Оглавление

[1. Введение 2](#_Toc92900749)

[1.1. Цель и задачи работы 2](#_Toc92900750)

[2. Техническая постановка 2](#_Toc92900751)

[2.1. Выбор средств реализации 2](#_Toc92900752)

[2.2. Минимальные требования к системе команд 2](#_Toc92900753)

[3. Реализация 2](#_Toc92900754)

[3.1. Список команд 2](#_Toc92900755)

[3.2. Логическая схема 2](#_Toc92900756)

[3.3. Ограничения языка 2](#_Toc92900757)

[4. Тестирование 2](#_Toc92900758)

[5. Заключение 2](#_Toc92900759)

[Список литературы 2](#_Toc92900760)

[Приложение 2](#_Toc92900761)

# Введение

Современная автоматизированная техника не обходится без языков программирования. Так или иначе, различные системы требуют контроля. Ранее этот контроль осуществлялся человеком, сегодня же для этого создаются различного вида программы. Однако автоматизация – это лишь одно из множества направлений применения языков программирования.

Язык программирования – формальный язык, инструмент, при помощи которого можно использовать мощности компьютера. При взаимодействии этих двух персонажей возникает проблема – машина понимает только бинарный код, а человек не хочет страдать, разбирая такой код. Любой язык выше бинарного кода нуждается в трансляторе, который, в свою очередь, состоит из собственной лексики и синтаксиса. Лексический анализатор в первой фазе компиляции читает входящий поток символов, разбивает его на специализированные лексемы и строит для каждой выходной токен. Далее, во второй фазе компиляции, в дело вступает синтаксический анализатор – он проверяет наличие синтаксических конструкций языка программирования и выполняет семантические действия [1].

Транслятор имеет определённую защиту от ошибок пользователя. Если в ходе лексического анализа какому-либо символу не нашлось шаблона, анализатор выдаёт лексическую ошибку. Если же под какую-то конструкцию не найдено шаблона, выдаётся синтаксическая ошибка.

Разобравшись со строением языка программирования, перейдём к причинам, приводящим к созданию новых языков. Во-первых, новые языки создаются по причине развития технологий/оборудования, для которого будет эффективнее разработать новый язык программирования, заточенный специально под него, нежели использовать старый, пусть даже популярный и удобный. Примером послужит развитие языков программирования, для которых потребовалось создание лексических и синтаксических анализаторов, вследствие чего были разработаны такие языки, как Lex и Yacc. Так же в качестве примера можно привести язык программирования FBD (Functional Block Diagram), использующийся для программирования логических контроллеров, в том числе промышленных. Во-вторых, из-за постоянного пополнения узкоспециализированных задач требуются языки программирования для их решения. Пример: язык MATLAB, используемый для решения задач технических вычислений таких, как линейная алгебра и численные методы.

Так же стоит отметить существование эзотерических языков программирования, назначение которых не определено. Примером может послужить язык LOLCODE, основанный на интернет-мемах о котиках и использующий английский интернет-сленг. Программирование на таких языках является отличнейшей тренировкой.

## 1.1. Цель и задачи работы

Цель работы: написать язык программирования для управления умным домом.

Задачи:

1. Ознакомится с методами создания языка программирования.
2. Выбрать подходящий способ.
3. Написать грамматику языка.
4. Придумать метод взаимодействия умного дома с программой.
5. Написать транслятор.
6. Провести тестирование.

# 2. Техническая постановка

Нужно на основе состояний датчиков и устройств умного дома производить управление им. При этом сохраняя простоту универсальность написания кода. Для реализации языка требуется использование генераторов лексических и синтаксических анализаторов.

## 2.1. Выбор средств реализации

Ознакомившись с теоретической основой для создания собственного языка программирования [2], я приступил к изучению способов для технической реализации. В итоге, для лексического анализа я выбрал генератор лексических анализаторов flex, который разбивает исходный поток данных на токены. Для синтаксического анализа я выбрал генератор синтаксических анализаторов bison, который строит синтаксическое дерево разбора данных токенов.

Преимущества данных программ:

1. Flex и bison являются усовершенствованными версиями lex и yacc соответственно.

2. Данные программные пакеты являются бесплатными на системах Linux

3. Данные пакеты позволяют легко реализовать лексический и синтаксический анализ собственного кода.

4. Пакеты написаны и работают на языке программирования C, который изучался нами в течение семестра.

5. Программы являются одними из первых подобных анализаторов, как следствие, другие программы так или иначе ссылаются на flex и bison.

