
"not"                         return TNOT;
"do"                          return TDO;
"ask"                         return TASK;
"because"                     return TBECAUSE;
"write"                       return TWRITE;
"nl"                          return TNL;
"run"                         return TEND;// Not a keyword
"becomes"                     return TBECOMES;
"name"                        return TNAME;
"number"                      return TINUMBER;
"integer"                     return TIINTEGER;
"group"                       return TGROUP;
"choose from"                 return TCHOOSE;

">"                           yylval.fn = 1; return CMP;
"<"                           yylval.fn = 2; return CMP;
">="                          yylval.fn = 3; return CMP;
"<="                          yylval.fn = 4; return CMP;
"="                           yylval.fn = 5; return CMP;
"is not"                      yylval.fn = 6; return CMP;
"is"                          yylval.fn = 7; return CMP;

"("                           return TLPAREN;
")"                           return TRPAREN;

"."                           return TSTOP;
";"                           return TQEND;
","                           return TCOMMA;

[0-9]+\.[0-9]*                yylval.d = atof(yytext); return TNUMBER;
[0-9]+                        yylval.d = atof(yytext); return TNUMBER;
[a-zA-Z_][a-zA-Z0-9_]*        yylval.s = lookup(yytext); return TIDENTIFIER;

'(\\.|''|[^'\n])*'  |
\"(\\.|\"\"|[^"\n])*\"        yylval.s = lookup(yytext); return TSTRING;
.                             printf("Unknown token!\n"); yyterminate();
```

Flexes.y

```
%{
# include <stdio.h>
# include <stdlib.h>
# include "flexes.h"
%}

%union {
      struct ast *a;
      double d;
      struct symbol *s;
      int fn;
}

/* The tokens */
%token <s> TIDENTIFIER TSTRING TNAME TINUMBER TIINTEGER
%token <d> TNUMBER
%token <token> TBECOMES TNOT TGROUP
%token <token> TLPAREN TRPAREN TCOMMA TNL TCHOOSE
%token <token> TIF TRULE TQUESTION TACTION TINPUT TSTOP TQEND
%token <token> TAND TOR TTHEN TASK TBECAUSE TDO TWRITE TEND
%token <token> TPLUS TMINUS TMUL TDIV

%type <s> ident
%type <a> flexes expr input comp comps
%type <a> program programs script stmts question_block
%type <a> rule question action
%type <a> group groups group_choices

%nonassoc <fn> CMP

%left TPLUS TMINUS
%left TMUL TDIV

%start flexes

%%

/* Variables */
ident : TIDENTIFIER                       { /*$$ = variable($1);*/ }
      ;

/* Comparisons */
comp : ident CMP ident              { $$ = newcmp($2, $1, $3); }
     | ident CMP TSTRING            { $$ = newcmp($2, $1, $3); }
     | ident CMP TNUMBER            { $$ = newcmp($2, $1, $<s>3); }
     | TNUMBER CMP TNUMBER          { $$ = newcmp($2, $<s>1, $<s>3); }
     ;

comps : comp                        { }
      | comps TAND comp             { }
      | comps TOR comp              { }
      ;

/* Expressions, such as value1 becomes value2, etc */
expr : TAND expr                   { }
     | TIF comps TTHEN expr         { $$ = flow('i', $2, $4); }
     | ident TBECOMES ident         { $$ = newassign($1, $3); }
     | ident TBECOMES TSTRING       { $$ = newassign($1, $3); }
     | ident TBECOMES TNUMBER      { $$ = newassign($1, $<s>3); }
     | TEND                         {  }
     | TNL                          {  }
     | TASK ident                   {  }
     | TLPAREN expr TRPAREN               {  }
     | TWRITE TLPAREN ident TRPAREN       { $$ = dowrite($3); }
     | TWRITE TLPAREN TSTRING TRPAREN    { $$ = dowrite($3); }
     ;

stmts : expr                       { $$ = newast('s', $1, NULL); }
      | stmts expr                 { $$ = newast('S', $1, $2); }
      ;

