Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №8**

***з курсу «Системне програмування»***

Виконали

студенти групи ІО-73

Бредюк Євген

Захожий Ігор

Бригада №3

Київ 2009

***Тема роботи:*** Побудова, настроювання та використання інтерпретатора цільової мови в транслюючи програмах.

***Мета роботи:*** Одержання навичок використання асемблерних вставок для оптимізації абстрактної машини інтерпретації комп’ютерної мови. Вивчення угод про зв’язки для створення процедур і функцій інтерпретації і звертання до них з операторів мови С.

***Варіант завдання:***

|  |  |  |
| --- | --- | --- |
| № вар. | Вираз, який відтворюється в графі внутрішнього подання | Мова відтво­рення |
| 3 | float b, a[4]; int n; b+=a[n]; | С |

***Завдання:***

Завдання на підготовку до роботи на комп’ютері:

1. Визначити варіант завдання для основних задач за табли­цею 25.1. Визна­чи­ти приклади лексем через файл в папці spLb8 модуля тестування spLb8.cpp.

2. Відповісти на контрольні запитання.

3. Підготувати настройки вхідної мови програмування.

4. Використати структуру елементу struct lxNode з файлу index.h шаб­лону програмного проекту spLb8 для побудови елементу індексу таблиць лексем і визначити.

5. Підготувати програмний модуль контрольної задачі, який виконує заданий варіант з таблиці 25.1 і дозволяє перевірити коректність виконання програм. Для цього організувати пошук в таблиці відповідності типів результатів типам операндів для семантичного аналізу.

Завдання на роботу на комп’ютері

6. Побудувати програмний проект, ввівши програмні модулі у відповідні файли проекту і налагодити синтаксис.

7. Побудувати виконавчий модуль тестової програми і налагодити змістовне виконання програми для перевірки результатів контрольних прикладів.

8. Одержати результати виконання, проаналізувати їх в режимі налагодження і зробити висновки.

9. Продивившись в режимі дизасемблювання фрагменти операцій, що виконуються в замінити оператори мови С/C++ операторами асемблерних вставок, приклад яких наведено в інтерпретації множення з фіксованою точкою.

10. Продемонструвати результати викладачам.

***Короткі теоретичні відомості:***

На основі попередньо визначених, заповнених на етапі лексичного аналізу і уточнених на етапах синтаксичного та семантичного аналізу полів базової структури елемента внутрішнього подан­ня лексем можна організувати виконання операторів програми у внутрішньому поданні комп’ютерної мови крок за кроком. Для цього необхідно розподілити пам’ять для змінних програми в процесі обробки операторів визначення або декларації змінних. Це надає можливість виконання інтерпретації фрагментів програм з константними даними в процесі трансляції, а також організувати налагоджувальний або виконавчий етап інтерпретації комп’ю­тер­ної мови на рівні обробки внутрішнього подання.

1. *ЗАПОВНЕННЯ ТАБЛИЦЬ І ПРАВИЛ ДЛЯ ФУНКЦІОНАЛЬНОЇ ІНТЕРПРЕТАЦІЇ*

Для успішної трансляції виконавчих операцій та операторів в коди на мові Асемблера необхідно розширити таблиці операцій до формату, приданого для генерації кодів в текстовому форматі Асемблера. Формат структури елемента таблиці для семантичного аналізу, інтерпретації та генерації кодів через макроси виведення функцій форматного виведення:

**typedef union** gnDat \_fop(union gnDat\*,union gnDat\*);

**struct** recrdSMA // структура рядка таблиці операцій

{**enum** tokType oprtn;// код операції

**unsigned** oprd1,ln1;//код типу та довжина першого аргументу

**unsigned** oprd2,ln2;//код типу та довжина другого аргументу

**unsigned** res,lnRes;//код типу та довжина результату

\_fop \*pintf; // покажчик на функцію інтерпретації

**char** \*assCd; // покажчик на текст макроса

};

Приклад початку таблиці в проекті наведено нижче:

**struct** recrdSMA ftTbl[]= //таблиця припустимості типів

push ebp ; Запис до стеку старої бази аргументів

mov ebp,esp; Формування нової бази

sub esp,5Ch; Резервування пам’яті локальних даних

push ebx ; Збереження робочих регістрів

push esi

push edi

lea edi,[ebp-5Ch]; Адреса початку локальних даних

mov ecx,17h ; Довжина блоку локальних даних

mov eax,0CCCCCCCCh ; Код неініціалізованих даних

rep stos dword ptr [edi] ; Заповнення блоку

Такий самий блок для 16-бітового режиму мав би вигляд:

push bp ; Запис до стеку старої бази аргументів

mov bp,sp; Формування нової бази

sub sp,5Ch; Резервування пам’яті локальних даних

push bx ; Збереження робочих регістрів

push si

push di

lea di,[bp-5Ch]; Адреса початку локальних даних

mov cx,17h ; Довжина блоку локальних даних

mov ax,0CCCCCCCCh ; Код неініціалізованих даних

rep stos dword ptr [di] ; Заповнення блоку

Блок епілогу функції для 32-бітового налагоджувального режиму має вигляд

mov eax,offset acc32 ; Завантаження покажчика на

; результат

pop edi ; Відновлення робочих регістрів

pop esi

pop ebx

add esp,5Ch ; Звільнення стеку від аргументів

cmp ebp,esp ; Контроль коректності виконання

call \_\_chkesp ; процедури за стеком

mov esp,ebp

pop ebp ; Відновлення старої бази стеку

ret

і забезпечує повернення покажчика на результат типу **union** gnDat\* в акумуляторі eax. А блок епілогу функції для 16-бітового режиму може мати вигляд

mov ax,offset acc32 ; Завантаження покажчика на

; результат

pop di ; Відновлення робочих регістрів

pop si

pop bx

add sp,5Ch ; Звільнення стеку від аргументів

cmp bp,sp ; Контроль коректності виконання

call \_\_chkesp ; процедури за стеком

mov sp,bp

pop bp ; Відновлення старої бази стеку

ret

Тоді код виклику vp1=SmIntrp(nd->prvNd,vp2->\_id) в 32-бітовому режимі налагодження матиме вигляд:

mov ecx,dword ptr [ebp-8]

mov edx,dword ptr [ecx]

push edx ; Підготовка 1-го аргументу

mov eax,dword ptr [ebp+8]

mov ecx,dword ptr [eax+4]

push ecx ; Підготовка 2-го аргументу

call SmIntrp

add esp,8; Компенсація адрес аргументів в стеку

mov dword ptr [ebp-4],eax ; Збереження результату

А код такого самого виклику в 16-бітовому режимі з 32-бітовими адресами може мати вигляд:

mov bx,dword ptr [bp-8]

mov edx,dword ptr [bx]

push dx ; Підготовка 1-го аргументу

mov bx,dword ptr [bp+8]

mov cx,dword ptr [bx+4]

push cx ; Підготовка 2-го аргументу

call SmIntrp

add sp,8; Компенсація адрес аргументів в стеку

mov dword ptr [bp-4],ax ; Збереження результату

Таблиці інтерпретації команд абстрактної машини машинними командами повинні бути розширеними покажчиками на інтерпретуючі функції або текстом інтерпретації окремих операторів та операцій.