## 2.2. Минимальные требования к системе команд

* Возможность постоянной работы (в бесконечном цикле).
* Возможность работы с файлом, содержащим системную информацию умного дома (логами системы).
* Язык должен воспринимать бесконечное количество устройств и датчиков.
* Язык должен поддерживать простейшие логические и арифметические выражения.
* Так же должен быть реализовать полный функционал оператора if, а также цикла while.

# 3. Реализация

После ознакомления с программными пакетами я перешел к реализации транслятора. Мной был написан лексический анализатор, а также грамматика синтаксического анализатора.

## 3.1. Список команд

По итогу разработки, в моем языке программирования есть следующие аспекты:

1. Переменные, задавать которые можно их следующими выражениями:

* число
* exp + exp
* exp – exp
* exp \* exp
* exp / exp

Где exp – это число, ранее определенная переменная или такое же выражение.

Работа с переменными реализована через таблицу символов. Таблица символов – это структура, содержащая записи для каждой переменной, а также ее атрибутов, в моем случае – значения переменной.

1. Операторы if и цикл while условия которых можно прописывать, используя следующие логические операторы и операции:

* И (&)
* ИЛИ (|)
* НЕ (!)
* Больше (>)
* Меньше (<)
* Равно (==)
* Неравно (!=)
* Больше или равно (>=)
* Меньше или равно (<=)

1. Функция print(имя\_переменной) которая выводит значение переменной в выходной файл в формате:

“имя\_переменной = значение\_переменной\n”.

1. Функция printf(имя\_переменной) которая выводит значение переменной в выходной файл в формате:

“дд.мм.гггг чч.мм.сс имя\_переменной = значение\_переменной\n”.

## 3.2. Логическая схема

Логическая схема работы транслятора выглядит следующим образом:

Запуск

Лексический анализатор получает входной поток символов

Разбиение кода программы на токены и передача их синтаксическому анализатору

Построение дерева разбора программы силами синтаксического анализатора

Считывание и занесение в таблицу символов состояний всех датчиков и устройств умного дома

Программа дошла до конца файла «логов» системы?

Нет

Да

Считывание строки из файла «логов» системы

Обновление значения состояния датчика/устройства в таблице символов

Обход дерева разбора кода программы и запись изменения состояния устройств в файл «действий» умного дома, если такое необходимо

Завершение

## 3.3. Ограничения языка

Из-за ограниченных сроков мой язык программирования имеет существенные ограничения и недостатки.

1. Отсутствие возможности написания пользовательских функций/алгоритмов.
2. Логические операторы (>, <, ==, !=, >=, <=) с операндами надо прописывать в скобках, если в выражении используются логические операции(и, или, не). Пример: ( (a>8) | (b==0) )
3. После оператора if, if – else, while надо писать ‘;’ .

Пример: if (…) {…} else {...};

4. Всем датчикам/устройствам которые будут использоваться в коде программы нужно заранее присвоить какое-то начальное значение в файле «логов» системы. Иначе их значение будет «-1». По окончанию записи написать строчку «end of start value».

# 4. Тестирование

В данном разделе представлены различные примеры работы программы:

|  |  |  |
| --- | --- | --- |
| Входной файл «логов» системы | Входной файл исходного кода | Выходной файл «действий» системы |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| -“- |  |  |
| -“- |  |  |

Проделанные тестирования показывают возможности разработанного языка, а также его недостатки и недоработки

# 5. Заключение

На момент написания данной курсовой языков программирования несчетное количество, но это не обесценивает значимость разработки своего ЯП. Данный язык имеет огромное количество недоработок, но все они связаны с ограничением по времени разработки. Весомость данной работы заключается понимании процесса работы программного кода изнутри, на уровне анализа текста и построения дерева разбора. При этом стоит отметить, что использовалось именно построение синтаксически абстрактного дерева разбора кода, то при помощи чего работают уже существующие ЯП. Даже учитывая ограничения, получился неплохой интерпретатор.

