Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный** **исследовательский политехнический университет»**

Факультет: Прикладной математики и механики

Кафедра: Вычислительной математики, механики и биомеханики

Направление: 09.03.02 «Информационные системы и технологии»

Программа: «Цифровые технологии и интеллектуальные системы управления»

У Т В Е Р Ж Д А Ю

**Зав. кафедрой ВММБ**

\_\_\_\_\_\_\_\_\_\_\_\_\_ В.Ю. Столбов

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ г.

***ЗАДАНИЕ***

***НА КУРСОВУЮ РАБОТУ***

***по дисциплине***

**«Языки программирования и методы трансляции»**

Сукерт Милена Константиновна ИСТ-22-1б

(фамилия, имя, отчество студента; группа)

1. **Тема курсовой работы**

Язык программирования для управления умным домом

2. **Срок сдачи студентом отчета:**

28.12.2023

3. **Содержание отчета:**

В курсовой работе были изучены грамматика и принципы работы лексического

анализатора flex и синтаксического анализатора bison, а также с их помощью был

разработан язык программирования для управления умным домом ,в котором продумана

логика if else, AST и взаимодействие с ним через root, также

был проведен анализ ограничений и возможностей созданного ЯП.

С.Е. Батин

Руководитель курсовой работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (расшифровка)

Сукерт М.К.

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (расшифровка)

Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный** **исследовательский**

**политехнический университет»**

Факультет: Прикладной математики и механики

Кафедра: Вычислительной математики, механики и биомеханики

Направление: 09.03.02 «Информационные системы и технологии»

Программа: «Цифровые технологии и интеллектуальные системы управления»

**КУРСОВАЯ РАБОТА**

по дисциплине

**«Языки программирования и методы трансляции»**

Тема: **«Язык управления умным домом»**

Выполнил:

ИСТ-22-1б

студент гр.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Сукерт М.К.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Ф.И.О.)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*(подпись, дата)*

Принял:

Доцент каф. ВММБ, к.т.н. С.Е. Батин

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(должность, ФИО)*

*\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(оценка) (подпись)*

*\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(дата)*

**Пермь, 2023**

# Введение

Данная курсовая работа направлена на создание языка программирования с помощью инструментов flex и bison. Данные инструменты используются для создания лексического и синтаксического анализаторов.

Работа должна содержать описание грамматики языка, инструменты реализации грамматики (вышеупомянутые flex и bison), исходные коды лексического и синтаксического анализаторов, а также ограничения, связанные с работой языка программирования.

Результатом работы будет являться свой язык программирования для умного дома.

Лексический анализатор

Термин "flex" часто используется для обозначения лексического анализатора, который является частью инструментов компиляторов. Flex (Fast Lexical Analyzer Generator) — это программный инструмент для автоматической генерации лексических анализаторов.

Лексический анализатор преобразует входной текст программы в последовательность лексем (токенов), которые затем обрабатываются синтаксическим анализатором. Flex работает на основе спецификаций в виде регулярных выражений и соответствующих действий. Он генерирует код лексического анализатора на языке программирования C или C++.

Пример использования Flex:

Пользователь определяет правила сопоставления лексем с использованием регулярных выражений.

Flex генерирует соответствующий код лексического анализатора, который может быть интегрирован в компилятор.

Использование инструментов, таких как Flex, позволяет разработчикам упростить процесс создания лексических анализаторов и улучшить поддержку для различных языков программирования.

# Синтаксический анализатор

Bison — это генератор синтаксических анализаторов, который работает в паре с лексическим анализатором Flex. Он широко используется в компиляторах и других инструментах обработки языков программирования. Bison генерирует синтаксические анализаторы на основе заданных пользователем правил грамматики.

Вот основные шаги по созданию синтаксического анализатора с использованием Bison:

**Определение грамматики:** Пользователь определяет грамматику своего языка программирования с использованием формальных правил, которые описывают структуру языка. Грамматика обычно записывается в форме контекстно-свободной грамматики.

