# Formal Languages and Compilers
# Proff. Breveglieri, Crespi Reghizzi, Sbattella
# Written exam[1]: laboratory question
# 07/02/2007

**WITH SOLUTIONS**

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

The laboratory question must be answered taking into account the implementation of the `Simple` 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 `Simple` language with the ability to handle unconditional jumps specified through goto statements to labels:

```
declarations
 integer a, b, c.
 ...
begin
        ...
 foo   : a := c + 3;
              goto foo;
        c := 5 + a;
              goto bar;
        b := e * 7;
 bar : a := 1;
        ...
end
```

The modifications must allow the `Simple` compiler to detect the abovedescribed statements as syntactically correct, and generate a correct translation in the `SimpleVM` assembly language.

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

1. Define the tokens and the Simple.lex and Simple.y declarations needed to achieve the required functionality.
   **SOLUTION:**

   ```
   Simple.lex:

   "goto" { return(GOTO); }

   Simple.y:

   %token GOTO
   ```

2. Define the syntactic rules needed to achieve the required functionality.
   **SOLUTION:**

   ```
   Simple.y:

   commands: /* empty */
      | commands labeled_command ';'

   labeled_command : command
      | IDENTIFIER ':' command

   command :
      ...
      | GOTO IDENTIFIER
   ```

3. Define the semantic actions needed to achieve the required functionality.

   **SOLUTION:**

   As jumps may be read before the respective labels are, it is necessary to have a mechanism to store the pairs (jump location, label name). A list like the following one suffices:

```
typedef struct _pending_goto {
   int pos;
   int backpatched;
   char *label;
   struct_pending_goto *next;
} pending_goto;

pending_goto *pending_goto_list = NULL;
```

   with the appropriate primitives for list insertion and search:

```
insert_pending (char *label) {
   pending_goto *new = malloc (sizeof (pending_goto));
   new->label = strdup (label);
   new->backpatched = 0;
   new->pos = reserve_code_loc ( );
   new->next = pending_goto_list;
   pending_goto_list = new;
}

pending_goto *get_next_pending_goto (
   pending_goto *start, char *label
) {
   if (start == NULL) start = pending_goto_list;
   else start = start->next;
   while (start)
       if (strcmp (start->label, label) == 0) return start;
       else start = start->next;
   return NULL;
}
```

   Moreover, one need change the symbol table for storing label names:

```
typedef struct _symrec {
    ...
   int type /* declare an enumerative type: (VAR, LABEL); */
   int target;
} symrec
```

```
Simple.y:

program:
   DECLARATIONS declarations BEGIN_PROGRAM commands END_PROGRAM
   { .....
      p = pending_goto_list;
      while (p != NULL) {
         if (!p->backpatched) {
            printf ("Error: unresolved GOTO at position %u", p->pos);
            exit(1);
         }
      }

labeled_command :
   ...
   | IDENTIFIER ':' {
       pending_goto *i;
       insert ($1, LABEL, current_code_loc ( )); /* insert must be modified
       to take additional parameters, as seen in other exams */
       for (
           i = get_next_pending_goto (NULL, $1);
           i != NULL;
           i = get_next_pending_goto (i, $1)
       ) gen_back_code (i->pos, GOTO, current_code_loc ( ));
           i->backpatched = 1;
    }

command :
   ...
   | GOTO IDENTIFIER {
           symrec *label;
           label = getsym ($2);
           if (!label) insert_pending ($1);
           else if (label->type != LABEL) {
               /* handle type mismatch error */
           } else {
                gen_code (GOTO, label->target);
           }
   }
```

4. Define semantic controls (and the related symbol table modifications) to check whether labels have unique names.

   **SOLUTION:**

   The uniqueness of the solution is granted by function “insert”, which prevents the symbols with the same name of being inserted twice.

```
int insert (char *sym_name, int type, int pos) {
   symrec *s;
   s = getsym (sym_name);
   if (s == 0) {
       s = putsym (sym_name);
       s->type = type;
       if (type == LABEL) s->target = pos;
       return 0;
   } else {
       errors++
       fprintf (stderr, "item %s already defined\n", syn_name);
       return 1;
   }
}
```