Міністерство освіти і науки, молоді та спорту України
Національний Технічний Університет України
“Київський Політехнічний Інститут ім.. Сікорського”
Факультет прикладної математики
Кафедра СПіСКС

Лабораторна робота №2

з дисципліни
“ІПЗ. Основи проектування трансляторів”

Тема: “Розробка генератору коду”

Виконав:
Студент групи КВ-82
Іваненко Олександр
Варіант 10

Київ 2021

Постановка задачі

1. Розробити програму генератора коду (ГК) для підмножини мови програмування SIGNAL, заданої за варіантом.

2. Програма має забезпечувати:

- читання дерева розбору та таблиць, створених синтаксичним аналізатором, який було розроблено в розрахунково-графічній роботі;
- виявлення семантичних помилок;
- генерацію коду та/або побудову внутрішніх таблиць для генерації коду.

3. Входом генератора коду (ГК) мають бути:

- дерево розбору;
- таблиці ідентифікаторів та констант з повною інформацією, необхідною для генерації коду;
- вхідна програма на підмножині мови програмування SIGNAL згідно з варіантом (необхідна для формування лістингу програми).

4. Виходом ГК мають бути:

• асемблерний код згенерований для вхідної програми та/або внутрішні таблиці для генерації коду;

• внутрішні таблиці генератора коду (якщо потрібні).

5. Скомпонувати повний компілятор, що складається з розроблених раніше лексичного та синтаксичного аналізаторів і генератора коду, який забезпечує наступне:

• генерацію коду та/або побудову внутрішніх таблиць для генерації коду;

• формування лістингу вхідної програми з повідомленнями про лексичні, синтаксичні та семантичні помилки.

6. Входом компілятора має бути програма на підмножині мови програмування SIGNAL згідно з варіантом;

7. Виходом компілятора мають бути:

• асемблерний код згенерований для вхідної програми та/або внутрішні таблиці для генерації коду;

• лістинг вхідної програми з повідомленнями про лексичні, синтаксичні та семантичні помилки.

Варіант 10

1. <signal-program> --> <program>
2. <program> --> PROGRAM <procedure-identifier> ; <block> ;
3. <block> --> <declarations> BEGIN <statements-list> END
4. <statements-list> --> <empty>
5. <declarations> --> <procedure-declarations>
6. <procedure-declarations> --> <procedure> <procedure-declarations> | <empty>
7. <procedure> --> PROCEDURE <procedure-identifier><parameters-list> ;
8. <parameters-list> --> (<declarations-list>) | <empty>
9. <declarations-list> --> <declaration> <declarations-list> | <empty>
10. <declaration> --><variable-identifier><identifiers-list>:<attribute><attributes-list> ;
11. <identifiers-list> --> , <variable-identifier> <identifiers-list> | <empty>
12. <attributes-list> --> <attribute> <attributes-list> | <empty>
13. <attribute> --> SIGNAL | COMPLEX | INTEGER | FLOAT | BLOCKFLOAT | EXT
14. <variable-identifier> --> <identifier>
15. <procedure-identifier> --> <identifier>
16. <identifier> --> <letter><string>
17. <string> --> <letter><string> | <digit><string> | <empty>
18. <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
19. <letter> --> A | B | C | D | ... | Z