**Создание файла с правилами Bison:** В этом файле пользователь использует специальный синтаксис для определения правил грамматики. Эти правила включают в себя определения нетерминалов, терминалов и действий, которые выполняются при распознавании конкретных конструкций.

**Генерация синтаксического анализатора:** Bison обрабатывает файл с правилами и генерирует код синтаксического анализатора на языке программирования C или C++.

**Интеграция с лексическим анализатором:** Полученный синтаксический анализатор интегрируется с лексическим анализатором, который был создан с использованием Flex.

**Разработка и отладка:** После генерации кода разработчик может дополнить его дополнительными функциями и провести отладку.

Пример грамматики для Bison может выглядеть следующим образом:

program: statement\_list

;

statement\_list: statement

| statement\_list statement

;

statement: create\_ast\_statement SEMICOLON

| expression\_statement SEMICOLON

| light\_command SEMICOLON

| conder\_command SEMICOLON

| status\_command SEMICOLON

| set\_temperature\_statement SEMICOLON

| print\_statement SEMICOLON

| if\_else\_statement

| variable\_declaration SEMICOLON

| assignment\_statement SEMICOLON

;

Еще в грамматике yacc описана с использование варианта формы Бекуса-Наура (BNF). Это техника, разработанная Джоном Бэкусом и Питером Науром. (рис. 1)

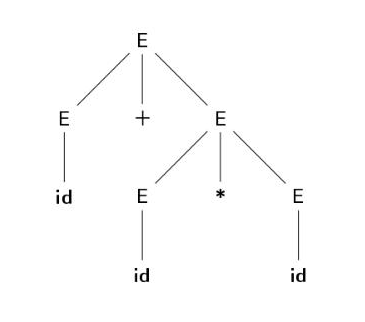


Рис 1. Дерево разбора

Дерево разбора (или синтаксическое дерево) в программировании представляет собой структуру данных, которая отражает синтаксическую структуру входного кода в соответствии с определенной грамматикой. Это дерево служит визуальным представлением иерархии элементов программы, обеспечивая более удобное представление для дальнейшего анализа и обработки.

Процесс построения дерева разбора обычно выполняется с использованием синтаксического анализатора (парсера), который анализирует входные данные (например, исходный код программы) в соответствии с определенной грамматикой языка программирования. Дерево разбора состоит из узлов, представляющих синтаксические элементы кода, и ребер, отображающих связи между этими элементами.

Деревья разбора используются в компиляторах и интерпретаторах для представления синтаксической структуры программы. После построения дерева разбора происходит следующий этап анализа (семантический анализ, оптимизация, генерация кода), который использует информацию, полученную из этой структуры, для дальнейших шагов обработки программы.

# Грамматика языка

В файле input.txt записан разработанный псевдокод.

1)

create\_ast "room";

"room".status();

"room".set\_temperature(23);

"room".light\_on();

**if** ("room".temperature == 23) {

"room".light\_off();

"room".set\_temperature(20);

} **else** {

"room".light\_off();

}

"room".status();

2)

create\_ast "room";

"room".status();

"room".set\_temperature(23);

"room".light\_on();

**if** ("room".temperature == 23) {

"room".light\_off();

"room".set\_temperature(20);

} **else** {

"room".light\_off();

}

"room".status();

int A == 3;

print(A \* 5);

После компилирования программы следующими командами:

flex lexer.l

bison -d parser.y

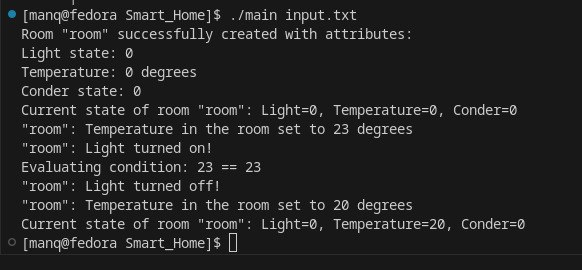
gcc -o main parser.tab.c lex.yy.c struct\_func.c -lm

