ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УПРАВЛЕНИЕ ДИСТАНЦИОННОГО ОБУЧЕНИЯ И ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

Сборка и трассировка выполнения простейшего однопроходного компилятора

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по специальностям
230105-«Программное обеспечение вычислительной техники и автоматизированных систем»
010503-«Математическое обеспечение и администрирование информационных систем»

Автор

Коледов Леонид Викторович

Ростов-на-Дону, 2013

Аннотация

Данная разработка может быть использована в качестве основного учебного материала по дисциплинам: «Теория языков программирования и методы трансляции» и «Теория вычислительных процессов»

Автор

Коледов Леонид Викторович, к. ф.-м. н., доцент, профессор кафедры

Область научных интересов

информационные технологии, системы искусственного интеллекта

Оглавление

ЛАБОРАТОРНАЯ РАБОТА 1.....4

Приложение 1..... 5
Приложение 2..... 12
Приложение 3..... 15

ТЕОРИЯ

СХЕМЫ ТРАНСЛЯЦИИ

ЯЗЫК ПРОГРАММИРОВАНИЯ «С»

ПРОСТОЙ ОДНОПРОХОДНЫЙ КОМПИЛЯТОР

ЛАБОРАТОРНАЯ РАБОТА 1

Задание 1.

Используя материалы Приложений 1,2 и описания, помещенные в SimpComp.doc, провести сборку в форме, поддерживающей трассировку исполнения проекта, описанного там. (В простейшем варианте можно использовать внесение в текст отладочных операторов вывода “printf”.)

Задание 2.

Проведите пошаговое исполнение программы с входными файлами «expl?.in» и включите в отчет выходные файлы. В комментарии (помимо отладочных печатей) включите номер входной строки, имя исполняемого модуля, имя функции и выходной символ. Попытайтесь «достать» компилятор, предложив ему задание, с которым он не справится.

Задание 3.

Модифицируйте модуль обработки ошибок из раздела 2.9, чтобы при ошибке он пропускал неверное выражение и переходил к следующему.

Приложение 1.

Ниже приводится список файлов, составляющих проект нашего компилятора:

global.h, emitter.c, init.c, lexer.c, main.c, parser.c, symbol.c.

В предпринятом далее модифицированном проекте добавлены функции, создающие и пополняющие log-файл, который придется в помощь для составления отчета.

********** emitter.c

```
#include"global.h"

emit(int t, int tval) //     Генерация вывода

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);
```

********** init.c

```
#include "global.h"

struct entry keywords[] =
"div", DIV,
"mod", MOD,
0, 0
;
init()
```

```
struct entry *p;
for(p = keywords; p->token; p++)
        insert(p->lexptr, p->token);

************************************* lexer.c
//--------------lexer.c----------------------
#include"global.h"

char lexbuf[BSIZE];
int lineno = 1;
int tokenval = NONE;

int lexan()     //Лексический анализатор

int t;
while(1)

        t = getchar();
        if(t == ' ' || t == '\t')
        ;       //Отбрасываем разделители-пробелы
        else if (t == 'n')
        lineno++;
        else if (isdigit(t))        //t - цифра

        ungetc(t, stdin);
        scanf("%d", &tokenval);
        return NUM;

        else if (isalpha(t)) // t - буква

        int p, b = 0;
        while(isalnum(t)) //t - буква или цифра

                lexbuf[b++] = t;
                t = getchar();
                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;
```

********** main.c

```
#include "global.h"

int main()

init();
parse();
exit(0);
```

**********984 parser.c

```
//----------------parser.c---------------
#include "global.h"

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(int t)

if (lookahead == t)
        lookahead = lexan();
else error("syntax error");

****************************************** symbol.c
#include "global.h"
#define STRMAX          999
#define SYMMAX          100

char lexemes[STRMAX];
int lastchar = -1;

struct entry symtable[SYMMAX];
int lastentry = 0;

int lookup(char s[])

int p;
for(p = lastentry; p>0; p--)      if(strcmp(symtable[p].lexptr, s) ==0)
        return p;
return 0;

int insert(char s[], int tok)

int len;
len = strlen(s);
if(lastentry + 1 >= SYMMAX)
        error("symbol table full");
if(lastentry + len + 1 >= STRMAX)
        error("lexemes array full");
lastentry++;
symtable[lastentry].token = tok;
```

```
symtable[lastentry].lexptr = &lexemes[lastchar + 1];
lastchar += len+1;
strcpy(symtable[lastentry].lexptr, s);
return lastentry;

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

#include<stdio.h>
#include<ctype.h>

#define BSIZE  128 // Размер буфера
#define NONE   -1
#define EOS    '\0'

#define NUM    256
#define DIV    257
#define MOD    258
#define ID     259
#define DONE   260

extern int         tokenval;
extern int         lineno;

struct entry

char *lexptr;
int token;
;
extern struct entry symtable[];            //Таблица символов
//------------------Zmeyoff----------------------
/*
int main()

return 0;

*/

********************************** compile.sh
#!/bin/sh
clear
```

```
gcc -c *.c
gcc *.o
```

Приложение 2.

