|  |
| --- |
|  |
| **システムプログラミング実験　　コンパイラの作成** |
| **実験レポート** |
|  |
| 学修番号12345678 日野太郎 |
| 提出日2023年12月20日 |

１．実験の目的

この節では，実験の目的と課題について指導書を参考に自分の言葉で説明してください．丸写しはダメです．

２．開発環境

使用した計算機とCコンパイラの情報（オンラインのCコンパイラを利用した場合はそのサイトの情報）を書いてください．４－４０５実験室のPCを利用した場合はそのスペックを書く．自分のPCを利用した場合はそのスペックとCコンパイラの情報を書くこと．

３．作成したプログラムの説明

この節では，作成したプログラムについて説明してください．ファイル（モジュール）毎ではなく，関数毎に説明を付けること．ただし，とくに変更していない関数については簡単な説明で構いません．新たに追加した関数については，その機能や目的について詳しく説明してください．作成したプログラムを貼り付けて説明するのがよいでしょう．ソースコードを1行毎に説明する必要はありませんが，コードの背景にある考え方（役割や意味など）が分かるように説明をつけること．プログラムの説明が（ほとんど）ない場合は評価が下がります．

４．プログラムの実行結果

代入文とwhile文の翻訳結果を貼り付けてください．begin-end文の翻訳ができた人は，図５の９の階乗を計算するプログラムの翻訳結果を貼り付けてください．さらに追加課題ができた人は，図７のプログラム例の翻訳結果を貼り付けてください．



自分の名前や学修番号を入れること！

図１．代入文の翻訳結果

５．まとめ

本実験に関する考察や感想を書いてください．

付録：プログラムリスト

ここに全プログラムのソースコードを貼り付けてください．スクリーンショットではなく，テキストデータとして貼り付けること．

/\*\*\*\*\* global.h \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <ctype.h>

#define BSIZE 128

#define NONE -1

#define EOS '\0'

#define NUM 256

#define DIV 257

#define MOD 258

#define ID 259

#define DONE 260

int tokenval;

int lineno;

struct entry {

char \*lexptr;

int token;

};

struct entry symtable[];

/\*\*\*\*\* lexer.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

char lexbuf[BSIZE];

int lineno = 1;

int tokenval = NONE;

int lexan()

{

int t;

while(1) {

t = getchar();

if (t == ' ' || t == '\t')

;

else if ( t == '\n')

lineno = lineno + 1;

else if (isdigit(t)) {

ungetc(t, stdin);

scanf("%d", &tokenval);

return NUM;

}

else if (isalpha(t)) {

int p,b = 0;

while (isalnum(t)) {

lexbuf[b] = t;

t = getchar();

b = b + 1;

if (b >= BSIZE)

error("compiler error");

}

lexbuf[b] = EOS;

if (t !=EOF)

ungetc(t, stdin);

p = lookup(lexbuf);

if (p == 0)

p = insert(lexbuf, ID);

tokenval = p;

return symtable[p].token;

}

else if (t == EOF)

return DONE;

else {

tokenval = NONE;

return t;

}

}

}

/\*\*\*\*\* parser.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int lookahead;

parse()

{

lookahead = lexan();

while (lookahead != DONE) {

expr(); match(';');

}

}

expr()

{

int t;

term();

while(1)

switch (lookahead) {

case '+': case '-':

t = lookahead;

match(lookahead); term(); emit(t, NONE);

continue;

default:

return;

}

}

term()

{

int t;

factor();

while(1)

switch (lookahead) {

case '\*': case '/': case DIV: case MOD:

t = lookahead;

match(lookahead); factor(); emit(t, NONE);

continue;

default:

return;

}

}

factor()

{

switch(lookahead) {

case '(':

match('('); expr(); match(')'); break;

case NUM:

emit(NUM, tokenval); match(NUM); break;

case ID:

emit(ID, tokenval); match(ID); break;

default:

error("syntax error");

}

}

match(t)

int t;

{

if (lookahead == t)

lookahead = lexan();

else error("syntax error");

}

/\*\*\*\*\* emitter.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

emit(t, tval)

int t, tval;

{

switch(t) {

case '+': case '-': case '\*': case '/':

printf("%c\n", t); break;

case DIV:

printf("DIV\n"); break;

case MOD:

printf("MOD\n"); break;

case NUM:

printf("%d\n", tval); break;

case ID:

printf("%s\n", symtable[tval].lexptr); break;

default:

printf("token %d, tokenval %d\n", t, tval);

}

}

/\*\*\*\*\* symbol.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define STRMAX 999

#define SYMMAX 100

char lexemes[STRMAX];

int lastchar = -1;

struct entry symtable[SYMMAX];

int lastentry = 0;

int lookup(s)

char s[];

{

int p;

for (p = lastentry; p > 0; p = p -1)

if (strcmp(symtable[p].lexptr, s) == 0)

return p;

return 0;

}

int insert(s, tok)

char s[];

int tok;

{

int len;

len = strlen(s);

if (lastentry +1 >= SYMMAX)

error("symbol table full");

if (lastchar + len +1 >= STRMAX)

error("lexemes array full");

lastentry = lastentry +1;

symtable[lastentry].token = tok;

symtable[lastentry].lexptr = &lexemes[lastchar + 1];

lastchar = lastchar + len + 1;

strcpy(symtable[lastentry].lexptr, s);

return lastentry;

}

/\*\*\*\*\* init.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct entry keywords[] = {

"div", DIV,

"mod", MOD,

0, 0

};

init()

{

struct entry \*p;

for (p = keywords; p -> token; p++)

insert(p -> lexptr, p -> token);

}

/\*\*\*\*\* error.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

error(m)

char \*m;

{

fprintf(stderr, "line %d: %s\n", lineno, m);

exit(1);

}

/\*\*\*\*\* main.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

main()

{

init();

parse();

exit(0);

}