# Список литературы

1. Ахо А.В., Лам М.С., Сети Р., Ульман Д.Д.. Компиляторы: принципы, технологии и инструментарий: пер. с англ. – 2008. – 2-е изд. – С. 768.
2. John R. Levine flex & bison – 2009 – С. 294

# Приложение

Код лексического анализатора (название файла v5.l):

/\* lexer V5

\*

\* recognize tokens

\*/

%option noyywrap nodefault yylineno

%{

#include "v5.h"

#include "v5.tab.h"

%}

%%

/\* single character ops \*/

"+" |

"-" |

"\*" |

"/" |

"&" |

"|" |

"!" |

"," |

";" |

"=" |

"{" |

"}" |

"(" |

")" { return yytext[0]; }

/\* comparison ops \*/

">" { yylval.fn = 1; return CMP; }

"<" { yylval.fn = 2; return CMP; }

"!=" { yylval.fn = 3; return CMP; }

"==" { yylval.fn = 4; return CMP; }

">=" { yylval.fn = 5; return CMP; }

"<=" { yylval.fn = 6; return CMP; }

/\* keywords \*/

"if" { return IF; }

"else" { return ELSE; }

"while" { return WHILE; }

"print" { return PRINT; }

"printf" { return PRINTF; }

[a-zA-Z]+("\_"?[a-zA-Z0-9]\*)\* { yylval.s = newrecord(yytext); return NAME; }

"-"?[0-9]+"."?[0-9]\* { yylval.d = atof(yytext); return NUMBER; }

/\* ignored text \*/

"//".\*"//" /\* comments \*/

[ \t] /\* white space \*/

"\n"

/\* if there was no token found \*/

. { yyerror("Mystery character %c\n", \*yytext); }

%%

Код синтаксического анализатора (название файла v5.y):

/\* lexer V5

\*

\* recognize tokens

\*/

%option noyywrap nodefault yylineno

%{

#include "v5.h"

#include "v5.tab.h"

%}

%%

/\* single character ops \*/

"+" |

"-" |

"\*" |

"/" |

"&" |

"|" |

"!" |

"," |

";" |

"=" |

"{" |

"}" |

"(" |

")" { return yytext[0]; }

/\* comparison ops \*/

">" { yylval.fn = 1; return CMP; }

"<" { yylval.fn = 2; return CMP; }

"!=" { yylval.fn = 3; return CMP; }

"==" { yylval.fn = 4; return CMP; }

">=" { yylval.fn = 5; return CMP; }

"<=" { yylval.fn = 6; return CMP; }

/\* keywords \*/

"if" { return IF; }

"else" { return ELSE; }

"while" { return WHILE; }

"print" { return PRINT; }

"printf" { return PRINTF; }

[a-zA-Z]+("\_"?[a-zA-Z0-9]\*)\* { yylval.s = newrecord(yytext); return NAME; }

"-"?[0-9]+"."?[0-9]\* { yylval.d = atof(yytext); return NUMBER; }

/\* ignored text \*/

"//".\*"//" /\* comments \*/

[ \t] /\* white space \*/

"\n"

/\* if there was no token found \*/

. { yyerror("Mystery character %c\n", \*yytext); }

%%

Заголовочный файл (v5.h):

/\*

\* declarations for V5

\*/

/\* symbol table \*/

struct symbol{

char \*name;

double value;

};

#define NHASH 1000

struct symbol symtab[NHASH];

struct symbol \*newrecord(char \*name);

double retnum(struct symbol \*name);

/\*

\* node types

\* '+' '-' '\*' '/' '&' '|' '!'

\* 1-6 comparison ops

\* P print function

\* I IF statement

\* W WHILE statement

\* N symbol ref

\* C constant

\* '=' assignment

\* L statement list

\*/

/\*

\* nodes in the Abstract Syntax Tree

\* all have common initial nodetype

\*/

struct origin{

struct ast \*b;

};

struct ast{

int nodetype;

struct ast \*l;

struct ast \*r;

};

struct symprint{ /\* type 'P' \*/

int nodetype;

int t;

struct symbol \*s;

};

struct symasgn{ /\* type '=' \*/

int nodetype;

struct symbol \*n;

struct ast \*v;

};

struct symref{ /\* type 'N' \*/

int nodetype;

struct symbol \*s;

};

struct numval{ /\* type 'C' \*/

int nodetype;

double value;

};

struct flow{ /\* type 'I' \*/

int nodetype;

struct ast \*cond; /\* condition \*/

struct ast \*tl; /\* if try list \*/

struct ast \*el; /\* optional else list \*/

};

/\* build an AST \*/

struct ast \*newast(char nodetype, struct ast \*l, struct ast \*r);

void newbranch(struct ast \*b);

struct ast \*newasgn(struct symbol \*n, struct ast \*v);

struct ast \*newref(struct symbol \*s);

struct ast \*newnum(double value);

struct ast \*newprint(int t, struct symbol \*s);

struct ast \*newflow(int nodetype, struct ast \*cond, struct ast \*tl, struct ast \*el);

struct ast \*newcmp(int cmptype, struct ast \*l, struct ast \*r);

