Министерство образования и науки Российской Федерации
Санкт-Петербургский Государственный Политехнический Университет
—
Высшая школа программной инженерии
Направление «Фундаментальная информатика и информационные технологии»

Курсовая работа
Расширение языка Milan для построения транслятора

Выполнила:
студент гр. 3530202/90002  Потапова А. М.

Руководитель: Тышкевич А. И.

Санкт-Петербург
2021

Оглавление

Задание

Изучить компилятор языка Milan и добавить в него поддержку операторов ***break*** и ***continue*** в цикле.

Описание решения

Для реализации данного задания были разобраны исходные коды компилятора ymilan, включающего в себя лексический анализатор, синтаксический анализатор, модуль работы с абстрактным синтаксическим деревом, модуль генерации кода для виртуальной машины MiLan'a, а также основной модуль, в котором функции предыдущих модулей вызываются в правильной последовательности.

Лексический анализатор осуществляет преобразование входного потока символов в потом терминальных символов грамматики MiLan'a, т.е. лексем. Например, 'C' 'O' 'N' 'T' 'I' 'N' 'U' 'E' при условии внесения описанных в данной работе изменений в язык, будет преобразовано в лексему T_CONTINUE. Синтаксическим разбором потока в соответствии с грамматикой языка занимается синтаксический анализатор. Во время своей работы он строит промежуточную структуру данных — дерево разбора. Построенное дерево разбора используется компилятором MiLan'a для генерации кода. Это происходит путем рекурсивного обхода дерева разбора и формированием цепочек команд, соответствующих каждому узлу дерева. Основной модуль программы обрабатывает параметры командной строки, открывает необходимые файлы и вызывает функцию синтаксического анализа. В случае успешного синтаксического разбора вызываются функции генерации кода и печати его в выходной файл, после чего файлы закрываются и выполнение программы завершается.

Решение

Измененные/добавленные фрагменты кода выделены **жирным шрифтом**.

Файл lexer.l

//добавлены новые лексемы

```
BREAK                   { return T_BREAK; }
CONTINUE                { return T_CONTINUE; }
```

Файл parser.y

//добавлены терминалы

```
%token T_BREAK
%token T_CONTINUE
```

//добавлены правила операторов break и continue в правую часть нетерминала stmt

```
stmt: /* Empty */                         { $$ = NULL;               }
     | T_WRITE '(' expr ')'               { $$ = new_write($3);      }
     | T_IDENT T_ASSIGN expr              { $$ = new_assign($1, $3); }
     | T_WHILE  cond T_DO stmt_list T_OD  { $$ = new_while($2, new_block($4)); }
     | T_IF cond T_THEN stmt_list rest_if { $$ = new_if($2, new_block($4), $5); }
     | T_BREAK ';' stmt_list              { $$ = new_break(new_block($3));   }
     | T_BREAK                            { $$ = new_break(NULL);            }
     | T_CONTINUE ';' stmt_list           {$$ = new_continue(new_block($3)); }
     | T_CONTINUE                         { $$ = new_continue(NULL);         }
     ;
```

Файл ast.c

//добавлены функции new_break и new_continue

```
ast_node* new_break(ast_node* arg1) {
    ast_node* node;
    node = (ast_node*)malloc(sizeof(ast_node));
    if(node) {
        node->type = Node_Break;
        node->data.ast_value = arg1;
    }

    return node;
}

ast_node* new_continue(ast_node* arg1) {
    ast_node* node;
    node = (ast_node*)malloc(sizeof(ast_node));
    if(node) {
        node->type = Node_Continue;
        node->data.ast_value = arg1;
    }

    return node;
}
```

Файл ast.h

//добавлены новые типы узлов Node_Break и Node_Continue

```
typedef enum {
        Node_Block = 0,
```

```
        Node_Const,
        Node_Var,
        Node_Read,
        Node_Expr,
        Node_Assign,
        Node_Write,
        Node_Cond,
        Node_If,
        Node_While,
        Node_Break,
        Node_Continue
} node_type;


ast_node* new_break(ast_node* arg1);
ast_node* new_continue(ast_node* arg1);
```

