# Выбор целевой платформы

Целевая платформа: Unix-подобная ОС с компилятором GCC. Язык программирования будет транслироваться в язык C.

# Разработка языка программирования

Концепт:

Язык программирования Simp (облегчённый английский) предназначен для начинающих программистов и тех, кто хочет быстро создавать простые программы. Основная идея Simp заключается в использовании интуитивно понятного синтаксиса, основанного на английском языке, что делает его доступным для пользователей без технического бэкграунда.

**Описание команд:**

1. **declare** – команда для объявления переменной. Используется как: declare a или declare a 5.
2. **equalize** – команда для приравнивания переменной. Используется как: equalize a 5 или equalize a b.
3. **append** – команда для прибавления значения к переменной. Используется как: append a 5 или append a b.
4. **multiply** – команда для умножения переменной. Используется как: multiply a 5 или multiply a b.
5. **divide** – команда для деления переменной. Используется как: divide a 5 или divide a b.
6. **subtract** – команда для вычитания значения из переменной. Используется как: subtract a 5 или subtract a b.
7. **loop** – команда для выполнения цикла. Используется как: if a > 7 или if b < a
8. **end** – команда для обозначения конца цикла или оператора «if».
9. **print** – выводит переменную на экран. Используется как: print a
10. **if** – выводит переменную на экран. Используется как: if a > 5 или if a < b
11. **counter** – команда - счётчик. Используется как: counter 3, выполняется 3 раза.
12. **function** – команда для объявления функции.
13. **start**– команда для вызова функции.

# Ручная трансляция под целевую платформу

Для примера ручной трансляции языка была написана следующая программа:

function it

declare f 0

counter 1000

append f 1

end

end

function count

declare f 0

counter 100000

append f 1

start it

end

end

function main

declare b 5

declare a 3

counter 3

multiply b a

multiply a 2

end

if b > 50

print b

end

if b < 51

print a

end

start count

return 0

end

Ниже представлен код, транслированный на язык С:

#include <stdio.h>

int it(){

int f = 0;

for (int i = 0; i < 1000; i++)

{

f += 1;

}

}

int count(){

int f = 0;

for (int i = 0; i < 100000; i++)

{

f += 1;

it();

}

}

int main(){

int b = 5;

int a = 3;

for (int i = 0; i < 3; i++)

{

b \*= a;

a \*= 2;

}

if (b > 50)

printf("%d", b);

if (b < 51)

printf("%d", a);

count();

return 0;

}

# Разработка лексического анализатора

Для разработанного языка был написан лексический анализатор:

%{

#include <stdlib.h>

#include <string.h>

#include "file\_b.h"

%}

digit [0-9]

letter [a-z]

regular [<>()]

