Formal Languages and Compilers
Proff. Breveglieri, Crespi Reghizzi, Morzenti
Written exam[1]: laboratory question - ACSE
05/03/2008

SURNAME: .....
NAME: ..... Student ID: .....
Course: ○ Laurea Specialistica ○ V. O. ○ Laurea Triennale ○ Other:.....
Instructor: ○ Prof. Breveglieri ○ Prof. Crespi ○ Prof. Morzenti

The laboratory question must be answered taking into account the implementation of the `Acse` compiler given with the exam text.

Modify the specification of the lexical analyzer (`flex` input) and the syntactic analyzer (`bison` input) and any other source file required to extend the `Lance` language with the ability to handle a new construct *switch* resembling the one in the following sample.

```
int a;
...
switch (a) {
      case 0:
            ...
            break;
      case 1:
            ...     /* Code block without any "break" statement */
      case 2:
            ...
            break;
      default:
            ...
}
...
```

The `switch` construct is built out of an arbitrary number of `case` blocks and may include an optional `default` block. The `switch` construct has the following semantics: the value of the variable between round brackets is checked for a match against the values indicated in the `case` statements. On a successful match, the corresponding code block is executed. The `break` statement brings the execution to the first instruction *after* the `switch` construct. In case a code block does not contain any `break` statement, the next `case` block (in declaration order) is executed.

[1] Time 45'. Textbooks and notes can be used.
Pencil writing is allowed. Write your name on any additional sheet.

If there are no more `case` blocks left, the code must exit the `switch` construct. If there is no match between the switch variable and the case values, the execution flow jumps to the `default` block (if present) or exits from the `switch` construct. At least a `case` block has to be declared. `Switch` blocks may be nested.

For instance: if we assign the value 0 to the variable `a` in the example, only the first code block (the one corresponding to `case 0`) will be ran. In case `a` evaluates as 1, the blocks related to both `case 1` and `case 2` will be executed. In case the value of `a` differs from $0, 1, 2$ the execution jumps to the `default` block.

Your modifications have to allow the `Acse` compiler to both correctly analyze the syntactical correctness of the aforementioned constructs and to generate a correct translation in the `Mace` assembly language.

The first step, before answering any question, is to devise a translation of the new construct into the assembly language. The solution chosen is presented in the following schema:

```
switch (VAR) {                    ADDI Ry VAR_REG #0
                                  BT begin_test
  case CONST1:                  begin_case1:
    ...                           ...
    break;                        BT switch_end
  case CONST2:                  begin_case2:
    ...                           ...
    /* Block with no break */
  case CONST3:                  begin_case3:
    ...                           ...
    break;                        BT switch_end
  default:                      default_label:
    ...                           ...
}                                 BT switch_end
                                begin_test:
                                  SUBI Rz Ry CONST1
                                  BEQ begin_case1
                                  SUBI Rz Ry CONST2
                                  BEQ begin_case2
                                  SUBI Rz Ry CONST3
                                  BEQ begin_case3
                                  BT default_label
                                switch_end:
```

where Rz and Ry are two temporary registers. In order to produce such a translation, we have to accumulate information about the various cases in some structures and finally produce the code between the labels `begin_test` and `switch_end`. **switch** constructs can be nested, therefore those structures should be contained in a global stack.

1. Define the tokens and the `Acse.lex` and `Acse.y` declarations needed to achieve the required functionality. (3 points)

   In **Acse.lex**:

   ```
   "switch"          { return SWITCH; }
   "break"           { return BREAK; }
   "case"            { return CASE; }
   "default"         { return DEFAULT; }
   ```

   In **Acse.y**:

   ```
   %token SWITCH
   %token BREAK
   %token CASE
   %token DEFAULT
   ```

2. Define the syntactic rules needed to achieve the required functionality. (8 points)

The switch construct is generated by the new non-terminal *switch_statement*, which is expanded by a few new rules:

switch_statement : SWITCH LPAR IDENTIFIER RPAR LBRACE *switch_block* RBRACE ;

switch_block : *case_statements*
| *case_statements default_statement* ;

case_statements : *case_statements case_statement*
| *case_statement*
;

case_statement : CASE NUMBER COLON *statements* ;

default_statement : DEFAULT COLON *statements* ;

switch_statement is added as a new possibility for the non-terminal *control_statement*; moreover, the **break** statement must be added as a possible statement:

control_statement : ...
| *break_statement* SEMI
| *switch_statement*
;

break_statement : BREAK ;

Please notice that the **break** statement is an ordinary statement, which could be appear anywhere, even inside an **if** construct for example. A semantic check will be provided to avoid that a **break** appears outside any **switch**.