1. *ДОЦІЛЬНІСТЬ ВИКОРИСТАННЯ МАКРОСІВ ПРИ РЕАЛІЗАЦІЇ ФРАГМЕНТІВ СКЛАДНИХ ОПЕРАТОРІВ*

В тих випадках, коли машинні команди інструментального процесору мають більшу потужність, ніж команди, які генеруються інструментальним трансляторами доцільно використовувати вставки на мові Асемблера.

Так наприклад, для даних описаних як глобальна змінна та локальні змінні \*vp1, \*vp2 типу

**union** gnDat

{**int** \_id; // цілі 4 байти

**short** \_sd; // цілі 2 байти

**char**  \_cd; // цілі 1 байт

**float** \_fd; // дійсні 4 байти

**double** \_dd; // дійсні 8 байтів

**\_\_int64** \_i8; // цілі 8 байтів

};

**union** gnDat acc32;

**union** gnDat \*vp1, \*vp2;

Оператор мови C/C++

acc32.\_id=vp1->\_id\*vp2->\_id;

можна замінити вставкою на мові Асемблера

**\_asm**{ mov ecx,dword ptr [ebp-4]

mov edx,dword ptr [ebp-8]

mov eax,dword ptr [ecx]

imul eax,dword ptr [edx]

mov dword ptr [acc32],eax

mov dword ptr [acc32+4],edx

}

де результат займає 64 біти.

А оператор мови C/C++

acc32.\_fd=vp1->\_fd\*vp2->\_fd;

можна замінити вставкою на мові Асемблера

**\_asm**{ mov ecx,dword ptr [ebp-4]; адреса 1-го множника

mov edx,dword ptr [ebp-8]; адреса 2-го множника

fld dword ptr [ecx]; копія 1-го множника

fmul dword ptr [edx]; множення на 2-й множник

fstp dword ptr [acc32]; молодші біти результату

}

де результат займає 32 біти.

***Лістинг***

*spLb8.cpp*

#include "stdafx.h"

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

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

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

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

#include "..\spLb5\syntaxP.h"

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

#include "..\spLb7\seman.h"

#include "interp.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);

printf("\n");

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

SmAnlzr(nodes+nr,nr);

prLxTxt(nodes+nr);

printf("\n");

SmIntrp(nodes+nr,1);

printf("\n");

return 0;

}

*interp.h*

union gnDat\* SmIntrp(struct lxNode \* nd, // вказівник на початок масиву вузлів

int nR); // номер кореневого вузла

*interp.cpp*

#include "stdafx.h"

#include "interp.h"

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

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

#include "..\spLb4\index.h"

#include "..\spLb5\syntaxP.h"

#include "..\spLb7\seman.h"

#include "interpF.h"

extern unsigned bcnst32\_buf[MAX\_UCNST];

extern int nInBg, nInCr;

extern struct indStrUS ndxNds[50];

extern struct recrdSMA ftImp;

extern struct recrdSMA ftTbl[170];

extern \_fop \*\_paddf;

//#define iMode 0 // первинний режим

#define iMode 1 // табличний режим

//#define iMode 2 // режим Асемблера

/\*void\* StIntrp(struct lxNode \* nd, // вказівник на корінь дерева вузлів

int nR) // номер кореневого вузла

{//void \*vp1, \*vp2; // розподіл памяті з кореню дерева, але можна зайти з таблиці

if(nd->ndOp==\_nam||nd->ndOp>=\_EOS)

}\*/

union gnDat acc32;

int cc;

union gnDat stk[32];

unsigned char sPtr=0;

void push(union gnDat d)

{stk[sPtr++]=d;

}

void pop(union gnDat \*pd)

{\*pd=stk[--sPtr];

}

unsigned nLcCr=0;

void clDtLst(struct lxNode \* nd, int tp, unsigned \*pD)

{if(nd->ndOp==\_comma)

{clDtLst(nd->prvNd, tp, pD);

switch(tp)

{case \_f: cnvTo\_f(nd->pstNd->dataType,

(union gnDat \*)(pD+nLcCr),(union gnDat \*)nd->pstNd->pstNd); break;//

case \_ui: case \_si: cnvTo\_i(nd->dataType,

(union gnDat \*)(pD+nLcCr),(union gnDat \*)nd->pstNd->pstNd);

}nLcCr++;//

}else if(nd->ndOp==\_srcn)

{switch(tp)

{case \_f: cnvTo\_f(nd->dataType,

(union gnDat \*)(pD+nLcCr),(union gnDat \*)nd->pstNd); break;//

case \_ui: case \_si: cnvTo\_i(nd->dataType,

(union gnDat \*)(pD+nLcCr),(union gnDat \*)nd->pstNd);

}nLcCr++;//

}

}

struct lxNode \*trmGrdt(struct lxNode \* nd)

{if(nd->ndOp!=\_nam)return trmGrdt(nd->prvNd);

return nd;

}

void prAss(struct lxNode \* nd)

{prLxTxt(nd);

if(nd->prvNd->dataType>=\_f)printf(" => %7.3g ->",acc32.\_fd);

else printf(" => %7d ->",acc32.\_id);

prLxTxt(nd->prvNd);printf("\n");

}

char dfnFlg=0;

union gnDat\* SmIntrp(struct lxNode \* nd, // вказівник на корінь дерева вузлів

int incR) // кількість повторень

{union gnDat \*vp1, \*vp2;

char\*name; struct lxNode \*nt;

struct indStrUS \*pRtNdx;

if(((nd->ndOp>=\_void&&nd->ndOp<=\_string)

||nd->ndOp>=\_EOS)&&nd->ndOp!=\_remL)

{

if(nd->ndOp>=\_void&&nd->ndOp<\_fork)dfnFlg=1;

if(nd->prvNd&&nd->ndOp==\_ass)

{vp1=SmIntrp(nd->prvNd,1);

if(nd->pstNd->ndOp==\_tdbz)

{nt=trmGrdt(nd->prvNd); nLcCr=0;

clDtLst(nd->pstNd->pstNd,nt->dataType&0x7FF,

(unsigned\*)nt->pstNd);

// if(nd->prvNd->ndOp==\_ixbz)return vp2;

}else{

// vp1=SmIntrp(nd->prvNd,1);

vp2=SmIntrp(nd->pstNd,1);

if(nd->prvNd->dataType>=\_f)

{acc32.\_fd=cnvTo\_f(nd->pstNd->dataType, vp1, vp2);

if(nd->prvNd->dataType>\_f)

acc32.\_dd=cnvTo\_d(nd->pstNd->dataType, vp1, vp2);

}else{if(nd->pstNd->dataType<\_f) acc32.\_id=vp1->\_id=vp2->\_id;

else acc32.\_id=vp1->\_id=vp2->\_fd;}}

prAss(nd);

return &acc32;}

if(nd->prvNd&&nd->ndOp!=\_ixbz)