/* Action block */
action : TACTION ident stmts TSTOP { $$ = function('a', $2, $3); }
       ;

/* Rule block */
rule : TRULE ident stmts TSTOP     { $$ = function('r', $2, $3); }
     ;

input : TINPUT TNAME               { $$ = newast('i', $<a>2, NULL); }
      | TINPUT TINUMBER            { $$ = newast('i', $<a>2, NULL); }
      | TINPUT TIINTEGER           { $$ = newast('i', $<a>2, NULL); }
      | TINPUT ident               { $$ = newast('i', $<a>2, NULL); }
      | TCHOOSE ident              { }
      ;

question_block : TSTRING TQEND input TQEND TBECAUSE TSTRING { $$ =
question_block($1,$3,$6); }
               | TSTRING TQEND input TBECAUSE TSTRING      { $$ =
question_block($1,$3,$5); }
               | TSTRING TQEND input                        { $$ =
question_block($1,$3, NULL); }
               ;

question : TQUESTION ident question_block TSTOP      { $$ = function('q', $2, $3); }
         ;

group_choices : ident                          { }
              | group_choices TCOMMA ident     { }
            ;

group : TGROUP ident group_choices TSTOP       { }
      ;

groups : group                                 { }
       | groups group                          { }
       ;

program : rule                           { $$ = newast('p', $1, NULL); }
        | question                       { $$ = newast('p', $1, NULL); }
        ;

programs : program                       { $$ = newast('p', $1, NULL); }
         | programs program              { $$ = newast('p', $1, $2); }
         ;

script: programs action                        { $$ = newast('p', $1, $2); }
      | groups programs action                 { $$ = newast('p', $1, $2); }
      ;

flexes: script                           { $$ = $1; return eval($1); }
        ;

%%
```

Flexes.h

```
/*
 * Thomas Cowell - 05017742
 * Univeristy of Glamorgan
 *
 * flexes.h
 * Header file for the Abstract Syntax Trees (AST)'s
 *
 * + - * /
 * 0 - 7 comparison ops
 * L expression or statement list
 * I IF statement
 * N symbol ref
 * B assignment (becomes)
 * S list of symbols
 * C rule/question/action
 * P input
 * U because (question optional answer)
 * D do (do something)
 */
#define VARNAME_SIZE          20
#define VARVALUE_SIZE         100
#include <string.h>


extern int yylineno;
void yyerror(char *s, ...);

struct symbol {         /* variable name */
      char *name[VARNAME_SIZE];
      double d_value;
      char *c_value[VARVALUE_SIZE];
      int isdouble;
      struct ast *func; /* stmt for the function */
      struct symlist *syms;
};

/* symtable of fixed size */
#define NHASH 9997
struct symbol symtab[NHASH];
struct symbol *lookup(char*);

struct symlist {
      struct symbol *sym;
      struct symlist *next;
};

struct symlist *newsymlist(struct symbol *sym, struct symlist *next);
void symlistfree(struct symlist *sl);


/* Node Types
 * r : Rule
 * q : Question
 * i : if statement
 * = : assignment
 * e : expression
 */

struct ast {
      int nodetype;
      struct ast *l;
      struct ast *r;
};

struct s_flow {     /* If - the if-then in flex has no else clause.*/
      int nodetype;           /* Type i */
      struct ast *cond; /* The condition */
      struct ast *tl;         /* Then branch */
};
```