./main input.txt

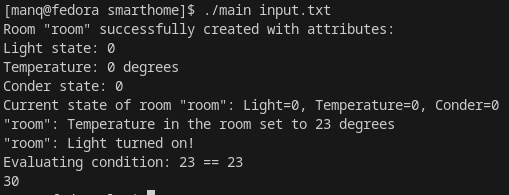
(не обращая внимания на предупреждения)

Мы получаем следующий вывод в консоль:

1)



2)



Реализована работа с таблицей символов и реализованы переменные и операции над ними:

*// Symbol Table*

**typedef** **struct** {

char\* name;

int type; *// 0 for int, 1 for string*

int intValue;

char\* stringValue;

} Symbol;

Symbol symbolTable[100];

int symbolCount = 0;

void addSymbol(char\* name, int type, int intValue, char\* stringValue) {

Symbol symbol;

symbol.name = name;

symbol.type = type;

symbol.intValue = intValue;

symbol.stringValue = stringValue;

symbolTable[symbolCount++] = symbol;

}

Symbol\* findSymbol(char\* name) {

**for** (int i = 0; i < symbolCount; i++) {

**if** (strcmp(symbolTable[i].name, name) == 0) {

**return** &symbolTable[i];

}

}

**return** NULL;

}

assignment\_statement:

ID EQUAL expression {

Symbol\* symbol = findSymbol($1);

**if** (symbol != NULL) {

**if** (symbol->type == 0) {

symbol->intValue = $3;

} **else** {

fprintf(stderr, "Error: Cannot assign integer value to string variable '%s'**\n**", $1);

exit(EXIT\_FAILURE);

}

} **else** {

fprintf(stderr, "Error: Variable '%s' not found**\n**", $1);

exit(EXIT\_FAILURE);

}

}

;

variable\_declaration:

INT\_TYPE ID EQUAL INTEGER {

addSymbol($2, 0, $4, NULL);

}

| STRING\_TYPE ID EQUAL STRING\_VALUE {

addSymbol($2, 1, 0, $4);

}

;

# Анализ ограничений

В данном языке не реализованы переменные, конструкции while и for которые бы значительно расширили возможности языка.

# Заключение

Был создан язык программирования для управления умным домом. Язык поддерживает взаимодействия с устройствами и базовую конструкцию if else.

# Список литературы

[Lex & Yacc](https://books.google.co.uk/books?id=fMPxfWfe67EC&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false)

By John R. Levine, Tony Mason, Doug Brown

[Compilers: Principles, Techniques, and Tools](https://api.pageplace.de/preview/DT0400.9781292037233_A24581712/preview-9781292037233_A24581712.pdf)

by Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman

**Приложение**

Файлы проекта :

[parser.y](https://github.com/MilenaSkrt/smarthome/blob/main/parser.y)

%{

#include <stdio.h>

#include <stdlib.h>

#include "struct\_func.h"

#include <string.h>

#include "parser.tab.h"

bool in\_false\_if\_block = false;

**extern** FILE\* yyin;

**extern** int yylex();

**extern** char\* yytext;

**struct** AST;

AST\* root = NULL;

CommandList\* current\_command\_list = NULL;

void yyerror(**const** char\* s);

*// Symbol Table*

**typedef** **struct** {

char\* name;

int type; *// 0 for int, 1 for string*

int intValue;

char\* stringValue;

} Symbol;

Symbol symbolTable[100];

int symbolCount = 0;

void addSymbol(char\* name, int type, int intValue, char\* stringValue) {

Symbol symbol;

symbol.name = name;

symbol.type = type;

symbol.intValue = intValue;

symbol.stringValue = stringValue;

symbolTable[symbolCount++] = symbol;

}

Symbol\* findSymbol(char\* name) {

**for** (int i = 0; i < symbolCount; i++) {

**if** (strcmp(symbolTable[i].name, name) == 0) {

**return** &symbolTable[i];

}

}

**return** NULL;

}

%}

%lex-param { YYSTYPE yylval; }

%**union** {

int intval;

char\* strval;

AST\* objectval;

Condition\* conditionval;