{vp1=SmIntrp(nd->prvNd,1);

if((nd->ndOp==\_cln||nd->ndOp==\_else)&&cc!=0)

{if(nd->dataType!=nd->prvNd->dataType)

{if(nd->dataType>=\_f)vp1->\_fd=vp1->\_id;

else vp1->\_id=vp1->\_fd;}

return vp1;}

}

if(nd->pstNd)

{if(nd->ndOp>\_ass&&nd->ndOp<=\_frkz&&nd->prvNd

&&nd->prvNd->ndOp>\_cnst&&nd->pstNd->ndOp>\_cnst)

push(acc32);

vp2=SmIntrp(nd->pstNd,1);

if(nd->ndOp>\_ass&&nd->ndOp<=\_frkz&&nd->prvNd

&&nd->prvNd->ndOp>\_cnst&&nd->pstNd->ndOp>\_cnst)

{--sPtr; vp1=stk+sPtr;}

}

if(nd->ndOp&&nd->ndOp==\_ixbz)

{vp1=SmIntrp(nd->prvNd,vp2->\_id);

// if(nodes[nd->prnNd].pstNd->ndOp==\_tdbz)

}

//вирівнювання типів

if(nd->ndOp>=\_void&&nd->ndOp<\_fork)

{dfnFlg=0;

return NULL;

}

if(dfnFlg!=0||(nd->ndOp>=\_eosP&&nd->ndOp<=\_EOS))

return NULL;

if(//nd->dataType>=\_f ||

(nd->prvNd!=0&&nd->prvNd->dataType>=\_f)

||(nd->pstNd!=0&&nd->pstNd->dataType>=\_f))

{if(nd->prvNd!=0&&nd->prvNd->dataType<\_f&&nd->ndOp!=\_cln)vp1->\_fd=vp1->\_id;

if(nd->pstNd!=0&&nd->pstNd->dataType<\_f&&nd->ndOp!=\_ixbz)vp2->\_fd=vp2->\_id;

// сюди вставляти потрібні операції

switch(nd->ndOp)

{case \_asAdd:

if(nd->prvNd)

acc32.\_fd=vp1->\_fd+=vp2->\_fd;

prAss(nd);

break;

case \_lt:

if(nd->prvNd)

acc32.\_id=vp1->\_fd<vp2->\_fd;

break;

case \_gt:

if(nd->prvNd)

acc32.\_id=vp1->\_fd>vp2->\_fd;

break;

case \_add:

if(nd->prvNd)

#if iMode==1

{ftImp.oprd1=nd->dataType; ftImp.ln1=nd->resLength;

ftImp.oprd2=nd->dataType; ftImp.ln2=nd->resLength;

ftImp.oprtn=nd->ndOp;

struct recrdSMA\*

pftImp = selBin(&ftImp, ftTbl, 179);

if(pftImp)

acc32=pftImp->pintf(vp1,vp2);/\*\_paddf\*/

}

#else

acc32.\_fd=vp1->\_fd+vp2->\_fd;

#endif

break;

case \_sub:

if(nd->prvNd)

acc32.\_fd=vp1->\_fd-vp2->\_fd; break;

case \_mul:

if(nd->prvNd)

\_asm{ mov ecx,dword ptr [ebp-4]; адреса 1-го множника

mov edx,dword ptr [ebp-8]; адреса 2-го множника

fld dword ptr [ecx]; копія 1-го множника

fmul dword ptr [edx]; множення на 2-й множник

fstp dword ptr [acc32]; молодші біти результату

}

break;

case \_div:

if(nd->prvNd)

acc32.\_fd=vp1->\_fd/vp2->\_fd; break;

case \_cln:

if(cc==0)

acc32.\_fd=vp2->\_fd; break;

case \_qmrk:

if(nd->prvNd)cc=vp1->\_id;

if(cc)acc32.\_fd=vp2->\_fd; break;

case \_else:

if(cc==0)

acc32.\_fd=vp2->\_fd; break;

case \_if:

if(nd->prvNd)

if(vp1->\_fd) cc=vp1->\_id; break;

case \_ixbz:

if(nd->prvNd)

acc32.\_fd=((float\*)vp1)[vp2->\_id];break;

case \_brkz:

if(nd->prvNd)

acc32.\_fd=vp2->\_fd;

}}else{

switch(nd->ndOp)

{case \_asAdd:

if(nd->prvNd)

acc32.\_id=vp1->\_id+=vp2->\_id;

prAss(nd);

break;

case \_ne:

if(nd->prvNd)

acc32.\_id=vp1->\_id!=vp2->\_id;

break;

case \_add:

if(nd->prvNd)

acc32.\_id=vp1->\_id+vp2->\_id;

break;

case \_sub:

if(nd->prvNd)

acc32.\_id=vp1->\_id-vp2->\_id; break;

case \_mul:

if(nd->prvNd)

#if iMode==2

\_asm {mov ecx,dword ptr [ebp-4]

mov edx,dword ptr [ebp-8]

mov eax,dword ptr [ecx]

imul eax,dword ptr [edx]

mov dword ptr [acc32],eax

mov dword ptr [acc32+4],edx

}

#else

\_asm{ mov ecx,dword ptr [ebp-4]

mov edx,dword ptr [ebp-8]

mov eax,dword ptr [ecx]

imul eax,dword ptr [edx]

mov dword ptr [acc32],eax

mov dword ptr [acc32+4],edx

}

#endif

break;

case \_div:

if(nd->prvNd)

acc32.\_id=vp1->\_id/vp2->\_id; break;

case \_cln:

if(cc==0)

acc32.\_fd=vp2->\_id; break;

case \_qmrk:

if(nd->prvNd) cc=vp1->\_id;

if(cc)acc32.\_id=vp2->\_id; break;

case \_mod:

if(nd->prvNd)

acc32.\_id=vp1->\_id%vp2->\_id; break;

case \_orB:

if(nd->prvNd)

acc32.\_id=vp1->\_id|vp2->\_id; break;

case \_andB:

if(nd->prvNd)

acc32.\_id=vp1->\_id&vp2->\_id; break;

case \_xorB:

if(nd->prvNd)

acc32.\_id=vp1->\_id^vp2->\_id; break;

case \_or:

if(nd->prvNd)

acc32.\_id=vp1->\_id||vp2->\_id; break;

case \_and:

if(nd->prvNd)

acc32.\_id=vp1->\_id&&vp2->\_id; break;

case \_ixbz:

if(nd->prvNd)

acc32.\_id=((unsigned\*)vp1)[vp2->\_id]; break;

case \_dcr:

if(nd->prvNd==0)

acc32.\_id=--(vp2->\_id);

else acc32.\_id=(vp1->\_id)--; break;

case \_inr:

if(nd->prvNd==0)

acc32.\_id=++(vp2->\_id);

else acc32.\_id=(vp1->\_id)++; break;

}}

// pRc=selBin(lPrv, tpTbl, 21);

}else if(nd->ndOp==\_nam)