%%

function { return(FUNCTION); }

start { return (START); }

declare { return(DECLARE); }

equalize { return(EQUAL); }

append { return(APPEND); }

subtract { return(SUBTRACT); }

multiply { return(MULTIPLY); }

divide { return(DIVIDE); }

loop { return(LOOP); }

end { return(END); }

print { return(PRINT); }

return { return(RETURN); }

if { return(IF); }

counter { return(COUNTER); }

{letter}+ { strcpy( yylval.var, yytext ) ; return( VAR ); }

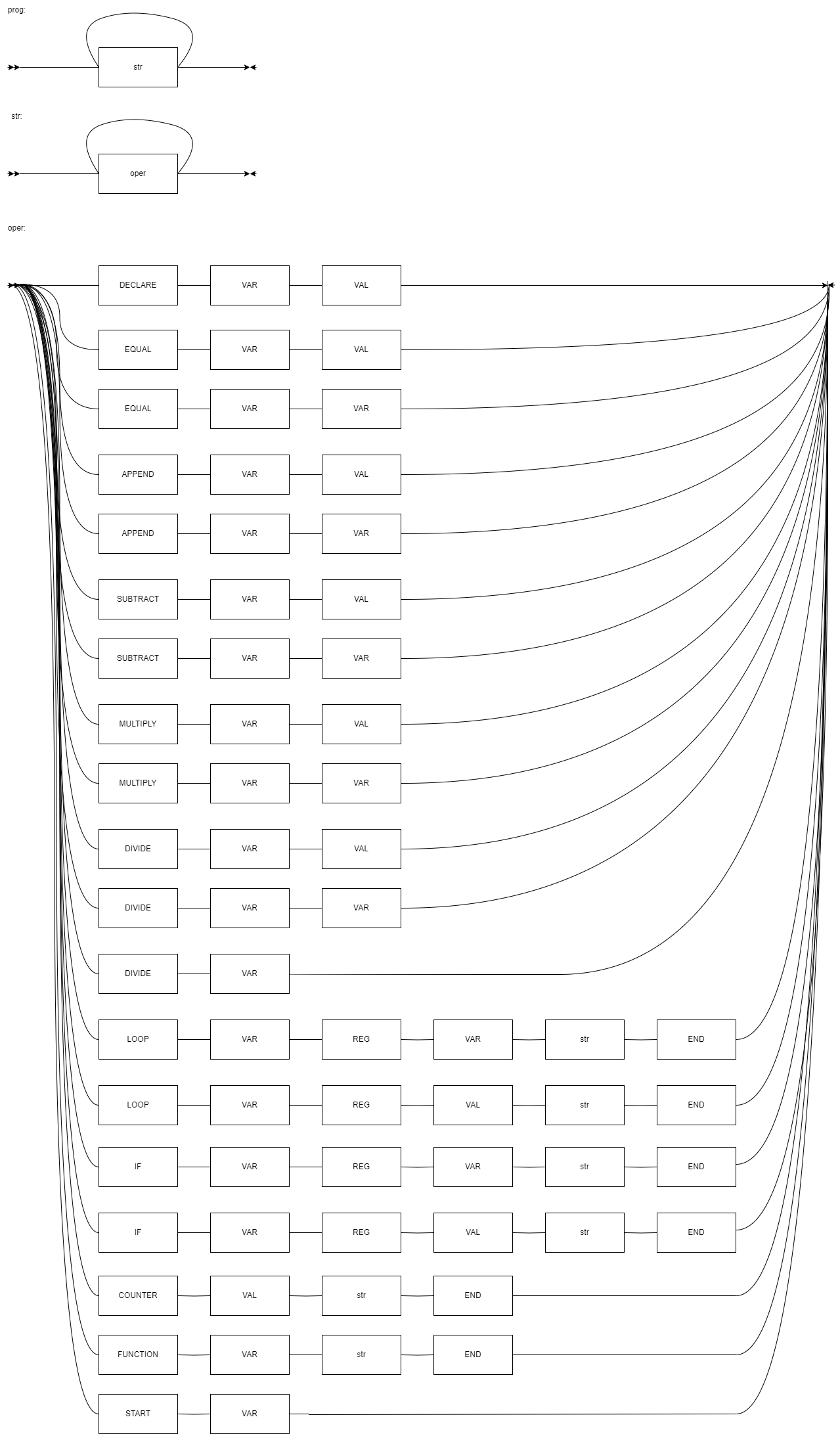
{digit}+ { sscanf( yytext, "%d", &yylval.val ) ; return( VAL ); }

{regular}+ { strcpy( yylval.var, yytext ) ; return( REG ); }

[ \n] {}

. { write(2,yytext,1); return( yytext[0] ); }

# Грамматика синтаксического разбора



# Разработка синтаксического анализатора

%{

%{

#include <stdio.h>

char buf[2048];

%}

%union {

int val;

char var[2048];

char reg[2048];

}

%token <val> VAL

%token <var> VAR

%token <reg> REG

%token DECLARE EQUAL APPEND SUBTRACT LOOP END MULTIPLY DIVIDE PRINT RETURN IF COUNTER FUNCTION START

%type <var> str

%type <var> oper

%%

prog: str {printf ("#include <stdio.h>\n\n%s", $1);}

str: oper {

sprintf($$, "%s", $1);

}

| oper str {

sprintf($$, "%s\n%s", $1, $2);