Block\* blockval;

}

%token <intval> INTEGER

%token <objectval> CREATE\_AST IF ELSE TURN\_ON TURN\_OFF

%token COLON SEMICOLON LPAREN RPAREN LBRACE RBRACE COMMA DOT SET\_TEMPERATURE PRINT

%token <strval> STRING\_VALUE

%token INT\_TYPE STRING\_TYPE

%token <strval> LIGHT\_ON LIGHT\_OFF CONDER\_ON CONDER\_OFF STATUS GREATER LESS EQUAL STRING ID

%type <intval> expression SET\_TEMPERATURE attribute argument\_list

%type <strval> attribute\_name relation\_operator

%type <objectval> create\_ast\_statement light\_command conder\_command status\_command object expression\_statement

%type <conditionval> condition

%type <blockval> statement\_list

%%

program: statement\_list

;

statement\_list: statement

| statement\_list statement

;

statement: create\_ast\_statement SEMICOLON

| expression\_statement SEMICOLON

| light\_command SEMICOLON

| conder\_command SEMICOLON

| status\_command SEMICOLON

| set\_temperature\_statement SEMICOLON

| print\_statement SEMICOLON

| if\_else\_statement

| variable\_declaration SEMICOLON

| assignment\_statement SEMICOLON

;

assignment\_statement:

ID EQUAL expression {

Symbol\* symbol = findSymbol($1);

**if** (symbol != NULL) {

**if** (symbol->type == 0) {

symbol->intValue = $3;

} **else** {

fprintf(stderr, "Error: Cannot assign integer value to string variable '%s'**\n**", $1);

exit(EXIT\_FAILURE);

}

} **else** {

fprintf(stderr, "Error: Variable '%s' not found**\n**", $1);

exit(EXIT\_FAILURE);

}

}

;

variable\_declaration:

INT\_TYPE ID EQUAL INTEGER {

addSymbol($2, 0, $4, NULL);

}

| STRING\_TYPE ID EQUAL STRING\_VALUE {

addSymbol($2, 1, 0, $4);

}

;

create\_ast\_statement: CREATE\_AST STRING { $$ = create\_ast($2); root = $$; }

;

light\_command: object DOT LIGHT\_ON LPAREN RPAREN {

add\_cmd\_list(light\_on, getast($1), 0);

}

| object DOT LIGHT\_OFF LPAREN RPAREN {

add\_cmd\_list(light\_off, getast($1), 0);

}

;

conder\_command: object DOT CONDER\_ON LPAREN RPAREN { add\_cmd\_list(conder\_on, getast($1), 0); }

| object DOT CONDER\_OFF LPAREN RPAREN { add\_cmd\_list(conder\_off, getast($1), 0); }

;

set\_temperature\_statement: object DOT SET\_TEMPERATURE LPAREN INTEGER RPAREN { add\_cmd\_list(set\_temp, getast($1), $5) ;};

status\_command: object DOT STATUS LPAREN RPAREN {add\_cmd\_list(print\_ast\_state, getast($1), 0) ; };

expression\_statement: object DOT attribute\_name LPAREN argument\_list RPAREN SEMICOLON

{

execute\_method($1, $3, $5);

}

print\_statement: PRINT LPAREN attribute RPAREN {add\_cmd\_list(print\_attribute, $3 , 0) ;}

;

condition: LPAREN expression relation\_operator expression RPAREN

{

$$ = create\_condition($2, $3, $4);

}

;

relation\_operator: GREATER

| LESS

| EQUAL

;

if\_else\_statement:

{

execute\_command\_list();

}

IF condition LBRACE

{

bool condition\_result = evaluate($3);

in\_false\_if\_block = !condition\_result;

}

statement\_list RBRACE ELSE LBRACE

{

in\_false\_if\_block = !in\_false\_if\_block;

}

statement\_list RBRACE

{

in\_false\_if\_block = false;

}

|

{

execute\_command\_list();

}

IF condition LBRACE

{

bool condition\_result = evaluate($3);

in\_false\_if\_block = !condition\_result;