{if(nd->dataType!=\_v)

{if(nd->pstNd==0)

{pRtNdx=selBTr(nd,ndxNds);// якщо не знайдено - неописане імя

name=(char\*)pRtNdx->pKyStr->prvNd;/\*\*/

nd->pstNd=pRtNdx->pKyStr->pstNd;

if(nd->pstNd==0)

{nd->pstNd=(struct lxNode \*)(bcnst32\_buf+nInCr);

while(incR--)

\*(bcnst32\_buf+nInCr++)=0xCCCCCCCC;

// nInCr+=incR;

}}

}

return vp1=(union gnDat\*)nd->pstNd;

}else if(nd->ndOp==\_srcn)//\_cnst

{return vp1=(union gnDat\*)nd->pstNd;

}else if(nd->ndOp==\_whileP)//\_cnst

{do{

vp1=SmIntrp(nd->prvNd->pstNd,1);

if(vp1->\_id)vp2=SmIntrp(nd->pstNd,1);

else return vp1;

}while(vp1->\_id);

}

return &acc32;

}

*interpF.h*

float cnvTo\_f(int td, union gnDat \*p1, union gnDat \*p2);

double cnvTo\_d(int td, union gnDat \*p1, union gnDat \*p2);

int cnvTo\_i(int td, union gnDat \*p1, union gnDat \*p2);

union gnDat \_fadd(union gnDat\*,union gnDat\*);

*interpF.cpp*

#include "stdafx.h"

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

#include "..\spLb7\seman.h"

#include "interpF.h"

extern union gnDat acc32;

extern int cc;

float cnvTo\_f(int td, union gnDat \*p1, union gnDat \*p2)

{switch(td)

{case \_d:

acc32.\_fd=p1->\_fd=p2->\_dd; break;

case \_f:

acc32.\_fd=p1->\_fd=p2->\_fd; break;

default:

acc32.\_fd=p1->\_fd=p2->\_id;

}

return acc32.\_fd;

}

double cnvTo\_d(int td, union gnDat \*p1, union gnDat \*p2)

{switch(td)

{case \_d:

acc32.\_dd=p1->\_fd=p2->\_dd; break;

case \_f:

acc32.\_dd=p1->\_fd=p2->\_fd; break;

default:

acc32.\_dd=p1->\_fd=p2->\_id;

}

return acc32.\_dd;

}

int cnvTo\_i(int td, union gnDat \*p1, union gnDat \*p2)

{switch(td)

{case \_d:

acc32.\_id=p1->\_id=p2->\_dd; break;

case \_f:

acc32.\_id=p1->\_id=p2->\_fd; break;

default:

acc32.\_id=p1->\_id=p2->\_id;

}

return 0;

}

union gnDat \_fadd(union gnDat \*p1,union gnDat \*p2)

{acc32.\_fd=p1->\_fd+p2->\_fd;

return acc32;

}

*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;

}

*langio.cpp*

#include "stdafx.h"

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

#include "automat.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_SCNST 1024 // граничний обсяг символьних образів

struct fileInp

{char \* src\_name;

FILE \* stream;

} include\_files[20];

int x=0, y=0, f=0, cl;

//extern

unsigned nImBg=0, nImCr=0; // Початковий та поточний номери образів в буфері

//extern

char imgBuf[MAX\_SCNST]; // буфер вхідних образів

extern struct lxNode nodes[MAX\_SCNST];

FILE \*fi\_progr;

void opFls(char\* fn)

{ //search in startup dir

fi\_progr=fopen(fn,"r+b");

if (!fi\_progr)

{printf("\nImpossible to open file: %s\n",fn);exit(2);}

include\_files[f].stream=fi\_progr;

include\_files[f].src\_name=fn;

f++;

}

FILE \*fi\_progrC;

void opFlsC(char\* fn)

{/\* //search in startup dir

char \*ps, s[30]="";

ps=strcat(s,"..\\spLb9c\\");//"");

ps=strcat(ps,fn);\*/

fi\_progrC = fopen(fn,"wb");

if (!fi\_progrC)

{printf("\nImpossible to open file: %s\n",fn);exit(2);}

}

void WriteCod(char\*cd)

{fprintf(fi\_progrC,cd);

}

char ReadLtr(void)

{static char c=0;

cl=fgetc(fi\_progr);

if (c==13||c==10)

{x=0; y++;

if(cl==10)cl=fgetc(fi\_progr);

}

x++;

c=char(cl);

if (cl==EOF)c=0; //c=13;

imgBuf[nImCr++]=c;//cl;

return c;

}

void lxInit(int lxNmb,enum ltrType cl)

{nodes[lxNmb].x=x;

nodes[lxNmb].y=y;

nodes[lxNmb].f=f;

nodes[lxNmb].ndOp=\_nil;

nodes[lxNmb].prnNd=NULL;

nodes[lxNmb].prvNd=(struct lxNode \*)(imgBuf+nImCr);

if(cl<nc/\*&&nImCr!=0\*/)nodes[lxNmb].prvNd=(struct lxNode \*)(imgBuf+nImCr-1);

nodes[lxNmb].pstNd=NULL;

nodes[lxNmb].dataType=0;

nodes[lxNmb].resLength=0;

nodes[lxNmb].stkLength=0;

}

void frmCns(enum autStat s, int lxNmb)

{;

}

void eNeut(int lxNmb)

{;

}

void dGroup(int lxNmb)

{;

}

*lexan.cpp*

#include "stdafx.h"

#include "automat.h"

#include "langio.h"

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

#include "tables.h"

#include "index.h"

#include <stdlib.h>

extern enum ltrType \*ltCls; // уточнюються для версій та режимів

extern enum autStat nxtStsC[Eo+1][ltrcode+1];

extern enum autStat nxtStsP[Eo+1][ltrcode+1];

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

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

char \*\*oprtr, \*\*cpr;

char modeP=0, // тип роздільника операторних дужок для Паскаля

modeC=1, // тип роздільника операторних дужок для С

modeL;

extern int x, y, f;

extern unsigned nImBg, nImCr; // Початковий та поточний номери образів в буфері

extern char imgBuf[]; // буфер вхідних образів

extern enum tokType \*dlCds;

extern enum ltrType \*ltCls;

unsigned nNode=0;

extern struct lxNode nodes[100];

extern unsigned nNdxNds;

extern struct indStrUS \*pRtNdx, ndxNds[50];

struct recrdKWD \*tablKWD;

extern struct recrdKWD tablKWDC[67];

extern struct recrdKWD tablKWDP[67];

enum autStat //\*nxtStsR=&nxtStsC[0][0],

nxtSts[Eo+1][ltrcode+1];//=&nxtStsR;//(enum autStat\*\*)nxtStsC;

void LxAnInit(char nl)

{char i=0;

switch (nl)

