|  |
| --- |
| Київський національний університет  «Київський політехнічний інститут» |
| Факультет інформатики та обчислювальної техніки |
| Кафедра обчислювальної техніки |
| Лабораторна робота №5 |
| з дисципліни «Системне програмування» |
| *Залікова книжка № 0103* |
|  |
| Виконав студент 3-го курсу |
| групи ІО-01 |
| Редько Олександр |

|  |
| --- |
| 2012 |

Мета: Одержання навичок настроювання таблиць для висхідних синтаксичних аналізаторів та програм побудови графів і дерев розбору на етапі синтаксичного аналізу з запам’ятовуванням покажчиків на вхідні образи та внутрішні коди. Побудова таблиць імен та констант за допомогою індексів над термінальними вузлами описів та виконавчих кодів

Варіант 3 : switch(c){case 0: b=2\*a[n]; break; default: b=d;};

+ [0] {ndOp=\_switch prvNd=0x004b5af8 pstNd=0x004b5b70 ...} lxNode

+ [1] {ndOp=\_nam prvNd=0x004b4c88 pstNd=0x00000000 ...} lxNode

+ [2] {ndOp=\_case prvNd=0x004b5af8 pstNd=0x004b5b70 ...} lxNode

+ [3] {ndOp=\_nam prvNd=0x004b4c8a pstNd=0x00000000 ...} lxNode

+ [4] {ndOp=\_cln prvNd=0x004b5b48 pstNd=0x004b5b98 ...} lxNode

+ [5] {ndOp=\_ass prvNd=0x00000000 pstNd=0x004b5bc0 ...} lxNode

+ [6] {ndOp=\_nam prvNd=0x004b4c8c pstNd=0x00000000 ...} lxNode

+ [7] {ndOp=\_mul prvNd=0x004b5b48 pstNd=0x004b5b98 ...} lxNode

+ [8] {ndOp=\_nam prvNd=0x004b4c8a pstNd=0x00000000 ...} lxNode

+ [9] {ndOp=\_break prvNd=0x00000000 pstNd=0x004b5bc0 ...} lxNode

+ [10] {ndOp=\_default prvNd=0x004b5af8 pstNd=0x004b5b70 ...} lxNode

+ [11] {ndOp=\_cln prvNd=0x004b5b48 pstNd=0x004b5b98 ...} lxNode

+ [12] {ndOp=\_nam prvNd=0x004b4c8a pstNd=0x00000000 ...} lxNode

+ [13] {ndOp=\_ass prvNd=0x00000000 pstNd=0x004b5bc0 ...} lxNode

+ [14] {ndOp=\_endcase prvNd=0x004b5af8 pstNd=0x004b5b70 ...} lxNode

+ [15] {ndOp=\_EOS prvNd=0x004b5b20 pstNd=0x00000000 ...} lxNode

+ [16] {ndOp=\_EOF prvNd=0x00000000 pstNd=0x00000000 ...} lxNode

# Лістинг програми

// spLb5.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include "..\spLb3\token.h"

#include "..\spLb3\visgrp.h"

#include "..\spLb4\tables.h"

#include "..\spLb4\lexan.h"

#include "syntaxP.h"

#include "..\spLb4\langio.h"

#include <stdio.h>

#include <string.h>

extern struct recrdKWD tablKWD[];

extern struct lxNode nodes[MAX\_NODES]; // масив приймач вузлів дерева

extern enum ltrType ltClsC[256];

extern enum ltrType ltClsP[256];

extern enum tokType dlCdsC[256];

extern enum tokType dlCdsP[256];

extern enum ltrType ltClsC[256];

extern enum ltrType ltClsP[256];

enum ltrType \*ltCls=ltClsC;

enum tokType \*dlCds=dlCdsC;

char file\_name[20];

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

{int nn=-1, nr=0, nc=1; //np,

if (argc>1)

{strcpy(file\_name,argv[1]);

printf("Processing file -- %s\n",file\_name);}

else

{printf("Please enter file Name: ");

scanf("%s",file\_name);

strcat(file\_name,".h");

}

opFls(file\_name);

LxAnInit('C');

// srtBin(tablKWD, 67);

do{//np=nn;

nn=LxAnlzr();