}

statement\_list RBRACE

{

in\_false\_if\_block = false;

}

;

argument\_list: expression

| argument\_list COMMA expression

;

expression: INTEGER

| ID

| object DOT attribute\_name LPAREN argument\_list RPAREN { $$ = (AST\*)$1; }

| attribute

| condition

;

object: STRING { $$ = getast($1); }

;

attribute\_name: ID

;

attribute: object DOT ID { $$ = get\_attribute\_value(root, $3); }

;

%%

void yyerror(**const** char\* s) {

fprintf(stderr, "Error: %s near '%s'**\n**", s, yytext);

exit(EXIT\_FAILURE);

}

int main(int argc, char\* argv[]) {

**if** (argc != 2) {

fprintf(stderr, "Usage: %s input\_file**\n**", argv[0]);

exit(EXIT\_FAILURE);

}

FILE\* input\_file = fopen(argv[1], "r");

**if** (!input\_file) {

perror("Error opening input file");

exit(EXIT\_FAILURE);

}

yyin = input\_file;

AST\* obj = NULL;

yyparse();

execute\_command\_list();

fclose(input\_file);

**return** 0;

}

[struct\_func.c](https://github.com/MilenaSkrt/smarthome/blob/main/struct_func.c)

#include "struct\_func.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

Block if\_block = NULL;

Block else\_block = NULL;

bool last\_condition\_result = false;

Variable variables[MAX\_VARIABLES];

int variableCount = 0;

#define MAX\_COMMANDS 100

Command commandList[MAX\_COMMANDS];

int commandCount = 0;

AST\* objects[MAX\_OBJECTS];

int objectCount = 0;

AST\* getast(**const** char\* name) {

**for** (int i = 0; i < objectCount; i++) {

**if** (strcmp(objects[i]->name, name) == 0) {

**return** objects[i];

}

}

**if** (objectCount < MAX\_OBJECTS) {

AST\* newObj = create\_ast(name);

objects[objectCount++] = newObj;

**return** newObj;

}

fprintf(stderr, "Maximum number of objects exceeded**\n**");

**return** NULL;

}

*// Это перменные*

*// Функции для работы с переменными*

void add\_int\_variable(**const** char\* name, int value) {

**if** (variableCount >= MAX\_VARIABLES) {

fprintf(stderr, "Превышено максимальное количество переменных**\n**");

**return**;

}

Variable\* var = &variables[variableCount++];

strncpy(var->name, name, **sizeof**(var->name) - 1);

var->type = INT;

var->int\_value = value;

}

void add\_string\_variable(**const** char\* name, **const** char\* value) {

**if** (variableCount >= MAX\_VARIABLES) {

fprintf(stderr, "Превышено максимальное количество переменных**\n**");

**return**;

}

Variable\* var = &variables[variableCount++];

strncpy(var->name, name, **sizeof**(var->name) - 1);

var->type = VAR\_STRING\_TYPE;

var->str\_value = strdup(value);

}

void print\_variable(Variable\* var) {

**if** (var->type == INT) {

printf("%d**\n**", var->int\_value);

} **else** **if** (var->type == VAR\_STRING\_TYPE) {

printf("%s**\n**", var->str\_value);

}

}

void print\_expression\_result(int result) {

printf("%d**\n**", result);

}

void set\_int\_variable(**const** char\* name, int value) {

**for** (int i = 0; i < variableCount; ++i) {

**if** (strcmp(variables[i].name, name) == 0 && variables[i].type == INT) {

variables[i].int\_value = value;

**return**;

}

}

fprintf(stderr, "Int variable '%s' not found**\n**", name);

}

void set\_string\_variable(**const** char\* name, **const** char\* value) {

**for** (int i = 0; i < variableCount; ++i) {

**if** (strcmp(variables[i].name, name) == 0 && variables[i].type == VAR\_STRING\_TYPE) {

free(variables[i].str\_value);

variables[i].str\_value = strdup(value);

**return**;

}

}

fprintf(stderr, "String variable '%s' not found**\n**", name);