{case 'P':modeL=modeP;

oprtr=oprtrP, cpr=cprP;

tablKWD = tablKWDP;

for(i=0;i<=Eo;i++)for(char j=0;j<=ltrcode;j++)

nxtSts[i][j]=nxtStsP[i][j];

break;

case 'V':

default:

case 'C':modeL=modeC;

oprtr=oprtrC, cpr=cprC;

tablKWD = tablKWDC;

for(i=0;i<=Eo;i++)for(char j=0;j<=ltrcode;j++)

nxtSts[i][j]=nxtStsC[i][j];

}

}

// функція лексичного аналізу чергової лексеми

int cntMdB=0;

int LxAnlzr(void)

{//static int lxNmb=0;

static enum autStat s=S0, sP; // поточний та попередній стан лексеми

enum autStat SP;

static enum ltrType c; // клас чергової літери

static char l=1; // чергова літера (початок фыктивний)

struct recrdKWD\* pRt;

int s1, c1;

char l1, l0; // чергова літера

lxInit(nNode,c);

do {SP=

sP=s; l1=l; // запам'ятовування стану

l0=l=ReadLtr(); // читання літери

c1=c=ltCls[l]; // визначення класу літери

if(s==Scl&&c!=dlmeorml)continue;

s=nxtSts[s][c];//[c<dlmaux?c:dlmaux];// стан лексеми

s1=s;

if(s==Scr)continue;

if((sP==S2||sP==S3)&&(c>nc&&c<dlmeormr))

{// пошук в таблиці групових роздільників

imgBuf[nImCr]=0;

pRt=selBin(imgBuf+nImBg, tablKWD, 67);

if(pRt!=NULL)

{nodes[nNode].ndOp=pRt->func;

if(pRt->func==\_remL)

{s=Scr; nodes[nNode].pstNd=(struct lxNode \*)(imgBuf+nImCr);

continue;}

if(pRt->func==\_rem)

{s=Scl;} continue;

break;}else

{if(sP!=S3)

{nodes[nNode].ndOp=/\*(enum tokType)\*/dlCds[l1];

imgBuf[nImBg]=imgBuf[nImBg+1];

}else{imgBuf[nImBg]=imgBuf[nImBg+1]; nImCr--;}

nImCr--;

sP=S0;

s=nxtSts[sP][c];//[c<dlmaux?c:dlmaux];// стан лексеми

return nNode++;

}

}

s1=s;}while(s!=S0&&s!=S2&&!((sP==S0||sP==S2||sP==S3)&&s<S2)); // перевірка кінця лексеми

s1=sP;

switch(sP)

{case Scr: case Scl:

imgBuf[nImCr++]=0;

// ((char\*)(nodes[nNode].prvNd))--;

nImBg=nImCr;

break;

case S2:

case S0:

/\* if(s==S0)

// dGroup(nNode);// аналіз групових роздільників

{//imgBuf[nImBg]=l1;

imgBuf[++nImCr]=0;//nImCr++;

}\*/

if(nodes[nNode].ndOp!=\_nil)

{nImCr=nImBg; l=' '; return nNode++;}

// if(sP!=S0)

nodes[nNode].ndOp=/\*(enum tokType)\*/dlCds[l1];//dlCds[l1];

if(nodes[nNode].ndOp==\_opbr&&(nodes[nNode-1].ndOp==\_ass||cntMdB))

{cntMdB++;nodes[nNode].ndOp=\_tdbr;}

if(nodes[nNode].ndOp==\_ocbr&&cntMdB)

{nodes[nNode].ndOp=\_tcbr;cntMdB--;}

if(nodes[nNode].ndOp!=\_nil)

// &&imgBuf[nImBg]==)

{nodes[nNode].prvNd=NULL;

if(nImBg+1!=nImCr)

// if(dlCdsC[l0]!=\_nil||ltClsC[l0]==dlmaux||ltClsC[l0]==dlmeormr)

{imgBuf[nImBg]=imgBuf[nImBg+1];

if(s!=S0){nImCr--;

imgBuf[nImBg]=imgBuf[nImCr];nImCr=nImBg+1;}// 04.07.07

else nImCr=nImBg;}

return nNode++;}

else if(ltCls[imgBuf[nImBg]]==dlmaux/\*&&ltClsC[imgBuf[nImBg]]>nc\*/)

{imgBuf[nImBg]=imgBuf[nImBg+1];

nImCr--;}

return nNode;

case S1n:// пошук ключових слів та імен

imgBuf[nImCr-1]=0;

// пошук у таблиці ключів;

if(\*(imgBuf+nImBg)==13)nImBg++;

pRt=selBin(imgBuf+nImBg, tablKWD, 67);

if(pRt){nodes[nNode].ndOp=pRt->func;

// якщо знайдено

nodes[nNode].prvNd=NULL;

nImCr=nImBg;

if(c!=dlmeormr&&c!=dlmaux)imgBuf[nImCr++]=l;

return nNode++;}

// якщо не знайдено

nodes[nNode].ndOp=\_nam;

insBTr(nodes+nNode,pRtNdx);

nImBg=nImCr;

if(c!=dlmeormr&&c!=dlmaux)imgBuf[nImCr++]=l;

break;

default: // не дійшли до класифікованих помилок

case Eu: case Ec: case Ep: case Eq: case En: case Eo:// обробка помилок

eNeut(nNode); // фіксація помилки

case S1c: case S2c: case S1p: case S2s: // формування констант

imgBuf[nImCr-1]=0;

nodes[nNode].resLength=sP;// frmCns(sP, nNode); break;

nodes[nNode].ndOp=\_srcn; nodes[nNode].dataType=sP;

insBTr(nodes+nNode,pRtNdx);

nImBg=nImCr;

if(c!=dlmeormr&&c!=dlmaux)imgBuf[nImCr++]=l;

break;

case S3: nImCr=nImBg;

nodes[nNode].prvNd=NULL;

imgBuf[nImBg]=imgBuf[nImBg+2];

s1=s;

if(s!=S0)nImCr=nImBg+1;//else nImCr--;

}

return nNode++;

}

*lexcalc.cpp*

#include "stdafx.h"

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

#include "..\spLb4\automat.h"

#include "..\spLb5\syntaxP.h"

#include "lexcalc.h"