}while(nodes[nn].ndOp!=\_EOF);

//prLaTxt(nodes,nn);

printf("\n");

SxAnInit('C');

nr=0; nc=1; nodes[0].prnNd=-1;

do nr=nxtProd(nodes,nr,nc);

while(++nc<nn);

// конверсія до семантичної обробки

prLxTxt(nodes+nr);

// nr=prCmpr(nodes,nn,nr);// компресія для скорочення графа

// prLxTxt(nodes+nr);

printf("\n");

return 0;

}

## Index.cpp

#include "stdafx.h"

#include "..\spLb3\token.h"

#include "tables.h"

#include "index.h"

#include <stdlib.h>

// порівняння рядків

// порівняння терміналів за відношенням порядку

int cmpTrm(struct lxNode\*k0,struct lxNode\*kArg)//cmpKys

{int i=cmpStr((unsigned char\*)k0->prvNd,

(unsigned char\*)kArg->prvNd);

if(i)return i;

return k0->stkLength - kArg->stkLength; // порівняння номерів модулів

}

unsigned nNdxNds=0;

struct indStrUS ndxNds[50]={{NULL,NULL,NULL,0}},

\*pRtNdx=ndxNds, nilNds={NULL,NULL,NULL,0};

// вибірка через пошук за двійковим деревом

struct indStrUS\*selBTr(struct lxNode\*kArg,struct indStrUS\*rtTb)

{int df;

while(df=cmpTrm(kArg,rtTb->pKyStr))

if(df>0){if(rtTb->pRtPtr)rtTb=rtTb->pRtPtr;

else break;}

else{if(rtTb->pLtPtr)rtTb=rtTb->pLtPtr;

else break;}

rtTb->dif=df;

return rtTb;

}

// включення через пошук за двійковим деревом

struct indStrUS\*insBTr(struct lxNode\*pElm,struct indStrUS\*rtTb)

{struct indStrUS\*pInsNod;//,\*pNod;

if(rtTb->pKyStr==NULL)

{rtTb->pKyStr=pElm;

return rtTb;

}

// if(rtTb->pKyStr->ndOp==\_nil)rtTb->pKyStr=pElm;

else{pInsNod=selBTr(pElm,rtTb);

if(pInsNod->dif)

{ndxNds[++nNdxNds]=nilNds;

if(pInsNod->dif<0)pInsNod=pInsNod->pLtPtr=ndxNds+nNdxNds;

else pInsNod=pInsNod->pRtPtr=ndxNds+nNdxNds;

ndxNds[nNdxNds].pKyStr=pElm;

}

}

return pInsNod;

}

syntaxP.cpp

#include "stdafx.h"

#include "..\spLb3\token.h"

#include "syntaxP.h"

extern enum tokPrec opPrFC[256];

extern enum tokPrec opPrGC[256];

extern enum tokPrec opPrFP[256];

extern enum tokPrec opPrGP[256];

enum tokPrec \*opPrF,\*opPrG;

extern char \*oprtrC[], \*oprtrP[], \*oprtrV[],

\*cprC[], \*cprP[], \*cprV[];

extern char \*\*oprtr, \*\*cpr,

modeP, // тип роздільника операторних дужок для Паскаля

modeC, // тип роздільника операторних дужок для С

modeL;

void SxAnInit(char nl)

{char i=0;

switch (nl)

{case 'P': opPrF=opPrFP; opPrG=opPrGP; modeC=0;

modeL=modeP; modeP=1; oprtr=oprtrP; cpr=cprP;

break;

case 'V':

default:

case 'C':opPrF=opPrFC; opPrG=opPrGC; modeP=0;

modeL=modeC; modeC=1; oprtr=oprtrC; cpr=cprC;

}

}

int nxtProd(struct lxNode\*nd, // вказівник на початок масиву вузлів

int nR, // номер кореневого вузла

int nC) // номер поточного вузла

{int n=nC-1; // номер попереднього вузла

enum tokPrec pC = opPrF[nd[nC].ndOp],// передування поточного вузла

\*opPr=opPrG;//F;// nd[nC].prvNd = nd+n;

while(n!=-1) // цикл просування від попереднього вузла до кореню

{if(opPr[nd[n].ndOp]<pC//)// порівняння функцій передувань

&&nd[n].ndOp</\*\_ctbz\*/\_frkz)