/\* evaluate an AST for orig \*/

void eval\_orig();

/\* evaluate an AST for expr \*/

double eval\_expr(struct ast \*);

/\* delete and free an AST \*/

void treefree(struct ast \*);

/\* interface to the lexer \*/

extern int yylineno;

void yyerror(char \*s, ...);

Код вспомогательного модуля (v5\_fun.c):

/\*

\* helper function for V5

\*/

# include <stdio.h>

# include <stdlib.h>

# include <string.h>

# include <stdarg.h>

# include "v5.h"

# define FILE\_IN "code.art"

# define FILE\_OUT "OUT"

# define FILE\_DATA "IN"

# define NBRANCH 1000

extern FILE \*yyin;

FILE \*data;

/\* time in system \*/

char time[20];

/\* symbol table \*/

int size\_tab = 0;

/\* record as programm \*/

struct symbol \*

newrecord(char \*name){

struct symbol \*sp = &symtab[0];

int scount = 0;

while(scount < NHASH){

if(sp->name && !strcmp(sp->name, name)) { return sp; }

if(!sp->name){

sp->name = strdup(name);

sp->value = -1;

size\_tab++;

return sp;

}

if(++sp >= symtab+NHASH) { sp = symtab; }

}

}

/\* record is data file \*/

void

addrecord(char \*name, double value){

struct symbol \*sp = &symtab[0];

int scount = 0;

while(scount < NHASH){

if(sp->name && !strcmp(sp->name, name)) { sp->value = value; return; }

if(!sp->name){

sp->name = strdup(name);

sp->value = value;

size\_tab++;

return;

}

if(++sp >= symtab+NHASH) { sp = symtab; }

}

}

void

tabprint(){

printf("---------------------------------\n");

for(int i=0; i<size\_tab; i++){

printf("%d|\t%s = %.3f\t\t|\n", i, symtab[i].name, (symtab[i]).value);

}

printf("---------------------------------\n");

}

double

retnum(struct symbol \*name){

if(size\_tab == 0) { yyerror("table is empty | retnum"); return 0.0; }

return name->value;

}

struct ast \*

newast(char nodetype, struct ast \*l, struct ast \*r){

struct ast \*a = malloc(sizeof(struct ast));

if(!a) { yyerror("out of space | newast"); exit(0); }

a->nodetype = nodetype;

a->l = l;

a->r = r;

return a;

}

int count\_branch = 0;

struct origin orig[NBRANCH];

void

newbranch(struct ast \*b){

orig[count\_branch].b = malloc(sizeof(struct ast \*));

orig[count\_branch].b = b;

count\_branch++;

}

struct ast \*

newasgn(struct symbol \*n, struct ast \*v){

struct symasgn \*a = malloc(sizeof(struct symasgn));

if(!a) { yyerror("out of space | newasgn"); exit(0); }

a->nodetype = '=';

a->n = n;

a->v = v;

return (struct ast \*)a;

}

struct ast \*

newref(struct symbol \*s){

struct symref \*a = malloc(sizeof(struct symref));

if(!a) { yyerror("out of space | newref"); exit(0); }

a->nodetype = 'N';

a->s = s;

return (struct ast \*)a;

}

struct ast \*

newnum(double value){

struct numval \*a = malloc(sizeof(struct numval));

if(!a) { yyerror("out of space | newnum"); exit(0); }

a->nodetype = 'C';

a->value = value;

return (struct ast \*)a;

}

struct ast \*

newprint(int t, struct symbol \*s){

struct symprint \*a = malloc(sizeof(struct symprint));

if(!a) { yyerror("out of space | newprint"); exit(0); }

a->nodetype = 'P';

a->t = t;

a->s = s;

return (struct ast \*)a;

}

struct ast \*

newflow(int nodetype, struct ast \*cond, struct ast \*tl, struct ast \*el){

struct flow \*a = malloc(sizeof(struct flow));

if(!a) { yyerror("out of space | newflow"); exit(0); }

a->nodetype = nodetype;

a->cond = cond;

a->tl = tl;

a->el = el;

return (struct ast \*)a ;

}

struct ast \*

newcmp(int cmptype, struct ast \*l, struct ast \*r){

struct ast \*a = malloc(sizeof(struct ast));

if(!a) { yyerror("out of space | newcmp"); exit(0); }

a->nodetype = '0' + cmptype;

a->l = l;

a->r = r;

return a;