}

Variable\* get\_variable(**const** char\* name) {

**for** (int i = 0; i < variableCount; ++i) {

**if** (strcmp(variables[i].name, name) == 0) {

**return** &variables[i];

}

}

**return** NULL;

}

void add\_cmd\_list(void (\*execute)(AST\*, int), AST\* obj, int arg) {

**if** (!in\_false\_if\_block) {

**if** (commandCount >= MAX\_COMMANDS) {

fprintf(stderr, "Maximum number of commands exceeded**\n**");

**return**;

}

commandList[commandCount].execute = execute;

commandList[commandCount].object = obj;

commandList[commandCount].arg = arg;

commandCount++;

}

}

void execute\_command\_list() {

**for** (int i = 0; i < commandCount; i++) {

Command cmd = commandList[i];

cmd.execute(cmd.object, cmd.arg);

}

commandCount = 0;

}

AST\* create\_ast(**const** char\* name) {

**for** (int i = 0; i < objectCount; i++) {

**if** (strcmp(objects[i]->name, name) == 0) {

**return** objects[i];

}

}

AST\* obj = (AST\*)malloc(**sizeof**(AST));

**if** (!obj) {

perror("Memory allocation error");

exit(EXIT\_FAILURE);

}

strncpy(obj->name, name, **sizeof**(obj->name) - 1);

obj->name[**sizeof**(obj->name) - 1] = '\0';

obj->light\_state = 0;

obj->thermostat\_temperature = 0;

obj->conder = 0;

**if** (objectCount < MAX\_OBJECTS) {

objects[objectCount++] = obj;

} **else** {

fprintf(stderr, "Maximum number of objects exceeded**\n**");

free(obj);

**return** NULL;

}

printf("Room %s successfully created with attributes:**\n**", obj->name);

printf("Light state: %d**\n**", obj->light\_state);

printf("Temperature: %d degrees**\n**", obj->thermostat\_temperature);

printf("Conder state: %d**\n**", obj->conder);

**return** obj;

}

void print\_ast\_state(AST\* obj) {

**if** (obj != NULL) {

printf("Current state of room %s: Light=%d, Temperature=%d, Conder=%d**\n**", obj->name, obj->light\_state, obj->thermostat\_temperature, obj->conder);

} **else** {

printf("Error: Object is NULL**\n**");

}

}

void light\_on(AST\* obj) {

obj->light\_state = 1;

printf("%s: Light turned on!**\n**", obj->name);

}

void light\_off(AST\* obj) {

obj->light\_state = 0;

printf("%s: Light turned off!**\n**", obj->name);

}

void set\_temp(AST\* obj, int temperature) {

obj->thermostat\_temperature = temperature;

printf("%s: Temperature in the room set to %d degrees**\n**", obj->name, temperature);

}

void conder\_on(AST\* obj) {

obj->conder = 1;

printf("%s: Conder turned on**\n**", obj->name);

}

void conder\_off(AST\* obj) {

obj->conder = 0;

printf("%s: Conder turned off**\n**", obj->name);

}

void print\_attribute(int attribute\_value) {

printf("%d**\n**", attribute\_value);

}

int get\_attribute\_value(AST\* obj, **const** char\* attribute\_name) {

**if** (strcmp(attribute\_name, "temperature") == 0) {

**return** obj->thermostat\_temperature;

} **else** **if** (strcmp(attribute\_name, "light\_state") == 0) {

**return** obj->light\_state;

}

}

void execute\_method(AST\* obj, **const** char\* method\_name, int argument) {

**if** (strcmp(method\_name, "light\_on") == 0) {

light\_on(obj);

} **else** **if** (strcmp(method\_name, "light\_off") == 0) {

light\_off(obj);

} **else** **if** (strcmp(method\_name, "conder\_on") == 0) {

conder\_on(obj);

} **else** **if** (strcmp(method\_name, "conder\_off") == 0) {

conder\_off(obj);

} **else** **if** (strcmp(method\_name, "status") == 0) {

print\_ast\_state(obj);