#include <string.h>

enum ltrTypeS ltrClsV[256] = // Початок таблиці класифікаторів для Verilog HDL

{ncS,ncS,ncS,ncS,ncS,ncS,ncS,ncS, ncS,dlmauxS,dlmauxS,ncS,ncS,dlmauxS,ncS,ncS, //16

ncS,ncS,ncS,ncS,ncS,ncS,ncS,ncS, ncS,ncS,ncS,ncS,ncS,ncS,ncS,ncS, //32

dlmauxS,dlmgrop1,ltrstrlmS,dlmunopS, ltrnmelmS,dlmunopS,dlmgrop2,ltrcnslmS,

dlmunopS,dlmunopS,dlmgrop3,ltrsignS, dlmunopS,ltrsignm,dldotS,ltrstrlmS, //48

d0,d1,d2,d3,d4,d5,d6,d7,

d8,d9,dlmunopS/\*dlmgrop\*/,dlmunopS/\*dlmbrlst\*/,dlmgrop4,dlmgrop5,dlmgrop6,dlmunopS,// 64

dlmunopS,xA,xB,xC, xD,ltrexpltS,xF,ltrnmelmS,

cH,ltrnmelmS,ltrnmelmS,ltrnmelmS, ltrnmelmS,ltrnmelmS,ltrnmelmS,cO,//80

ltrnmelmS,ltrnmelmS,ltrnmelmS,ltrnmelmS, ltrnmelmS,ltrnmelmS,ltrnmelmS,ltrnmelmS,

ltrhxdgtS/\*ltrnmelm\*/,ltrnmelmS,xZ/\*ltrnmelm\*/,dlmunopS, ltrtrnfmS,dlmunopS,dlmgrop7,dlmund,//96

dlmunopS,xA,xB,xC, xD,ltrexpltS,xF,ltrnmelmS,

cH,ltrnmelmS,ltrnmelmS,ltrnmelmS, ltrnmelmS,ltrnmelmS,ltrnmelmS,cO,//112

ltrnmelmS,ltrnmelmS,ltrnmelmS,ltrnmelmS, ltrnmelmS,ltrnmelmS,ltrnmelmS,ltrnmelmS,

ltrhxdgtS/\*ltrnmelm\*/,ltrnmelmS,xZ/\*ltrnmelm\*/,dlmunopS, dlmgrop8,dlmunopS,dlmgrop9,ncS//,128

//...

};extern struct imElem kwTab[LENGTHKW+1]; // для Verilog HDL

unsigned bcnst32\_buf[MAX\_UCNST];//={0,0,-1,-1,0,1,0};для Verilog HDL

// \*bcnst32\_beg, \*bcnst32\_cur;

//extern struct imElem dlmTb23V[10][4];

char bcnst8\_buf[MAX\_CCNST];//={0,1,2,3};//, // buffer of optimized bit constant

int //x=0, y=0, f=0, // координати розміщення тексту в файлі

nImBg=0, nImCr=0,// Початковий та поточний номери образів в буфері

nInBg=7, nInCr=7,// Початковий та поточний номери 32-бітових образів в буфері

nIcBg=4, nIcCr=4,// Початковий та поточний номери 32-бітових образів в буфері

fPrdfCnstr=0, // Номер напередвизначеної конструкції

fLnkCnstr=0, //

brkCntr=0, brkStck[BRK\_LIM];

extern char imgBuf/\*imageBuf\*/[MAX\_SCNST]; // буфер вхідних образів

//=============================================================

union ui64\_t value, valuea, value1, value1a, value0;//, \*valptr;

//long // можлива занадто велика точність

double fvalue64, fvalueR;//, \*fvlptr;

/\* у Visual C/C++ 6.0 без сервіс-паків не працює

struct up80\_t{unsigned \_\_int64 mant;

unsigned short pow;} c2p32={0x8000000000000000l,0x401f};

union ui80\_t{long double lfvalue;

struct up80\_t up80;} c2pn32, fvalue;\*/

unsigned \_\_int64 c2p32=0x41f0000000000000l;

union ui80\_t{double lfvalue;

\_\_int64 up80;} c2pn32, fvalue;

unsigned char aTb[4]={2,10,8,16};

int nPwr=0, n32=0;

unsigned FMode=0;//nBit=0,

int /\*LexNode::\*/ convNum(struct lxNode \*nd //, вказівник на початок масиву вузлів

// unsigned n, int disp, enum ltrTypeS ltrCls[256]

)

{unsigned char aSS, sSS, c=0xff;

int i, dispP, disp=(char\*)(nd->prvNd)-imgBuf;

nPwr=0; n32=0; FMode=0;

// ndOp = n;;

nd->pstNd=(struct lxNode \*)(bcnst32\_buf+nInBg);

if(nd->resLength!=S2s)disp=convInt(disp, 10, 10, 0xFFFFFFFF);

switch(nd->resLength)

{case S1c: nd->resLength=32; break;

case S1p: case S2c: nd->resLength=64;

// if(n32<(((ln)/32)<<1))n32=((ln)/32)<<1;

if((FMode&FPWR)!=0||ltrClsV[imgBuf[disp]]==ltrexplt)

{value.s64.lo32=bcnst32\_buf[nInBg];

// if(n32==0)

value.s64.hi32=0;

// else value.s64.hi32=bcnst32\_buf[nInBg+2];

fvalueR/\*.lfvalue\*/=(\_\_int64)value.ui64;

c2pn32.up80 = c2p32;

fvalue64=1.0;

// fvalue.up80.mant=value.ui64;

// fvalue.up80.pow=0x403f;

for(i=0;i<n32;i+=2)

{fvalue64\*=c2pn32.lfvalue;

value.s64.lo32=bcnst32\_buf[nInBg+i+2];

fvalueR+=((\_\_int64)value.ui64)\*fvalue64;

}

fvalue.lfvalue=fvalueR;

if(ltrClsV[imgBuf[disp]]==ltrexpltS){FMode&=~FPWR;

dispP=disp+1; if((ltrClsV[imgBuf[dispP]]&0xf0)==ltrsignS)disp=disp+1;

disp=convInt(disp+1, 10, 10, 0);

if(value.s64.lo32){

if(imgBuf[dispP]!='-')nPwr+=value.s64.lo32;

else nPwr-=value.s64.lo32;}}

if(FMode&FMIN)fvalue.lfvalue=-fvalue.lfvalue;

FMode&=~FMIN;

if(nPwr>0)while(nPwr){nPwr--; fvalue.lfvalue\*=1e1l;/\*/=1e-1l\*/}

else while(nPwr){nPwr++; fvalue.lfvalue/=1e1l;/\*=1e-1l\*/}

value.fi64=fvalue.lfvalue;

bcnst32\_buf[nInBg+0]=value.s64.lo32;

bcnst32\_buf[nInBg+1]=value.s64.hi32;

n32=0;

}

break;

case S2s: strcpy(bcnst8\_buf+nIcBg,imgBuf+nImBg+1);

nd->pstNd=(struct lxNode \*)(bcnst8\_buf+nIcBg);

nIcBg=nIcCr=nIcBg+strlen(imgBuf+nImBg+1);

\*(bcnst8\_buf+nIcCr-1)=0; // вилучити повторення

return 0;

break;

case S3c:

nd->resLength=value.s64.lo32;

if(nd->resLength==0xFFFFFFFF)nd->resLength=32;

sSS=(ltrClsV[imgBuf[disp+1]]&14)>>1; if(sSS>4)sSS-=4;

aSS=aTb[sSS-1];

convInt(disp+2, aSS, sSS, nd->resLength);

}

nd->stkLength=(n32+2)<<4; nInCr=nInBg-n32-2;

do{if(nInCr<0)break; c=0==0;

for(i=n32+1;i>=0;i--)

if(!(c&=bcnst32\_buf[nInBg+i]==bcnst32\_buf[nInCr+i]))break;

nInCr--;} while(c==0);

if(c==0){nInCr=nInBg+=n32+2;

nImBg=nImCr;}

else {nd->pstNd=(struct lxNode \*)(bcnst32\_buf+nInCr+1); nInCr=nInBg;

nImBg=nImCr;}// це треба б мінімізувати, якщо вхідні коди співпадають

return n32;