}

void

eval(struct ast \*a){

if(!a) { yyerror("internal error, null eval | eval"); }

switch(a->nodetype){

case 'P':

if( ((struct symprint \*)a)->t == 0 ) { printf("%s\t", time); }

printf("%s = %.3f\n", ((struct symprint \*)a)->s->name, ((struct symprint \*)a)->s->value); break;

case '=': ((struct symasgn \*)a)->n->value = eval\_expr(((struct symasgn \*)a)->v); /\*tabprint();\*/ break;

case 'L': eval(a->l); if(a->r != NULL) { eval(a->r); } break;

case 'I':

if( eval\_expr( ((struct flow \*)a)->cond ) == 1 ) { eval( ((struct flow \*)a)->tl ); }

else { if( ((struct flow \*)a)->el ) {eval( ((struct flow \*)a)->el );} }

break;

case 'W':

while( eval\_expr( ((struct flow \*)a)->cond ) != 0 ) { eval( ((struct flow \*)a)->tl ); } break;

}

}

void

eval\_orig(){

for(int i=0; i<count\_branch; i++){

eval(orig[i].b);

}

}

double

eval\_expr(struct ast \*a){

double v ;

if(!a) { yyerror("internal error, null eval | eval\_expr"); return 0.0; }

switch(a->nodetype){

/\* name referense \*/

case 'N': v = retnum(((struct symref \*)a)->s); break;

/\* constant \*/

case 'C': v = ((struct numval \*)a)->value; break;

/\* expressions \*/

case '+': v = eval\_expr(a->l) + eval\_expr(a->r); break;

case '-': v = eval\_expr(a->l) - eval\_expr(a->r); break;

case '\*': v = eval\_expr(a->l) \* eval\_expr(a->r); break;

case '/': v = eval\_expr(a->l) / eval\_expr(a->r); break;

/\* comparisons \*/

case '&': v = ( (eval\_expr(a->l) == 1.0) && (eval\_expr(a->r) == 1.0) )? 1 : 0; break;

case '|': v = ( (eval\_expr(a->l) == 1.0) || (eval\_expr(a->r) == 1.0) )? 1 : 0; break;

case '!': v = ( eval\_expr(a->l) == 0 )? 1 : 0; break;

case '1': v = (eval\_expr(a->l) > eval\_expr(a->r))? 1 : 0; break;

case '2': v = (eval\_expr(a->l) < eval\_expr(a->r))? 1 : 0; break;

case '3': v = (eval\_expr(a->l) != eval\_expr(a->r))? 1 : 0; break;

case '4': v = (eval\_expr(a->l) == eval\_expr(a->r))? 1 : 0; break;

case '5': v = (eval\_expr(a->l) >= eval\_expr(a->r))? 1 : 0; break;

case '6': v = (eval\_expr(a->l) <= eval\_expr(a->r))? 1 : 0; break;

default: printf("internal error: bad node - '%c'\n", a->nodetype);

}

return v;

}

void

get\_data(){

char \*name;

double value;

char str[64];

char \*str\_split = malloc(sizeof(str));

char \*end = "end of start value\n";

char \*spl\_2 = " ";

char \*estr;

estr = fgets(str, sizeof(str), data);

while( strcmp(str, end) ){

str\_split = strtok(str, spl\_2);

name = str\_split;

str\_split = strtok(NULL, spl\_2);

value = atof(str\_split);

addrecord(name, value);

estr = fgets(str, sizeof(str), data);

}

}

void

add\_data(){

char \*name;

double value;

char str[64];

char \*str\_split = malloc(sizeof(str));

char \*spl\_1 = "|";

char \*spl\_2 = " ";

char \*estr;

estr = fgets(str, sizeof(str), data);

str\_split = strtok(str, spl\_1);

strcpy(time, str\_split);

str\_split = strtok(NULL, spl\_1);

str\_split = strtok(str\_split, spl\_2);

name = str\_split;

str\_split = strtok(NULL, spl\_2);

value = atof(str\_split);

addrecord(name, value);

}

int

main(){

data = fopen(FILE\_DATA, "r");

yyin = fopen(FILE\_IN, "r");

FILE \*file\_out = freopen(FILE\_OUT, "w+", stdout);

yyparse();

get\_data();

do{

add\_data();

eval\_orig();

}while( feof(data) == 0 );

return 0;

}

void

yyerror(char \*s, ...){

va\_list ap;

va\_start(ap, s);

fprintf(stderr, "in line %d error: ", yylineno);

vfprintf(stderr, s, ap);

fprintf(stderr, "\n");

}