} **else** **if** (strcmp(method\_name, "set\_temperature") == 0) {

set\_temp(obj, argument);

} **else** {

fprintf(stderr, "Error: Unknown method '%s'**\n**", method\_name);

exit(EXIT\_FAILURE);

}

}

void if\_else(Condition\* condition, Block if\_block, Block else\_block) {

**if** (if\_block != NULL) {

if\_block();

} **else** {

fprintf(stderr, "Error: if\_block is NULL**\n**");

exit(EXIT\_FAILURE);

}

**if** (else\_block != NULL) {

else\_block();

} **else** {

fprintf(stderr, "Error: else\_block is NULL**\n**");

exit(EXIT\_FAILURE);

}

}

bool evaluate(**const** Condition\* condition) {

**if** (condition != NULL) {

int left\_value = condition->left\_value;

int right\_value = condition->right\_value;

char\* operator = condition->operator;

**if** (operator == NULL) {

printf("Operator is not specified**\n**");

**return** false;

}

printf("Evaluating condition: %d %s %d**\n**", left\_value, operator, right\_value);

**if** (strcmp(operator, ">") == 0) {

**return** left\_value > right\_value;

} **else** **if** (strcmp(operator, "<") == 0) {

**return** left\_value < right\_value;

} **else** **if** (strcmp(operator, "==") == 0) {

**return** left\_value == right\_value;

}

}

**return** false;

}

void free\_condition(Condition\* condition) {

free(condition->attribute);

free(condition->operator);

free(condition);

}

Condition\* create\_condition(int left\_value, char\* operator, int right\_value) {

Condition\* condition = (Condition\*)malloc(**sizeof**(Condition));

**if** (condition == NULL) {

perror("Memory allocation error");

exit(EXIT\_FAILURE);

}

condition->left\_value = left\_value;

condition->right\_value = right\_value;

condition->operator = strdup(operator);

**return** condition;

}

void execute\_block\_list(Block\* block\_list) {

**if** (!block\_list) {

**return**;

}

**while** (\*block\_list != NULL) {

(\*block\_list)();

block\_list++;

}

}

CommandList\* create\_command\_list() {

CommandList\* list = (CommandList\*)malloc(**sizeof**(CommandList));

**if** (list == NULL) {

fprintf(stderr, "Error: Memory allocation failed for CommandList**\n**");

exit(EXIT\_FAILURE);

}

list->commands = NULL;

list->count = 0;

**return** list;

}

Condition\* create\_condition\_attribute(AST\* object, **const** char\* attribute\_name, char\* operator, int right\_value) {

Condition\* condition = (Condition\*)malloc(**sizeof**(Condition));

**if** (condition == NULL) {

perror("Memory allocation error");

exit(EXIT\_FAILURE);

}

condition->object = object;

condition->attribute = strdup(attribute\_name);

condition->operator = strdup(operator);

condition->right\_value = right\_value;

**return** condition;

}

[struct\_func.h](https://github.com/MilenaSkrt/smarthome/blob/main/struct_func.h)

#include <stdbool.h>

#ifndef STRUCT\_FUNC\_H

#define STRUCT\_FUNC\_H

#define MAX\_OBJECTS 100

#define MAX\_VARIABLES 50

**typedef** **struct** {

char name[50];

int light\_state;

int thermostat\_temperature;

int conder;

} AST;

**typedef** void (\*Block)();

**extern** bool last\_condition\_result;

**extern** AST\* objects[MAX\_OBJECTS];

**extern** int objectCount;

**typedef** void (\*CommandFunction)(AST\* obj);

**typedef** **enum** { INT, VAR\_STRING\_TYPE } VarType;

**typedef** **struct** {

char name[50];

VarType type;

**union** {

int int\_value;

char\* str\_value;

};

} Variable;

**extern** Variable variables[MAX\_VARIABLES];

**extern** int variableCount;

**typedef** **struct** Command {

void (\*execute)(AST\*, int); *// Обновите указатель на функцию, чтобы он мог принимать аргумент*

