语义分析实验报告

1. 编译与测试

1.1 编译环境

- Linux version 5.15.153.1-microsoft-standard-WSL2
- g++ (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0

1.2 编译命令

```
cd MyCompiler
g++ -o sysc AST.cpp ASM.cpp lex.yy.c syntax.tab.c
```

1.3 测试用例

采用自动化脚本 `run_tests.sh` 进行测试，测试用例见 `tests` 文件夹，包括前后发布的11个测试用例，以及两个隐藏测试用例。

```
./run_tests.sh
```

测试结果输出到 `./test_result` 文件。

```
tests/0: Success
tests/1: Success
tests/10: Success
tests/2: Success
tests/3: Success
tests/4: Success
tests/5: Success
tests/6: Success
tests/7: Success
tests/8: Success
tests/9: Success
tests/hide-1: Success
tests/hide-2: Success
```

2. 语义分析

2.1 前端

采用抽象语法树作为中间语言表示，文件 `AST.cpp` 定义了抽象语法树的节点类型，文件 `syntax.y` 定义了语法规则，文件 `lex.l` 定义了词法规则。

```
class Tree {
    public:
        virtual ~Tree() = default;
        virtual void print(int parent, string part) = 0;

        int newLabel();

        bool err_empty = false;

        vector<int> next_list;
        vector<int> true_list;
        vector<int> false_list;
        int quad = 0;
};
```

语法树构建过程实现在 `syntax.y` 文件中，在语法实验部分已基本实现并介绍，语义实验仅针对部分语法做出少许调整。

```
UnaryExp
: '&' IDENT ArrayIndex {
    auto unaryExp = new UnaryExp();
    unaryExp->unaryExpType = UnaryExpType::OP_Exp;
    unaryExp->op = "&";
    unaryExp->ident = *($2);
    unaryExp->ptr_to_array = true;
    unaryExp->arrayIndex = shared_ptr<ArrayIndex>((ArrayIndex* )$3);
    $$ = unaryExp;
}
```

2.2 后端

通过自上而下遍历语法树，实现语义分析，并生成 `x86-64 intel` 汇编代码。

在 ASM.h 与 ASM.cpp 中定义 CodeGenerator 类，实现语义分析和代码生成。

```
class CodeGenerator {
    public:

        CodeGenerator();

        int str_num = 0;
        int label = 1;

        shared_ptr<SymbolTable> global_table;

        inline int search_const_str(string str);

        pair<bool, int> handle_exp(shared_ptr<SymbolTable> table, shared_ptr<Tree> node);

        pair<bool, int> handle_initval(shared_ptr<SymbolTable> table, shared_ptr<Tree> node);
        int handle_arr_initval(shared_ptr<SymbolTable> table, shared_ptr<ArraySymbol> array, sha
        void handle_constdef(shared_ptr<SymbolTable> table, shared_ptr<ConstDef> node);
        void handle_vardef(shared_ptr<SymbolTable> table, shared_ptr<VarDef> node);
        shared_ptr<SymbolTable> handle_funcDef(shared_ptr<SymbolTable> table, shared_ptr<FuncDef
        void handle_cond(shared_ptr<SymbolTable> table, shared_ptr<Tree> node, int true_label, i
        void handle_assign(shared_ptr<SymbolTable> table, shared_ptr<Stmt> node);
        void traverse(shared_ptr<SymbolTable> table, shared_ptr<Tree> node);

        shared_ptr<SymbolType> lookup(shared_ptr<SymbolTable>curr_table, string name);

        void dump(shared_ptr<CompUnit> comp_unit, string filename);

        void fill_zero(shared_ptr<SymbolTable> table, int array_base, int offset, int fill_size)

        unordered_map<string, int> const_str;
        vector<shared_ptr<SymbolTable>> tables;
        vector<string> rdata;
        vector<string> global_data;
        vector<string> text;

        unordered_map<string, shared_ptr<FuncSymbol>> lib_funcs;
        bool check_is_lib(string name);

        string judge_const(pair<bool, int>& res, bool is_rax = true);

        ~CodeGenerator() = default;
};
```

`CodeGenerator::traverse` 函数实现了自上而下遍历语法树，通过递归调用处理不同类型的节点，调用不同的处理函数。

```
void CodeGenerator::traverse(shared_ptr<SymbolTable> table, shared_ptr<Tree> node);
```