Файл code.c

```
const int BREAK_ADDR_MAX_SIZE = 128; //максимальный размер массива break_addr
int break_addr[128][128] = {{0}}; //адрес вставки инструкции JMP
int break_counter[128] = {0}; //счетчик break на каждом уровне вложенности


const int CONTINUE_ADDR_MAX_SIZE = 128; //максимальный размер массива continue_addr
int continue_addr[128][128] = {{0}}; //адрес вставки инструкции JMP
int continue_counter[128] = {0}; //счетчик continue на каждом уровне вложенности

int in_while = 0; //счетчик уровня вложенности


unsigned int generate_code(FILE* stream, unsigned int address, ast_node* ast)
{
   ast_node* ptr;
   opcode op;
   unsigned int tmpaddr1, tmpaddr2;
   unsigned int new_address;
   int i = 0;
   int j = 0;

   if(ast) {
       switch(ast->type) {
          case Node_Block:
             ptr = ast->sub[0];
             new_address = address;
             while(ptr) {
                new_address = generate_code(stream, new_address, ptr);
                ptr = ptr->next;
             }
             break;

          case Node_Const:
             new_address = generate_command(stream, address, PUSH, ast-
>data.integer_value);
             break;

          case Node_Var:
             new_address = generate_command(stream, address, LOAD,
get_var_address(ast));
             break;

          case Node_Read:
             new_address = generate_command(stream, address, INPUT, 0);
             break;

          case Node_Expr:
```

```
            new_address = generate_code(stream, address, ast->sub[0]);

            if(ast->data.integer_value != OP_NEG) {
               new_address = generate_code(stream, new_address, ast->sub[1]);
            }

            switch(ast->data.integer_value) {
               case OP_ADD: op = ADD; break;
               case OP_SUB: op = SUB; break;
               case OP_MUL: op = MULT; break;
               case OP_DIV: op = DIV; break;
               case OP_NEG: op = INVERT; break;
               default:
                 milan_error("Unknown arithmetical operator");
                 break;
             }

             new_address = generate_command(stream, new_address, op, 0);
             break;



         case Node_Assign:
            new_address = generate_code(stream, address, ast->sub[0]);
            new_address = generate_command(stream, new_address, STORE,
get_var_address(ast));
            break;

         case Node_Write:
            new_address = generate_code(stream, address, ast->sub[0]);
            new_address = generate_command(stream, new_address, PRINT, 0);
            break;

         case Node_Cond:
            new_address = generate_code(stream, address, ast->sub[0]);
            new_address = generate_code(stream, new_address, ast->sub[1]);
            new_address = generate_command(stream, new_address, COMPARE, ast-
>data.integer_value);
            break;

         case Node_If:
            new_address = generate_code(stream, address, ast->data.ast_value);
            tmpaddr1 = new_address++;
            new_address = generate_code(stream, new_address, ast->sub[0]);

            if(ast->sub[1]) {
               generate_command(stream, tmpaddr1, JUMP_NO, new_address + 1);
               tmpaddr1 = new_address++;
               new_address = generate_code(stream, new_address, ast->sub[1]);
               generate_command(stream, tmpaddr1, JUMP, new_address);
            }
            else
            {
               generate_command(stream, tmpaddr1, JUMP_NO, new_address);
            }
            break;

         case Node_While:
            ++in_while;
            if (in_while > 128) {
                milan_error("Too many nested loops");
            }

            tmpaddr1 = address;
            new_address = generate_code(stream, address, ast->data.ast_value);
            tmpaddr2 = new_address++;

            new_address = generate_code(stream, new_address, ast->sub[0]);
            new_address = generate_command(stream, new_address, JUMP, tmpaddr1);
```

```
            generate_command(stream, tmpaddr2, JUMP_NO, new_address);

            //записываем JUMP во все сохраненные адреса break_addr и continue_addr
            for (i = 0; i < break_counter[in_while - 1]; ++i){
                generate_command(stream, break_addr[in_while - 1][i], JUMP,
new_address);
            }

            for (j = 0; j < continue_counter[in_while - 1]; ++j){
                generate_command(stream, continue_addr[in_while - 1][j], JUMP,
tmpaddr1);
            }

            //обнуляем массив break_addr и continue_addr на данном уровне вложенности
            memset(break_addr[in_while - 1], 0, sizeof(int) * BREAK_ADDR_MAX_SIZE);
            memset(continue_addr[in_while - 1], 0, sizeof(int) *
CONTINUE_ADDR_MAX_SIZE);

            //обнулили счетчик
            break_counter[in_while - 1] = 0;
            continue_counter[in_while - 1] = 0;

            //уменьшаем счетчик уровня вложенности
            --in_while;
            break;

        case Node_Break:
            if (in_while > 0) {
                if (break_counter[in_while - 1] < BREAK_ADDR_MAX_SIZE) {
                    //первая свободная адресная ячейка после кода
                    new_address = address;

                    //присваиваем ячейке break_addr адрес, куда поместим команду JMP и
увеличиваем счетчик break _counter
                    break_addr[in_while - 1][break_counter[in_while - 1]++] =
new_address++;
                    if (ast->data.ast_value != NULL) {
                        //если за BREAK есть код, то будем его генерировать
                        new_address = generate_code(stream, new_address, ast-
>data.ast_value);
                    }

                } else
                    milan_error("Too much breaks in while");
            } else
                milan_error("Break should be in loop");

            break;

          case Node_Continue:
            if (in_while > 0) {
                if (continue_counter[in_while - 1] < CONTINUE_ADDR_MAX_SIZE) {
                    //первая свободная адресная ячейка после кода
                    new_address = address;

                    //присваиваем ячейке continue_addr адрес, куда поместим команду JMP и
увеличиваем счетчик continue_counter
                    continue_addr[in_while - 1][continue_counter[in_while - 1]++] =
new_address++;

                    if (ast->data.ast_value != NULL) {
                        //если за CONTINUE есть код, то будем его генерировать
                        new_address = generate_code(stream, address, ast-
>data.ast_value);
                    }
```

```
                  } else
                        milan_error("Too much continues in while");
                  } else
                       milan_error("Continue should be in loop");
                  break;

              default:
                  milan_error("Unknown node type");
                  break;
              }

              return new_address;
          }
          else {
                  return address;
          }
}
```