Лістинг програми

```
#include "generator.h"

Generator::Generator(Parser *parser, Lexer lexer) {
    this->parser = parser;
    all_lexemes = lexer.get_all_lexemes();
}

void Generator::generate() {
    traverseTree(parser->syn_tree.root);
    push_asm("code ends");
}

void Generator::traverseTree(TreeNode* node) {
    if(error) return;
    if (node->rule == "<program>"){
        TreeNode *id = node->children[1]->children[0];
        ids_proc.push_back(id->token);
        push_asm("@prog " + node->children[1]->children[0]->name + ":");
        push_asm("data segment");
        push_asm("data ends\n");
        push_asm("code segment");
    }
    if (node->rule == "<procedure>"){
        generateProcedure(node);
    }
    for (int i = 0; i < node->children.size(); i++) {
        traverseTree(node->children[i]);
    }
}

void Generator::generateProcedure(TreeNode *node) {

    TreeNode *id = node->children[1]->children[0];
    checkOnError(id);
    ids_proc.push_back(id->token);
    ids_params.push_back(id->token);
    push_asm("@proc " + id->name);
    curr_name_proc = id->name;
    push_asm("\tpush ebp");
    push_asm("\tmov ebp, esp");
    generateParameters(node->children[2]);


    push_asm("\tpop epb");
    push_asm("\tret");
    push_asm("@endp");
    ids_params.clear();
}

void Generator::generateParameters(TreeNode *node) {
    if(node->children[0]->rule == "<empty>") return;
    traverseDeclaration(node->children[1], 0);
}
```

```
void Generator::traverseDeclaration(TreeNode *node, int offset) {
    if(error) return;
    if(node->children[0]->rule == "<empty>") return;
    traverseVars(node->children[0], offset);
    traverseDeclaration(node->children[1], offset);
}

void Generator::traverseVars(TreeNode *node, int &offset) {
    if(error) return;
    if(node->children[0]->rule == "<empty>") return;
    offset += 4;
    TreeNode *id;
    if(node->rule == "<declaration>")id = node->children[0]->children[0];
    else id = node->children[1]->children[0];
    checkOnError(id);
    ids_params.push_back(id->token);
    push_asm("\tmov " + id->name + ", [ebp - " + to_string(offset) + "]");
    if(node->rule == "<declaration>")traverseVars(node->children[1], offset);
    else traverseVars(node->children[2], offset);
}

void Generator::printAsm(string test_folder) {
    if(error) return;
    ofstream out;
    out.open(test_folder + "generated.txt", ios::app);
    out << "\n\nASM:\n\n";
    cout << "\n\nASM:\n\n";
    for(string str :asm_code){
        out << str << endl;
        cout << str << endl;
    }
}

void Generator::push_asm(string str){
    asm_code.push_back(str);
}

void Generator::checkOnError(TreeNode *node){
    vector<int>::iterator it = find (ids_params.begin(), ids_params.end(),
node->token);
    if(it != ids_params.end()){
        cout << "[Generator error]: variable name [" + node->name + "]
repeated in " + "[" + curr_name_proc + "] procedure";
        error = true;
    }
    it = find (ids_proc.begin(), ids_proc.end(), node->token);
    if(it != ids_proc.end()){
        cout << "[Generator error]: there is already a procedure with the
name [" + node->name + "] on " + getRow(node->token);
        error = true;
    }
}

string Generator::getRow(int token) {
    int row = 0;
```

```
    for(Lexeme one : all_lexemes){
        if(one.get_code() == token){
            row = one.get_row();
            break;
        }
    }
    return "row #" + to_string(row);
}
```

```
#ifndef LAB1_GENERATOR_H
#define LAB1_GENERATOR_H

#include "parser.h"

class Generator {
    private:
    string curr_name_proc = "";
    bool error = false;
    vector<string> asm_code;
    vector<int> ids_proc;
    vector<int> ids_params;
public:
    vector<Lexeme> all_lexemes;
    Generator(Parser *parser, Lexer lexer);

    Parser* parser;
    Lexer* lexer;

    void generate();

    void traverseTree(TreeNode* node);

    void traverseDeclaration(TreeNode *, int);

    void printAsm(string test_folder);

    void push_asm(string str);

    void generateProcedure(TreeNode *pNode);

    void generateParameters(TreeNode *node);

    void traverseVars(TreeNode *node, int &);

    void checkOnError(TreeNode *node);

    string getRow(int token);
};


#endif
```

Приклад роботи програми

Повний приклад без помилок:

```
program qw;

procedure bla(a1 : float;);

procedure check1(a1, b1, c1 : float blockfloat;
a2, b2, c2 : float blockfloat;);

procedure check (float1 : float;);

procedure V1 ();

procedure k;

begin

end ;

(**)
```

Результат генерації:

```
@prog qw:
data segment
data ends

code segment
@proc bla
        push ebp
        mov ebp, esp
        mov a1, [ebp - 4]
        pop epb
        ret
@endp
@proc check1
        push ebp
        mov ebp, esp
        mov a1, [ebp - 4]
        mov b1, [ebp - 8]
        mov c1, [ebp - 12]
        mov a2, [ebp - 16]
        mov b2, [ebp - 20]
        mov c2, [ebp - 24]
        pop epb
        ret
@endp
@proc check
```

```
        push ebp
        mov ebp, esp
        mov float1, [ebp - 4]
        pop epb
        ret
@endp
@proc V1
        push ebp
        mov ebp, esp
        pop epb
        ret
@endp
@proc k
        push ebp
        mov ebp, esp
        pop epb
        ret
@endp
code ends
```

Приклади з помилками:

```
program qw;
procedure proc(fie, fie : float;);
begin
end;
```

```
[Generator error]: variable name [fie] repeated in [proc] procedure
```

```
program qw;
procedure proc(fie, fie1 : float;);
procedure proc(fie, fie1 : float;);
begin
end;
[Generator error]: there is already a procedure with the name [proc] on row #2
```