}

//------------------------------------------------------------------------

/\* converting source program to lexeme string "wstr" \*/

int convInt(unsigned nc, unsigned char aSS, unsigned char sSS, unsigned nb)

{//try to skip space after constant base like 'b, 'd, 'h

unsigned char cwrk; int i;

unsigned difV, difVM, nBit=0;//, difV, difVM;

value.ui64=value1.ui64=0; //nPwr=0;

if(nb==0xFFFFFFFF)value.s64.lo32=nb;

n32=0; bcnst32\_buf[nInBg+1]=bcnst32\_buf[nInBg]=0;

/\* if(lwstr<=2||wstr[lwstr].code!=div\_diez

||(wstr[lwstr-1].code!=div\_equ&&wstr[lwstr-1].code!=kws\_leeq))

while (c==32||c==9) Read\_symbol();

if (c=='`')

{scan\_identifier();buffer[0]=0;}\*/

while(((cwrk=ltrClsV[imgBuf/\*imageBuf\*/[nc]])&0xcf)<aSS||imgBuf[nc]=='.'

// ||c=='\_'||c=='.'||c=='?'

// ||c=='X'||c=='x'||c=='Z'||c=='z'// x=0, z=-1

)

{if(imgBuf[nc]!='.')

{if(aSS==10)

{difV=(cwrk&0xf);difVM=0;

if((cwrk&0x20)!=0)difVM = 15&(aSS-1);

if(FMode&FPWR)nPwr--;

for(i=0;i<=n32;i+=2)

{value.s64.lo32=bcnst32\_buf[i+nInBg]; value.s64.hi32=0;

value1.s64.lo32=bcnst32\_buf[i+nInBg+1]; value1.s64.hi32=0;

value.ui64 = value.ui64 \* aSS + difV;

value1.ui64 = (value1.ui64 \* sSS) | difVM;

bcnst32\_buf[i+nInBg]=value.s64.lo32;

bcnst32\_buf[i+nInBg+1]=value1.s64.lo32;

difV=value.s64.hi32;

difVM=value1.s64.hi32;

}

if(difV|difVM){n32++; n32++;

bcnst32\_buf[i+nInBg]=difV;bcnst32\_buf[i+nInBg+1]=difVM;}

}else

{difV = cwrk&(aSS-1); if(cwrk==xZ/\*ltrhxdgt\*/)difV = aSS-1;

difVM=0; nBit+=sSS;

if((cwrk&0xfe/\*20\*/)==ltrhxdgt/\*0\*/)difVM = 15&(aSS-1);

for(i=0;i<=n32;i+=2)

{value.s64.lo32=bcnst32\_buf[i+nInBg]; value.s64.hi32=0;

value1.s64.lo32=bcnst32\_buf[i+nInBg+1]; value1.s64.hi32=0;

value.ui64 = (value.ui64 << sSS) + difV;

value1.ui64 = (value1.ui64 << sSS) | difVM;

bcnst32\_buf[i+nInBg]=value.s64.lo32;

bcnst32\_buf[i+nInBg+1]=value1.s64.lo32;

difV=value.s64.hi32;

difVM=value1.s64.hi32;

}

if(difV|difVM){n32++; n32++;

bcnst32\_buf[i+nInBg]=difV; bcnst32\_buf[i+nInBg+1]=difVM;}

}}else if(imgBuf[nc]=='.')

{FMode|=FPWR;} // треба захиститись вiд повторної крапки

// if (strlen(imageBuf+nc)<=1025)strncat(imageBuf+nc,nc/\*&c\*/,1);

//Read\_symbol();

nc++;}

if(nb/\*ulength.ui64\*/!=0xFFFFFFFF){//ulength.s64.lo32=32;

valuea.ui64=1; valuea.ui64<<=((nBit-1)&63);

if (aSS!=10&&nBit</\*ulength.s64.lo32\*/nb&&(valuea.ui64&value1.ui64)!=0)

{value1a.ui64=1; value1a.ui64<<=(nb/\*ulength.s64.lo32\*/);//&63

value1a.ui64=value1a.ui64-valuea.ui64;

bcnst32\_buf[i+nInBg-1]|=value1a.ui64;

if(value1a.s64.hi32){n32++; n32++; bcnst32\_buf[i+nInBg+1]=value1a.s64.hi32;

bcnst32\_buf[i+nInBg]=0;

if((valuea.ui64&value.ui64)!=0)bcnst32\_buf[i+nInBg]|=value1a.s64.hi32;}

if((valuea.ui64&value.ui64)!=0)bcnst32\_buf[i+nInBg-2]|=value1a.ui64;

}else if(nBit>nb/\*ulength.s64.lo32\*/&&aSS!=10)

{bcnst32\_buf[i+nInBg-1]&=valuea.s64.lo32-1;

bcnst32\_buf[i+nInBg-2]&=valuea.s64.lo32-1;

}else

if(i=((nb/\*ulength.s64.lo32\*/+31)>>5)-1-n32/2)

while(i-->0)

{n32+=2; bcnst32\_buf[n32+nInBg]=0;

bcnst32\_buf[n32+nInBg+1]=0;}//?????

}

return nc;}

*operSpc.cpp*

#include "stdafx.h"

#include "syntaxP.h"

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

#include <stdio.h>

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

{{\_nil,NULL,NULL,0,0,0,0,0,NULL,0},{\_nil,NULL,NULL,0,0,0,0,0,NULL,0},

{\_nil,NULL,NULL,0,0,0,0,0,NULL,0},{\_nil,NULL,NULL,0,0,0,0,0,NULL,0},

{\_nil,NULL,NULL,0,0,0,0,0,NULL,0},{\_nil,NULL,NULL,0,0,0,0,0,NULL,0},

{\_nil,NULL,NULL,0,0,0,0,0,NULL,0},{\_nil,NULL,NULL,0,0,0,0,0,NULL,0},

{\_nil,NULL,NULL,0,0,0,0,0,NULL,0},{\_nil,NULL,NULL,0,0,0,0,0,NULL,0},

{\_nil,NULL,NULL,0,0,0,0,0,NULL,0},{\_nil,NULL,NULL,0,0,0,0,0,NULL,0},

{\_nil,NULL,NULL,0,0,0,0,0,NULL,0},{\_nil,NULL,NULL,0,0,0,0,0,NULL,0},

{\_nil,NULL,NULL,0,0,0,0,0,NULL,0},{\_nil,NULL,NULL,0,0,0,0,0,NULL,0},

{\_nil,NULL,NULL,0,0,0,0,0,NULL,0},{\_nil,NULL,NULL,0,0,0,0,0,NULL,0},

{\_nil,NULL,NULL,0,0,0,0,0,NULL,0},{\_nil,NULL,NULL,0,0,0,0,0,NULL,0},

};

*seman.cpp*

#include "stdafx.h"