```
struct ucall {     /* Stores a question, rule or action */
      int nodetype;          /* Question or Rule */
      struct symbol *s; /* Name, code block */
};

struct assign {
      int nodetype;          /* Assignment (becomes) */
      struct symbol *s1;
      struct symbol *s2;
};

struct numval {
      int nodetype;          /* Number */
      double number;
};

struct s_compare {
  int cmptype;
  struct symbol *l;
  struct symbol *r;
};

struct s_variable {
  int nodetype;
  struct symbol *var;
};

struct s_ref {
      int nodetype;
      struct symbol *s;
};

struct s_rule {
  int nodetype;
  struct symbol *name;
  struct ast *stmts;
};

struct s_question {
  int nodetype;
  struct symbol *question;
  struct ast *input;
  struct symbol *because;
};

struct s_dowrite {
  int nodetype;
  struct symbol *sentence;
};

struct s_function {
      int nodetype;
      struct symbol *name;
      struct ast *statements;
};

/* Build an AST */
struct ast *newast(int nodetype, struct ast *l, struct ast *r);
struct ast *newcmp(int cmptype, struct symbol *l, struct symbol *r);
struct ast *newassign(struct symbol *s1, struct symbol *s2);
struct ast *num(double d);
struct ast *flow(int nodetype, struct ast *cond, struct ast *l);

struct ast *function(int nodetype, struct symbol *name, struct ast *statements);
struct ast *question_block(struct symbol *question, struct ast *input, struct symbol
*because);
```

```
struct ast *dowrite(struct symbol *sentence);
struct ast *sentence(struct symbol *s);
struct ast *variable(struct symbol *var);

/* Evaulate an AST */
double eval(struct ast *);

/* Delete and free up memory from an AST */
void treefree(struct ast *);

extern int yylineno;
void yyerror(char *s, ...);
```

FlexesFuncs.c

```
/* This file should contain the contents of all the AST's that are
 * declared in flexes.h.
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <math.h>
#include "flexes.h"
#include "flexes.tab.h"

/* Symbol table */
static unsigned
symhash(char *sym)
{
  unsigned int hash = 0;
  unsigned c;

  while (c = *sym++) hash = hash*9 ^ c;

  return hash;
}

struct symbol *
lookup(char *sym)
{
/*
  struct symbol *sp = &symtab[symhash(sym)%NHASH];
  int scount = NHASH;

  while (--scount >= 0) {
    if (sp->name[VARNAME_SIZE] && !strcmp(sp->name[VARNAME_SIZE], sym)) { return
sp; }

    if (!sp->name) {
      sp->name[VARNAME_SIZE] = strdup(sym);
      sp->d_value = 0;
      sp->c_value[VARVALUE_SIZE] = strdup(sym);
      sp->func = NULL;
      return sp;
    }

    if (++sp >= symtab+NHASH) sp = symtab;
  }

  yyerror("symbol table overflow\n");
  abort();
*/
}

struct ast *
newast(int nodetype, struct ast *l, struct ast *r)
{
  struct ast *a = malloc(sizeof(struct ast));

  printf("Firing newast.\n");

  if (!a) {
    yyerror("Out of memory.");
    exit(0);
  }

  a->nodetype = nodetype;
  a->l = l;
  a->r = r;
  return a;
}

struct ast *
num(double d)
{
  struct numval *a = malloc(sizeof(struct numval));

  if (!a) {
    yyerror("Out of memory.");
    exit(0);
  }

  a->nodetype = 'K';
  a->number = d;
  return (struct ast *)a;
}

struct ast *
newcmp(int cmptype, struct symbol *l, struct symbol *r)
{
  struct s_compare *a = malloc(sizeof(struct s_compare));

  if (!a) {
    yyerror("Out of memory.");
    exit(0);
  }

  a->cmptype = '0' + cmptype;
  a->l = l;
  a->r = r;
  return (struct ast *)a;
}

struct ast *
newassign(struct symbol *s1, struct symbol *s2)
{
  struct assign *a = malloc(sizeof(struct assign));

  if (!a) {
    yyerror("Out of memory.");
    exit(0);
  }
  a->nodetype = 'B';  /* becomes */
  a->s1 = s1;
  a->s2 = s2;
  return (struct ast *)a;
}

struct ast *
variable(struct symbol *var)
{

      printf("Attemting to create a variable.\n");