};

oper: DECLARE VAR {

sprintf($$, "int %s;", $<var>2);

}

| DECLARE VAR VAL {

sprintf($$, "int %s=%d;", $<var>2, $<val>3);

}

| EQUAL VAR VAL {

sprintf($$, "%s=%d;", $<var>2, $<val>3);

}

| EQUAL VAR VAR {

sprintf($$, "%s=%s;", $<var>2, $<var>3);

}

| APPEND VAR VAL {

sprintf($$, "%s+=%d;", $<var>2, $<val>3);

}

| APPEND VAR VAR {

sprintf($$, "%s+=%s;", $<var>2, $<var>3);

}

| SUBTRACT VAR VAL {

sprintf($$, "%s-=%d;", $<var>2, $<val>3);

}

| SUBTRACT VAR VAR {

sprintf($$, "%s-=%s;", $<var>2, $<var>3);

}

| MULTIPLY VAR VAL {

sprintf($$, "%s\*=%d;", $<var>2, $<val>3);

}

| MULTIPLY VAR VAR {

sprintf($$, "%s\*=%s;", $<var>2, $<var>3);

}

| DIVIDE VAR VAL {

sprintf($$, "%s/=%d;", $<var>2, $<val>3);

}

| DIVIDE VAR VAR {

sprintf($$, "%s/=%s;", $<var>2, $<var>3);

}

| PRINT VAR {

sprintf($$, "printf(\"%s\\n\", %s);", "%d", $<var>2);

}

| RETURN VAL {

sprintf($$, "return %d;", $<val>2);

}

| LOOP VAR REG VAR str END {

sprintf($$, "while(%s %s %s) {\n%s\n}", $<var>2, $<reg>3, $<var>4, $<var>5);

}

| LOOP VAR REG VAL str END {

sprintf($$, "while(%s %s %d) {\n%s\n}", $<var>2, $<reg>3, $<val>4, $<var>5);

}

| IF VAR REG VAR str END {

sprintf($$, "if(%s %s %s) {\n%s\n}", $<var>2, $<reg>3, $<var>4, $<var>5);

}

| IF VAR REG VAL str END {

sprintf($$, "if(%s %s %d) {\n%s\n}", $<var>2, $<reg>3, $<val>4, $<var>5);

}

| COUNTER VAL str END {

sprintf($$, "for(int \_counter\_i\_=0; \_counter\_i\_<%d; \_counter\_i\_++) {\n%s\n}", $<val>2, $<var>3);

}

| FUNCTION VAR str END {

sprintf($$, "int %s() {\n%s\n}", $<var>2, $<var>3);

}

| START VAR {

sprintf($$, "%s();", $<var>2);

}

;

%%

yyerror (s)

char \*s;

{

printf ("err:%s\n", s);

}

main()

{

yyparse();

}

1. }MakeFile

TARGET := simen

COMPILER := simpCo

SOURCES := ./

LIBS = -lfl

MFILES := $(foreach dir,$(SOURCES), $(wildcard $(dir)\*.simp))

LFILES := $(foreach dir,$(SOURCES), $(wildcard $(dir)\*.l))

YFILES := $(foreach dir,$(SOURCES), $(wildcard $(dir)\*.y))

CC = $-gcc

CFLAGS = -o

FLEX = $-flex

FFLAGS = -o

BISON = $-bison

BFLAGS = -d -o

all: $(YFILES) $(LFILES) $(MFILES)

$(BISON) $(BFLAGS) file\_b.c $(YFILES)

$(FLEX) $(FFLAGS) file\_f.c $(LFILES)

$(CC) $(CFLAGS) $(COMPILER) file\_b.c file\_f.c $(LIBS)

./$(COMPILER) <$(MFILES)> $(TARGET).c

$(CC) -O0 -pg $(CFLAGS) ./simp ./$(TARGET).c

./simp

gprof ./simp gmon.out > analysis.txt

gedit analysis.txt &

clean:

@rm -rf \*.c \*.h simp $(COMPILER) \*.out analysis.txt