Анализ результатов

Для тестирования модернизированного языка Milan был созданы файлы break.mill, continue.mill и break_continue.mill:

Файл break.mill

```
BEGIN
    i := READ;
    WHILE i > 0 DO
        WRITE(i);
        IF i = 5 THEN
            WRITE(i);
            BREAK
        FI;
        i := i - 1
    OD
END
```

Файл continue.mill

```
BEGIN
    i := READ;
    WHILE i > 0 DO
        i := i - 1;

        IF i = 3 THEN
            CONTINUE
        FI;

        IF i = 4 THEN
            CONTINUE
        FI;

        WRITE(i)
    OD
END
```

Файл break_continue.mill

```
BEGIN
    i := READ;
    WHILE i > 0 DO
          i := i - 1;

          IF i = 3 THEN
              CONTINUE
          FI;

          IF i = 4 THEN
              CONTINUE
          FI;

          IF i = 2 THEN
              WRITE(i);
              BREAK
          FI;

          WRITE(i)
    OD
END
```

Файл break_error.mill (файл с ошибкой)

Описание ошибки: оператор BREAK вне цикла

```
BEGIN
    i := READ;
    BREAK
END
```

Результатом работы компилятора языка MiLan стал следующий набор команд стековой машины:

Файл break.out

```
SET     0       0       ; i

0:      INPUT
1:      STORE           0
2:      LOAD            0
3:      PUSH            0
4:      COMPARE         3
6:      LOAD            0
7:      PRINT
8:      LOAD            0
9:      PUSH            5
10:     COMPARE         0
12:     LOAD            0
13:     PRINT
11:     JUMP_NO         15
15:     LOAD            0
16:     PUSH            1
17:     SUB
18:     STORE           0
19:     JUMP            2
5:      JUMP_NO         20
14:     JUMP            20
20:    STOP
```

Файл continue.out

```
SET     0       0       ; i

0:      INPUT
1:      STORE           0
2:      LOAD            0
3:      PUSH            0
4:      COMPARE         3
6:      LOAD            0
7:      PUSH            1
8:      SUB
9:      STORE           0
10:     LOAD            0
11:     PUSH            3
12:     COMPARE         0
13:     JUMP_NO         15
15:     LOAD            0
16:     PUSH            4
17:     COMPARE         0
18:     JUMP_NO         20
20:     LOAD            0
21:     PRINT
22:     JUMP            2
5:      JUMP_NO         23
14:     JUMP            2
19:     JUMP            2
23:    STOP
```

Файл break_continue.out

```
SET     0       0       ; i

0:      INPUT
1:      STORE           0
2:      LOAD            0
3:      PUSH            0
4:      COMPARE         3
6:      LOAD            0
7:      PUSH            1
8:      SUB
9:      STORE           0
10:     LOAD            0
11:     PUSH            3
12:     COMPARE         0
13:     JUMP_NO         15
15:     LOAD            0
16:     PUSH            4
17:     COMPARE         0
18:     JUMP_NO         20
20:     LOAD            0
21:     PUSH            2
22:     COMPARE         0
24:     LOAD            0
25:     PRINT
23:     JUMP_NO         27
27:     LOAD            0
28:     PRINT
29:     JUMP            2
5:      JUMP_NO         30
26:     JUMP            30
14:     JUMP            2
19:     JUMP            2
30:    STOP
```

Файл break_error.out (файл с ошибкой)

```
SET     0 0     ; i

0:      INPUT
1:      STORE           0
Milan error: break should be in loop
```

При запуске виртуальной машины и вводе числа 10 были получены следующие результаты:

Запуска файла break.out	Запуска файла continue.out
Reading input from break.out > 10 10 9 8 7 6 5 5	Reading input from continue.out > 10 9 8 7 6 5 2 1 0

Запуска файла break _continue.out
Reading input from break_continue.out > 10 9 8 7 6 5 2

Заключение

Полученные данные полностью совпали с ожидаемыми. В ходе проведения различных тестов наблюдалась корректная работа вновь добавленных операторов break и continue в цикле.