  struct s_variable *a = malloc(sizeof(struct s_variable));

  if (!a) {
    yyerror("Out of memory.");
    exit(0);
  }

  a->nodetype = 'N';
  a->var = var;
  return (struct ast *)a;
}

struct ast *
flow(int nodetype, struct ast* cond, struct ast *tl)
{
  struct s_flow *a = malloc(sizeof(struct s_flow));

  if (!a) {
    yyerror("Out of memory.");
    exit(0);
  }
  a->nodetype = nodetype;
  a->cond = cond;
  a->tl = tl;
  return (struct ast *)a;
}

struct ast *
rule(struct symbol *name, struct ast *stmts)
{
  printf("Firing rule\n");
  struct s_rule *a = malloc(sizeof(struct s_rule));

  if (!a) {
    yyerror("Out of memory.");
    exit(0);
  }

  a->nodetype = 'R';
  a->name = name;
  a->stmts = stmts;

  return (struct ast *)a;
}

struct ast *
question_block(struct symbol *question, struct ast *input, struct symbol *because)
{
  struct s_question *a = malloc(sizeof(struct s_question));

  if (!a) {
    yyerror("Out of memory.");
    exit(0);
  }

  a->nodetype = 'b';
  a->question = question;
  a->input = input;
  a->because = because;

  return (struct ast *)a;
}

struct ast *
function(int nodetype, struct symbol *name, struct ast *statements)
{
      if (nodetype == 'q') printf("Firing question.\n");
      if (nodetype == 'r') printf("Firing rule.\n");

      struct s_function *a = malloc(sizeof(struct s_function));

      if (!a) {
            yyerror("Out of memory.");
            exit(0);
      }
      a->nodetype = nodetype;
      a->name = name;
      a->statements = statements;

      return (struct ast *)a;
}

struct ast *
dowrite(struct symbol *sentence)
{
  struct s_dowrite *a = malloc(sizeof(struct s_dowrite));

  if (!a) {
    yyerror("Out of memory.");
    exit(0);
  }

  a->nodetype = 'W';
  a->sentence = sentence;

  return (struct ast *)a;
}

struct ast *
sentence(struct symbol *sentence)
{
      struct s_ref *a = malloc(sizeof(struct s_ref));

      if (!a) {
            yyerror("Out of memory.");
            exit(0);
      }

      a->nodetype = 's';
      a->s = sentence;

      return (struct ast *)a;
}

/* Free a tree of AST's */
void treefree(struct ast *a)
{
  switch(a->nodetype) {
    /* two subtrees */
    case '+':
    case '-':
    case '*':
    case '/':
    case '1': case '2': case '3': case '4': case '5': case '6':
    case 'L':
      treefree(a->r);

    /* no subtree */
    case 'K': case 'N':
      break;

    case 'B':
      free( ((struct assign *)a)->s2);
      break;

    /* upto three subtrees */
    case 'I':
      free( ((struct s_flow *)a)->cond);
      if (((struct s_flow *)a)->tl) treefree(((struct s_flow *)a)->tl);
      break;
    default: printf("internal error: free bad node %c\n", a->nodetype);
      break;
  }

  free(a); /* always free the node itself */
}
/*
struct symlist *
newsymlist(struct symbol *sym, struct symlist *next)
{
      struct symlist *sl = malloc(sizeof(struct symlist));

      if (!sl) {
            yyerror("Out of space.");
            exit(0);
      }

      sl->sym = sym;
      sl->next = next;
      return sl;
}
*/
void
symlistfree(struct symlist *sl)
{
      struct symlist *nsl;

      while (sl) {
            nsl = sl->next;
            free(sl);
            sl = nsl;
      }
}


double
eval(struct ast *a)
{
  double v;

  if (!a) {
      yyerror("internal error, null eval");
      return 0.0;
  }

  switch (a->nodetype)
  {
      /* assignment */
    case 'r':
      printf("Rule detected.\n");
      break;

    case 'b':
            printf("question block.\n");
            break;