{if(n!=nC-1&&nd[n].pstNd!=0) // перевірка необхідності вставки

{nd[nC].prvNd = nd[n].pstNd; // підготовка зв’язків

nd[nC].prvNd->prnNd=/\*nd+\*/nC;} // для вставки вузла

if(opPrF[nd[n].ndOp]==pskw&&nd[n].prvNd==0)nd[n].prvNd = nd+nC;

else nd[n].pstNd = nd+nC;

nd[nC].prnNd=/\*nd+\*/n; // додавання піддерева

return nR;}

if(opPrG[nd[n].ndOp]==pC&&

(nd[n].ndOp==\_brkt||nd[n].ndOp==\_ixbr||nd[n].ndOp==\_opbr||nd[n].ndOp==\_tdbr))

{nd[n].ndOp=(enum tokType)((nd[n].ndOp-\_fork)/2+\_frkz);//09.04.07

nd[nC]=nd[n];

if(nd[nC].prnNd==-1){nR=nC; nd[nR].prnNd=-1;}

else if(opPrF[nd[nd[nC].prnNd].ndOp]==pskw&&nd[nC].ndOp<\_frkz)

nd[nd[nC].prnNd].prvNd = nd+nC;

else if(opPrF[nd[nd[nC].prnNd].ndOp]==pekw&&nd[nC].ndOp==\_opbz)

{nd[nd[nC].prnNd].prvNd =nd+nC;nd[nd[nC].prnNd].pstNd=0;}

return nR;}

/\* if(nd[n].ndOp==\_brkt||nd[n].ndOp==\_ixbr||nd[n].ndOp==\_opbr||nd[n].ndOp==\_tdbr)

{nd[nC].prnNd=n; nd[nC].prvNd=nd[n].pstNd;

nd[n].pstNd->prnNd=nC; nd[n].pstNd= nd+nC;

return nR;}\*/

n=nd[n].prnNd;

opPr=opPrG;} // просування до кореню

// if(n<=) else

nd[nC].prvNd = nd+nR; nd[nR].prnNd=/\*nd+\*/nC; nR = nC; nd[nR].prnNd=-1;

return nR;}

int prCmpr(struct lxNode\*nd, int nn, int nr) //компресія для скорочення графа

{int nR, nN=0, nC=0;

do{

if((nd+nN)->ndOp==\_remL||(nd+nN)->ndOp==\_rem)

{if(nd[nN].pstNd>&nd[nN]&&nd[nN].prnNd!=-1)

{if(nC-nN>1)nd[nN]=nd[nC];

// nN++;

nC++; continue;}

}

// if(nR<-1){nC++; continue;}

if(nr==nC)nr=nN;

if(nd[nC].ndOp==\_brkz&&nd[nC].prvNd==0)

{if(nd[nd[nC].prnNd].prvNd==&nd[nC])

{nd[nC].pstNd->prnNd=nd[nC].prnNd;

nd[nd[nC].prnNd].prvNd=nd[nC].pstNd;}

if(nd[nd[nC].prnNd].pstNd==&nd[nC]) // b 02/06/07

{nd[nC].pstNd->prnNd=nd[nC].prnNd;

nd[nd[nC].prnNd].pstNd=nd[nC].pstNd;}// e 02/06/07

nC++;

}

if(nN!=nC)

{nR=nd[nC].prnNd;

nd[nN]=nd[nC];

//зв'язок з батьківським вузлом

if(nd[nR].prvNd==&nd[nC])

nd[nR].prvNd=&nd[nN];

if(nd[nR].pstNd==&nd[nC])

nd[nR].pstNd=&nd[nN];

if(nd[nN].ndOp>\_cnst){

//зв'язок з лівим наступником

if(nd[nN].prvNd)

nd[nN].prvNd->prnNd=nN;

//зв'язок з правим наступником

if(nd[nN].pstNd)

nd[nN].pstNd->prnNd=nN;

}

// if(nR<nN)nd[nN].prnNd=nR;

// nd[nR].prvNd

}

// else nC

//}else

nN++; nC++;

}while(nC<nn);

return nr;

}