AST\* object; *// Указатель на объект*

int arg; *// Аргумент для команды*

} Command;

**typedef** **struct** CommandList {

Command\* commands; *// Массив команд*

int count; *// Количество команд в списке*

} CommandList;

**typedef** **struct** {

AST\* object; *// Указатель на объект (если требуется)*

char\* attribute; *// Название атрибута объекта (если требуется)*

int left\_value; *// Левый операнд для сравнения*

int right\_value; *// Правый операнд для сравнения*

char\* operator; *// Оператор сравнения*

} Condition;

**extern** AST\* current\_object;

**extern** Block if\_block;

**extern** Block else\_block;

**extern** bool in\_false\_if\_block;

void add\_int\_variable(**const** char\* name, int value);

void add\_string\_variable(**const** char\* name, **const** char\* value);

void set\_int\_variable(**const** char\* name, int value);

void set\_string\_variable(**const** char\* name, **const** char\* value);

Variable\* get\_variable(**const** char\* name);

void print\_variable(Variable\* var);

void print\_expression\_result(int result);

void add\_cmd(CommandFunction func, AST\* obj);

void execute\_commands();

void clear\_commands();

void execute\_block\_list(Block\* block\_list);

AST\* getast(**const** char\* name);

void execute\_if\_else(Condition\* condition, Block if\_block, Block else\_block);

void execute\_block(**const** char\* command, AST\* obj);

void add\_cmd\_list(void (\*execute)(AST\*, int), AST\* obj, int arg);

void print\_ast\_state(AST\* obj);

AST\* create\_ast(**const** char\* name);

void print\_attribute(int attribute\_value);

void light\_on(AST\* obj);

bool evaluate(**const** Condition\* condition);

void execute\_method(AST\* obj, **const** char\* method\_name, int argument) ;

int get\_attribute\_value(AST\* obj, **const** char\* attribute\_name);

void light\_off(AST\* obj);

void set\_temp(AST\* obj, int temperature);

void conder\_on(AST\* obj);

void conder\_off(AST\* obj);

Condition\* create\_condition(int left\_value, char\* operator, int right\_value);

Condition\* create\_condition\_attribute(AST\* object, **const** char\* attribute\_name, char\* operator, int right\_value);

Condition\* create\_condition\_object(**const** char\* object\_name);

Condition\* create\_condition\_time(**const** char\* time\_expression);

#endif

[lexer.l](https://github.com/MilenaSkrt/smarthome/blob/main/lexer.l)

%{

#include "struct\_func.h"

#include "parser.tab.h"

**struct** AST;

%}

**struct** AST;

%%

\"[^"]\*\" { yylval.strval = strdup(yytext); return STRING; }

[0-9]+ { yylval.intval = atoi(yytext); **return** INTEGER; }

"create\_ast" { yylval.strval = strdup(yytext); **return** CREATE\_AST; }

"if" { **return** IF; }

"int" { **return** INT\_TYPE; }

"string" { **return** STRING\_TYPE; }

"else" { **return** ELSE; }

"light\_on" { **return** LIGHT\_ON; }

"light\_off" { **return** LIGHT\_OFF; }

"conder\_on" { **return** CONDER\_ON; }

"conder\_off" { **return** CONDER\_OFF; }

"status" { **return** STATUS; }

"set\_temperature" { **return** SET\_TEMPERATURE; }

"print" { **return** PRINT; }

"(" { **return** LPAREN; }

")" { **return** RPAREN; }

"{" { **return** LBRACE; }

"}" { **return** RBRACE; }

">" { yylval.strval = strdup(yytext); **return** GREATER; }

"<" { yylval.strval = strdup(yytext); **return** LESS; }

"==" { yylval.strval = strdup(yytext); **return** EQUAL; }

";" { **return** SEMICOLON; }

[a-zA-Z\_][a-zA-Z0-9\_]\* { yylval.strval = strdup(yytext); **return** ID; }

[ \t\n] ;

. { **return** DOT; }

%%

int yywrap() {

**return** 1;

}