      /* expressions */

      /* comparisons */
      case '1': v = (eval(a->l) > eval(a->r))? 1 : 0; break;
      case '2': v = (eval(a->l) < eval(a->r))? 1 : 0; break;
      case '3': v = (eval(a->l) >= eval(a->r))? 1 : 0; break;
      case '4': v = (eval(a->l) <= eval(a->r))? 1 : 0; break;
      case '5': v = (eval(a->l) >= eval(a->r))? 1 : 0; break;
      case '6': v = (eval(a->l) != eval(a->r))? 1 : 0; break;
      case '7': v = (eval(a->l) == eval(a->r))? 1 : 0; break;

      /* if-then */
      case 'i':
            if ( eval( ((struct s_flow *)a)->cond) != 0) {
                  if ( ((struct s_flow *)a)->tl) {
                        v = eval( ((struct s_flow *)a)->tl);
                        printf("True.\n");
                  } else {
                        v = 0.0;    // default value, 'nothing'.
                        printf("Nothing.\n");
                  }
            }
            break;

      /* Create an identifier */
      case 'N':
            /* We need a list of identifiers to make sure we don't
               create one with the same name.  If so, ignore. */
            printf("Variable detected.\n");

            break;

      case 'q':
            printf("Question detected in eval.\n");
            break;

  }
}

void
yyerror(char *s, ...)
{
  va_list ap;
  va_start(ap, s);

  fprintf(stderr, "%d: error: ", yylineno);
  vfprintf(stderr, s, ap);
  fprintf(stderr, "\n");
}


main(argc, argv)
int argc;
char **argv;
{

      if (argc > 1) {
            printf("Loading script...\n");
            extern FILE* yyin;
            if(!(yyin = fopen(argv[1], "r"))) {
                  perror(argv[1]);
                  return (1);
            }
            printf("Script loaded.\n");
            printf("Executing...\n");
      }
      else {
            printf("> ");
      }

    // Create the BST
    //BST();

    return yyparse();
}
```

Makefile

```
flexes: flexes.l flexes.y flexes.h
            bison -d flexes.y
            flex -oflexes.lex.c flexes.l
            cc -o $@ flexes.tab.c flexes.lex.c flexesfuncs.c
```

Test 1

```
rule setup
      if 1=1
      then animal becomes 'unknown'
      and ask q_animal .

question q_animal
      "What is your favourite animal?" ;
      input name ;
      because "We need to know what is your favourite animal" .

action go
      run .
```

Test 2

```
rule setup
      if q_animal is unknown
      then animal becomes unknown
      and ask q_animal .

question q_animal
      "What noise does your favourite animal make?" ;
      input name ;
      because "We want to know what animal you like" .

rule cat
      if q_animal is 'meow'
      then animal becomes 'cat' .

rule dog
      if q_animal is 'woof'
      then animal becomes 'dog' .

rule cow
      if q_animal is 'moo'
      then animal becomes 'cow' .

rule pig
      if q_animal is 'oink'
      then animal becomes 'pig' .

rule output
      if q_animal is not unknown
      and animal is not unknown
      then write("The animal is ")
      and write(animal) .

action go
      run .
```

Test 3

```
rule output
      write('This test should fail.') .

action go
      run .
```

Test 4

```
rule setup
    if order is unknown
    and cost is unknown
    and size is unknown
    then ask order .

question order
    "Which item would you like  Please enter a product." ;
    input name ;
    because "Enter a number for the order you want" .

question o_size
    "What size would you like the item Enter a number, 1 is small, 2 is medium, 3 is
large" ;
    input number ;
    because "We need to know how big you want your order" .

rule collate
    if order is not unknown
    and o_size is not unknown
    then cost becomes 0.00 .

rule chips
    if order is 'chips'
    then cost becomes 0.90 .