3. Define the semantic actions needed to achieve the required functionality, without considering the `break` and `default` statements. (10 points)

   We have to define a few structures in **axe_struct.h**:

```
typedef struct
{
    t_axe_label *begin_case_label;
    int number;
} t_case_statement;

typedef struct
{
    int cmp_register;
    t_list *cases;   /* List of t_case_statement elements;
                      * each element contains the constant
                      * number and the label for a case */
    t_axe_label *switch_end;  /* Label of the end of the
                               * switch construct */
    t_axe_label *begin_test;  /* Label of the tests */
} t_switch_statement;
```

   and in **Acse.y**:

```
%{
    ...
    t_list *switchStack = NULL;  /* Elements are of type
                                  * t_switch_statement */
%}
%union {
    ...
    t_switch_statement *switch_stmt;
}
%token <switch_stmt> SWITCH
```

   The stack for the **switch** structures uses the functions contained in **collections.h**.

The semantic actions that produce the above translation:

```
switch_statement : SWITCH LPAR IDENTIFIER RPAR LBRACE
    {
        $1 = (t_switch_statement *)malloc(
             sizeof(t_switch_statement));
        $1->cmp_register = getNewRegister(program);
        gen_addi_instruction(program, $1->cmp_register,
             get_symbol_location(program,$3,0), 0);
        $1->begin_test = reserveLabel(program);
        $1->switch_end = reserveLabel(program);
        switchStack = addFirst(switchStack, $1); // PUSH
        gen_bt_instruction(program, $1->begin_test, 0);
    }
    switch_block RBRACE
    {
        t_list *p;
        int cmpReg; // This is the Rz register above
        fixLabel(program, $1->begin_test);
        cmpReg = getNewRegister(program);
        p = $1->cases;
        while (p!=NULL) {
            gen_subi_instruction(program, cmpReg,
                 $1->cmp_register,
                 ((t_case_statement *)p->data)->number);
            gen_beq_instruction(program,
                 ((t_case_statement *)p->data)
                 ->begin_case_label, 0);
            p = p->next;
        }
        fixLabel(program,$1->switch_end);
        switchStack = removeFirst(switchStack); // POP
    }
  ;

switch_block : case_statements
    {
        gen_bt_instruction(program,
             ((t_switch_statement *)LDATA(
               getFirst(switchStack)))->switch_end, 0);
    }
  ;
```

```
case_statement: CASE NUMBER COLON
    {
        t_case_statement *c = (t_case_statement *)
            malloc(sizeof(t_case_statement));
        c->number = $2;
        c->begin_case_label = assignNewLabel(program);
        ((t_switch_statement *)LDATA(
              getFirst(switchStack)))->cases =
            addLast(((t_switch_statement *)LDATA(
               getFirst(switchStack)))->cases, c);
    }
    statements
  ;
```

4. Define the semantic actions needed to handle the optional `default` block. (6 points)

   The structure in **axe_struct.h** must be extended with a new field:

```
typedef struct
{
    ...
    t_axe_label *default_label; /* Label of the default
                                 * block (can be NULL) */
} t_switch_statement;
```

   The actions for the rules related to the **default** block:

```
switch_block : ...
  | case_statements default_statement
    {
        gen_bt_instruction(program,
             ((t_switch_statement *)LDATA(
               getFirst(switchStack)))->switch_end, 0);
    }
  ;

default_statement: DEFAULT COLON
    {
        ((t_switch_statement *)LDATA(
              getFirst(switchStack)))->default_label =
            assignNewLabel(program);
    }
    statements
  ;
```

   and the generation of the jump to the default block (if any), just before the **switch_end** label:

```
switch_statement : SWITCH LPAR IDENTIFIER RPAR LBRACE
    { ... }
    switch_block RBRACE
    {
        ...
        if ($1->default_label !=  NULL)
            gen_bt_instruction(program,
                $1->default_label, 0);
        // fixLabel(program,$1->switch_end);
        ...
    }
  ;
```

5. Define the semantic actions needed to handle the `break` construct. (6 points)

```
break_statement : BREAK
    {
        if (switchStack == NULL) {
            // If there is no active switch,
            // GENERATE AN ERROR!!!!
            abort();
        } else {
            gen_bt_instruction( program,
                ((t_switch_statement *)LDATA(
                  getFirst(switchStack)))->switch_end, 0 );
        }
    }
  ;
```