Make-файл для компиляции под BC5 mycomp.mak, находящийся в каталоге BC5.

```
#---------------------------------------------------------------------------
#
# Borland C++ IDE generated makefile
# Generated 24.02.2005 at 13:34:37
#
.AUTODEPEND


#
# Borland C++ tools
#
IMPLIB  = Implib
BCC32   = Bcc32 +BccW32.cfg
BCC32I  = Bcc32i +BccW32.cfg
TLINK32 = TLink32
TLIB    = TLib
BRC32   = Brc32
TASM32  = Tasm32
#
# IDE macros
#


#
# Options
#
IDE_LinkFLAGS32 =  -LC:\BC5\LIB
LinkerLocalOptsAtC32_mycompdexe =  -n -Tpe -ap -c
ResLocalOptsAtC32_mycompdexe =
BLocalOptsAtC32_mycompdexe =
CompInheritOptsAt_mycompdexe = -IC:\BC5\INCLUDE
LinkerInheritOptsAt_mycompdexe = -x
LinkerOptsAt_mycompdexe = $(LinkerLocalOptsAtC32_mycompdexe)
ResOptsAt_mycompdexe = $(ResLocalOptsAtC32_mycompdexe)
BOptsAt_mycompdexe = $(BLocalOptsAtC32_mycompdexe)

#
# Dependency List
```

```
#
Dep_mycomp = \
  mycomp.exe

mycomp : BccW32.cfg $(Dep_mycomp)
  echo MakeNode

Dep_mycompdexe = \
  parser.obj\
  symbol.obj\
  init.obj\
  lexer.obj\
  emitter.obj\
  main.obj

mycomp.exe : $(Dep_mycompdexe)
  $(TLINK32) @&&|
 /v  $(IDE_LinkFLAGS32)  $(LinkerOptsAt_mycompdexe)  $(Linker-
InheritOptsAt_mycompdexe) +
C:\BC5\LIB\c0x32.obj+
parser.obj+
symbol.obj+
init.obj+
lexer.obj+
emitter.obj+
main.obj
$<,$*
C:\BC5\LIB\import32.lib+
C:\BC5\LIB\cw32mt.lib



|
parser.obj :  parser.c
  $(BCC32) -P- -c @&&|
 $(CompOptsAt_mycompdexe)  $(CompInheritOptsAt_mycompdexe)  -
o$@ parser.c
|

symbol.obj :  symbol.c
  $(BCC32) -P- -c @&&|
 $(CompOptsAt_mycompdexe)  $(CompInheritOptsAt_mycompdexe)  -
```

```
o$@ symbol.c
        |

        init.obj :  init.c
          $(BCC32) -P- -c @&&|
         $(CompOptsAt_mycompdexe)  $(CompInheritOptsAt_mycompdexe)  -
o$@ init.c
        |

        lexer.obj :  lexer.c
          $(BCC32) -P- -c @&&|
         $(CompOptsAt_mycompdexe)  $(CompInheritOptsAt_mycompdexe)  -
o$@ lexer.c
        |

        emitter.obj :  emitter.c
          $(BCC32) -P- -c @&&|
         $(CompOptsAt_mycompdexe)  $(CompInheritOptsAt_mycompdexe)  -
o$@ emitter.c
        |

        main.obj :  main.c
          $(BCC32) -P- -c @&&|
         $(CompOptsAt_mycompdexe)  $(CompInheritOptsAt_mycompdexe)  -
o$@ main.c
        |

        # Compiler configuration file
        BccW32.cfg :
           Copy &&|
        -w
        -R
        -v
        -vi
        -H
        -H=mycomp.csm
        -WM
        -WC
        | $@
```

Приложение 3.

В приведенном ниже образце загрузочного файла mycomp.bat упоминаются входной и выходной файлы. Помимо этого формируется log-файл c:\temp\mylog.txt.

```
simpcomp2 < expl2.in > expl2.out
```

Файл expl1.in.

```
42+31*5/3;
```

Файл expl1.out.

```
42
31
5
*
3
/
+
```

```
c:\temp\mylog.txt
```

```
02:59:43:Line Main 1. Начали выполнение
02:59:43:Line init. init
02:59:43:Line parcer.c. parce()
02:59:43:Line lexer.c. lexan started
02:59:43:Line lexer.c lexema. 4
02:59:43:Line lexan . 256
02:59:43:Line parcer.c. factor()
02:59:43:Line match. 256
02:59:43:Line lexer.c. lexan started
02:59:43:Line lexer.c lexema. +
02:59:43:Line match. 43
02:59:43:Line lexer.c. lexan started
02:59:43:Line lexer.c lexema. 3
02:59:43:Line parcer.c. factor()
02:59:43:Line match. 256
02:59:43:Line lexer.c. lexan started
```

02:59:43:Line lexer.c lexema. *
02:59:43:Line match. 42
02:59:43:Line lexer.c. lexan started
02:59:43:Line lexer.c lexema. 5
02:59:43:Line parcer.c. factor()
02:59:43:Line match. 256
02:59:43:Line lexer.c. lexan started
02:59:43:Line lexer.c lexema. /
02:59:43:Line match. 47
02:59:43:Line lexer.c. lexan started
02:59:43:Line lexer.c lexema. 3
02:59:43:Line parcer.c. factor()
02:59:43:Line match. 256
02:59:43:Line lexer.c. lexan started
02:59:43:Line lexer.c lexema. ;
02:59:43:Line match. 59
02:59:43:Line lexer.c. lexan started
02:59:43:Line lexer.c lexema. я
02:59:43:Line Main 5. Выполнение закончили