rule cod
    if order is 'cod'
    then cost becomes 1.50 .

rule pie
    if order is 'pie'
    then cost becomes 1.20 .

rule small
    if o_size = 1
    then size becomes 'small' .

rule medium
    if o_size = 2
    then size becomes 'medium' .

rule large
    if o_size = 3
    then size becomes 'large' .

rule sell
    if o_size is not unknown
    and order is not unknown
    then write('Order is ')
    and write(size)
    and write(' ')
    and write(order) .

action go
```

```
    run .
```

Test 5

```
rule test5
    write('This test should not run') .
```

Test 6

```
% Set up variables
rule setup
    if 1=1
    then result becomes unknown .

% Ask the user for their name
question ur_name
    "What is your name?" ;
    input name ;
    because "We need to know your name" .

% Ask the user for their age
question ur_age
    "How old are you?" ;
    input number ;
    because "We need to know how old you are" .

% Put together the details
rule teenager
    if ur_name is not unknown
    and ur_age is not unknown
    and ur_age < 18
    then result becomes 'you are too young to drink' .

% Rule for folks that are 18-29
rule twenties
    if ur_name is not unknown
    and ur_age >= 18
    and ur_age < 30
    then result becomes 'you are in your prime' .

% rule for folks that are 30-39
rule thirties
    if ur_name is not unknown
    and ur_age >= 30
    and ur_age < 40
    then result becomes 'you are in a great age' .

rule fourtyplus
    if ur_name is not unknown
    and ur_age >= 40
    then result becomes 'you are getting on a bit' .

rule print
    if ur_name is not unknown
    and ur_age is not unknown
    then write(ur_name)
    and write(' ')
    and write(result) .

action go
    run .
```

Test 7

```
rule test7
    "Hello world"
    49 .

action go
    run .
```

Test 8

```
% Declare groups
group drink tea, coffee, chocolate .
group yn yes, no .
group sugar none, one, two, three .

% setup
rule setup
    if 1=1
    then result becomes 'nothing'
    and ask q_drink .

question q_drink
    "What would you like to drink?" ;
    choose from drink ;
    because "We need to know what you would like to drink" .

rule milk
    if q_drink is tea
    or q_drink is coffee
    then ask q_milk .

question q_milk
    "Do you want milk in your drink?" ;
    choose from yn .

rule sugar
    if q_sugar is unknown
    then ask q_sugar .

question q_sugar
    "How many sugars would you like in your drink?" ;
    choose from sugar .

rule output
    if q_drink is not unknown
    then write('Your order is:')
    and nl
    and write(q_drink)
    and nl
    and write(q_sugar)
    and nl
    and write(q_milk) .

action go
    run .
```

Test 9 – Chip Shop Flex

```
group selection yes, no .
group distance near, far .
group openings open, closed .

% determine facts needed

rule ask_hungry
    if hungry is unknown
    then ask hungry .

question hungry
    "Are you hungry?" ;
    choose from selection
    because "you will not want chips if you are not hungry" .

rule ask_money
    if money is unknown
    then ask money .

question money
    "Do you have enough money to buy chips?" ;
    choose from selection
    because "you will not be able to buy chips if you have no money" .

rule ask_near_shop
    if position is unknown
    then ask position .

question position
    "How far is the nearest chip shop?" ;
    choose from distance
    because "the shop may be too far away to buy chips" .

rule shop_open
    if open_shop is unknown
    then ask open_shop .

question open_shop
    "Is the shop open?" ;
    choose from openings
    because "if the shop is closed you will not be buying any chips" .

% rules about buying chips

rule buy_chips
    if hungry is yes
    and money is yes
    and position is near
    and open_shop is open
    then write('You can buy chips')
    and nl .

rule do_not_buy_chips
    if hungry is no
    or money is no
    or position is far
    or open_shop is closed
    then write('You can not buy chips')
    and nl .

% start the program

action buy_or_not
    run .
```