`CodeGenerator::global_table` 为全局符号表，存储全局变量、函数等信息，由于 C 语言不支持函数嵌套定义，因此每次遍历到函数定义节点时，创建一个新的符号表，节点处理完毕后，将符号表加入到 `vector<shared_ptr<SymbolTable>> tables` 中。

```
table = make_shared<SymbolTable>(SymbolTable::Scope::LOCAL, node->ident);

if (node->funcType->type == "void")
    symbol->kind = FuncSymbol::FuncKind::VOID;
else
    symbol->kind = FuncSymbol::FuncKind::INT;

table->insert(node->ident, symbol);
table->curr_func_kind = symbol->kind;

table->local_label = node->ident;

text.push_back(node->ident);

table->footer_code.push_back(".L" + to_string(table->get_func_ret_label()) + ":");
table->footer_code.push_back("\tleave");
table->footer_code.push_back("\tret");

global_table->insert(node->ident, symbol);
...
tables.push_back(table);
```

在每个符号表内维护作用域。

```
shared_ptr<SymbolType> SymbolTable::lookup(string name) {
    for (int i = level; i >= 0; i--) {
        if (map_table[i].find(name) != map_table[i].end())
            return map_table[i][name];
    }
        return nullptr;
}

void SymbolTable::insert(string name, shared_ptr<SymbolType> symbol) {
    map_table[level][name] = symbol;
}

void SymbolTable::enter_scope() {
    map_table.push_back(unordered_map<string, shared_ptr<SymbolType>>());
    level++;
}

void SymbolTable::exit_scope() {
    map_table.pop_back();
    level--;
}
```

一些原始的寄存器分配。

```
inline void SymbolTable::new_regs(int id) {
    if (id < 0 || id > 5) {
        fprintf(stderr, "Invalid register id!\n");
        exit(1);
    }
    if (regs_used[id] > 0) {
        assembly_code.push_back("\tpush  " + param_regs[id]);
        push_cnt++;
    }
    regs_used[id]++;
}

inline void SymbolTable::free_regs(int id) {
    if (id < 0 || id > 5) {
        fprintf(stderr, "Invalid register id!\n");
        exit(1);
    }
    regs_used[id]--;
    if (regs_used[id] > 0) {
        assembly_code.push_back("\tpop  " + param_regs[id]);
        push_cnt--;
    }
}
```

为了实现简单，规定函数调用时由 `caller` 保存寄存器， `callee` 不保存寄存器。

```
inline void SymbolTable::save_regs() {
    for (int i = 0; i < param_reg_num; i++) {
        if (regs_used[i] > 0) {
            assembly_code.push_back("\tpush  " + param_regs[i]);
            push_cnt++;
        }
    }
}

inline void SymbolTable::restore_regs() {
    for (int i = param_reg_num - 1; i >= 0; i--) {
        if (regs_used[i] > 0) {
            assembly_code.push_back("\tpop  " + param_regs[i]);
            push_cnt--;
        }
    }
}
```

条件分支跳转，栈指针平衡等实现在上机课验收时已介绍，此处不再赘述。

关于复杂数组的初始化，采用递归的方式实现，每一维度维护偏移量与对齐。

```
// handle initialization of array like this : int arr[3][2][2] = {1,2,3,4,{ {6} ,-1,-2 },{9,10}}
int CodeGenerator::handle_arr_initval(shared_ptr<SymbolTable> table, shared_ptr<ArraySymbol> arr
    int max_size = 1;

    for (int i = dim; i < array->dim_size.size(); i++)
        max_size *= array->dim_size[i];

    int next_align_size = dim < array->dim_size.size() ? max_size / array->dim_size[dim] : 1;

    if (init_val->varKind == VarKind::Var) {
        // fill the array with exp value
        pair<bool, int> res = handle_exp(table, init_val->exp);

        if (array->kind == ArraySymbol::ArrayKind::GLOBAL_INT) {
            if (res.first)
                global_data.push_back("\t.long  " + to_string(res.second));
            else {
                fprintf(stderr, "Error: %s\n", "Initialize global array with non-const exp");
                exit(1);
            }
        }
        else if (array->kind == ArraySymbol::ArrayKind::PARAM_PTR) {
            fprintf(stderr, "Error: %s\n", "Should not initialize a parameter array");
            exit(1);
        }
        else if (array->kind == ArraySymbol::ArrayKind::GLOBAL_CONST) {
            if (res.first)
                rdata.push_back("\t.long  " + to_string(res.second));
            else {
                fprintf(stderr, "Error: %s\n", "Initialize global const array with non-const exp
                exit(1);
            }
        }
        else {
            if (res.first) {
                table->assembly_code.push_back("\tmov  dword ptr [rbp" + my_to_string(array->off
                if (array->kind == ArraySymbol::ArrayKind::CONST_INT)
                    array->const_val.push_back(res.second);
            }
            else {
                table->assembly_code.push_back("\tmov  dword ptr [rbp" + my_to_string(array->off
                if (array->kind == ArraySymbol::ArrayKind::CONST_INT) {
                    fprintf(stderr, "Error: %s\n", "Initialize const array with non-const exp");
```

```
                exit(1);
            }
        }
    }
    return 1;
}

else {
    if (init_val->initValList == nullptr) {
        // fill this dimension with 0, using rep stosq
        if (array->kind == ArraySymbol::ArrayKind::GLOBAL_INT || array->kind == ArraySymbol
            global_data.push_back("\t.zero  " + to_string(max_size * 4));
        else {
            fill_zero(table, array->offset, offset, max_size);
            if (array->kind == ArraySymbol::ArrayKind::CONST_INT) {
                for (int i = 0; i < max_size; i++)
                    array->const_val.push_back(0);
            }
        }
        return max_size;
    }
    else {
        if (dim >= array->dim_size.size() && init_val->initValList->initVals.size() > 1) {
            fprintf(stderr, "Error: %s\n", "Initialization of array with too many dimensions
            exit(1);
        }
        int filled_size = 0;
        for (auto init_val_it : init_val->initValList->initVals) {
            if (filled_size >= max_size) {
                fprintf(stderr, "Error: %s\n", "Initialization of array with too many argume
                exit(1);
            }
            if (init_val_it->varKind == VarKind::Var) {
                // still in the same dimension!
                filled_size += handle_arr_initval(table, array, init_val_it, dim, offset + f
            }
            else {
                int next_depth = array->dim_size.size() - 1;
                int temp_filled = filled_size;
                while (next_depth > dim) {
                    if (temp_filled % array->dim_size[next_depth])
                        break;
                    else {
```

```
                        temp_filled /= array->dim_size[next_depth];
                        next_depth--;
                    }
                }

                if (next_depth == array->dim_size.size() - 1) {
                    fprintf(stderr, "Error: %s\n", "Initialization of array not aligned");
                    exit(1);
                }

                int next_filled = handle_arr_initval(table, array, init_val_it, next_depth -
                if (next_filled > next_align_size) {
                    fprintf(stderr, "Error: %s\n", "Initialization of array with too many ar
                    exit(1);
                }
                // align
                if (next_filled < next_align_size) {
                    // fill with 0
                    if (array->kind == ArraySymbol::ArrayKind::GLOBAL_INT)
                        global_data.push_back("\t.zero  " + to_string((next_align_size - nex
                    else {
                        fill_zero(table, array->offset, offset + filled_size + next_filled,
                        if (array->kind == ArraySymbol::ArrayKind::CONST_INT) {
                            for (int i = 0; i < next_align_size - next_filled; i++)
                                array->const_val.push_back(0);
                        }
                    }
                }
                filled_size += next_align_size;
            }
        }
        if (filled_size < max_size) {
            if (array->kind == ArraySymbol::ArrayKind::GLOBAL_INT)
                global_data.push_back("\t.zero  " + to_string((max_size - filled_size) * 4))
            else
                fill_zero(table, array->offset, offset + filled_size, max_size - filled_size
        }
        return max_size;
    }
}
}
```

写入文件。

```
void CodeGenerator::dump(shared_ptr<CompUnit> comp_unit, string filename) {
    traverse(global_table, comp_unit);

    // output assembly code to file

    ofstream out_file;
    out_file.open(filename);

    out_file << ".intel_syntax noprefix" << endl;

    out_file << endl;

    out_file << ".section .rodata" << endl;
    for (auto code : rdata)
        out_file << code << endl;

    out_file << endl;

    out_file << ".section .data" << endl;
    for (auto code : global_data)
        out_file << code << endl;

    out_file << endl;

    out_file << ".section .text" << endl;
    for (auto name : text)
        out_file << ".globl  " << name << endl;

    out_file << endl;

    for (auto code : global_table->assembly_code)
        out_file << code << endl;

    for (auto table : tables) {
        // align the stack pointer to 16 bytes
        int stack_ptr = table->stack_ptr;
        stack_ptr *= -1;
        stack_ptr = (stack_ptr + 15) / 16 * 16;
        table->header_code.push_back("\tsub  rsp, " + to_string(stack_ptr));
        for (auto head : table->header_code)
            out_file << head << endl;
        for (auto code : table->assembly_code)
            out_file << code << endl;
```

```
        for (auto foot : table->footer_code)
            out_file << foot << endl;
    }

    out_file.close();
}
```

3. 实验总结

本次实验实现了简单的语义分析，通过抽象语法树的遍历，实现了符号表的维护，以及对不同类型节点的处理，生成了 `x86-64 intel` 汇编代码。

尽管通过了所有测试用例，但仍有一些未实现的功能。

- 不支持浮点数与长整数
- 不支持显式指针运算，指针仅能用于参数传递
- 不支持 `bool` 转 `int`
- 不支持动态内存分配