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

#include "..\spLb4\index.h"

#include "..\spLb4\automat.h"

#include "seman.h"

#include "lexcalc.h"

#include <stdlib.h>

unsigned nbBlk=0;

extern struct recrdSMA ftTbl[179];// таблиця припустимості типів для операцій

struct recrdSMA ftImp= // таблиця припустимості типів для операцій

{\_nil,\_v,0,\_v,0,\_v,0};

extern int nInCr, lnCod[]; // вектор довжин типів

extern enum datType tpLx[];// масив кодів типів констант від типів лексем

extern struct recrdTPD tpTbl[]; // таблиця модифікованих типів

enum tokType lPrv[3]={\_void,\_void,\_void};

extern struct indStrUS ndxNds[50];

// порівняння рядків

int cmpStr(struct recrdSMA \*s1, struct recrdSMA \*s2)

{unsigned n=0;

if(s1->oprtn!=s2->oprtn)return (int)(s1->oprtn)-(int)(s2->oprtn);

if(s1->oprd1-s2->oprd1)return s1->oprd1-s2->oprd1;

if(s1->ln1-s2->ln1)return s1->ln1-s2->ln1;

if(s1->oprd2-s2->oprd2)return s1->oprd2-s2->oprd2;

return s1->ln2-s2->ln2;

}

// вибірка за двійковим пошуком

struct recrdSMA\*selBin(struct recrdSMA \*kArg, struct recrdSMA\*tb, int ln)

{int i, nD=-1, nU=ln, n=(nD+nU)>>1;

while(i=cmpStr(tb+n,kArg))

{if(i>0)nU=n;else nD=n;

n=(nD+nU)>>1;

if(n==nD)return NULL;

}

return &tb[n];

}

// порівняння рядків

int cmpStr(enum tokType s1[3], enum tokType s2[3])

{unsigned n=0;

while(s1[n]==s2[n]&&n<2)n++;

return (int)(s1[n])-(int)(s2[n]);

}

// вибірка за двійковим пошуком

struct recrdTPD\*selBin(enum tokType kArg[3], struct recrdTPD\*tb, int ln)

{int i, nD=-1, nU=ln, n=(nD+nU)>>1;

while(i=cmpStr(tb[n].kTp,kArg))

{if(i>0)nU=n;else nD=n;

n=(nD+nU)>>1;

if(n==nD)return NULL;

}

return &tb[n];

}

void prDtLst(struct lxNode \* nd)

{if(nd->ndOp==\_comma)

{prDtLst(nd->prvNd);

nd->pstNd->dataType=tpLx[nd->pstNd->dataType];

convNum(nd->pstNd); //nInCr++;//, enum ltrTypeS ltrCls[256]);

}else if(nd->ndOp==\_srcn)

{nd->dataType=tpLx[nd->dataType];

convNum(nd); //nInCr++;//, enum ltrTypeS ltrCls[256]);

}

}

enum datType SmAnDcl(int tpCod, struct lxNode \* nd)

{if(nd->ndOp==\_comma)

{SmAnDcl(tpCod,nd->prvNd);

SmAnDcl(tpCod,nd->pstNd);

}else if(nd->ndOp==\_ass)

{SmAnDcl(tpCod,nd->prvNd);

if(nd->pstNd->ndOp==\_srcn)

{nd->pstNd->dataType=tpLx[nd->pstNd->dataType];

convNum(nd->pstNd); //nInCr++;//, enum ltrTypeS ltrCls[256]);

// nd->pstNd->resLength=lnCod[nd->pstNd->dataType];

}else if(nd->pstNd->ndOp==\_tdbz)

{prDtLst(nd->pstNd->pstNd);}

}else if(nd->ndOp==\_mul)//унарна \*

{if(nd->prnNd)SmAnDcl(tpCod+cdPtr,nd->pstNd);

}else if(nd->ndOp==\_ixbz)

{SmAnDcl((tpCod+cdPtr)|cdCns,nd->prvNd);

if(nd->pstNd->ndOp==\_srcn)

{nd->pstNd->dataType=\_ui;

convNum(nd->pstNd); //nInCr++;//, enum ltrTypeS ltrCls[256]);

// nd->pstNd->resLength=lnCod[nd->pstNd->dataType];

}

// SmAnDcl(\_ui,nd->pstNd);

nd->dataType=tpCod;

nd->resLength=lnCod[tpCod&0x7FF];

}else if(nd->ndOp==\_nam)

{nd->dataType=tpCod;

nd->resLength=lnCod[tpCod&0x7fff];

}

return (enum datType)tpCod;

}

enum datType SmAnlzr(struct lxNode \* nd, // вказівник на початок масиву вузлів

int nR) // номер кореневого вузла

{enum datType tPrv, tPst; int lnPrv, lnPst;

char \*name; struct recrdTPD\*pRc;

struct indStrUS \*pRtNdx;

if(nd->ndOp>=\_void&&nd->ndOp<=\_string)

{lPrv[0]=nd->ndOp;

if(nd->prvNd&&nd->prvNd->ndOp<=\_cnst)

{lPrv[1]=nd->prvNd->ndOp;

}

else lPrv[1]=\_void;

pRc=selBin(lPrv, tpTbl, 21);

if(pRc)

{tPrv=pRc->dTp;

lnPrv=pRc->ln;

if(nd->ndOp>=\_enum&&nd->ndOp<=\_union)

{nd->prvNd->dataType=nd->ndOp;

tPrv=(enum datType)(tPrv+(++nbBlk));

}

// if(nd->ndOp==\_enum)

}

SmAnDcl(tPrv,nd->pstNd);// визначити тип

}

else if(nd->ndOp==\_nam)

{pRtNdx=selBTr(nd,ndxNds);// якщо не знайдено - неописане імя

name=(char\*)pRtNdx->pKyStr->prvNd;

/\* nd->pstNd=pRtNdx->pKyStr->pstNd;\*/

nd->dataType=tPrv=(enum datType)pRtNdx->pKyStr->dataType;

nd->resLength=(int)pRtNdx->pKyStr->resLength;

}

else if(nd->ndOp==\_srcn)

{nd->dataType=tpLx[nd->dataType];

tPrv=(enum datType)nd->dataType;

convNum(nd/\*->pstNd\*/); //nInCr++;//, enum ltrTypeS ltrCls[256]);

// nd->resLength=lnCod[tPrv];

}

else{if(nd->ndOp==\_remL)return \_v;

if(nd->prvNd&&nd->ndOp!=\_nam&&nd->ndOp!=\_srcn)

{tPrv=SmAnlzr(nd->prvNd,nR);

lnPrv=nd->prvNd->resLength;}

else{ftImp.oprd1=tPrv=\_v;ftImp.ln1=0;}

if(nd->pstNd&&nd->ndOp!=\_nam&&nd->ndOp!=\_srcn)

{tPst=SmAnlzr(nd->pstNd,nR);

lnPst=nd->pstNd->resLength;}

else{ftImp.oprd2=tPst=\_v;ftImp.ln2=0;}

if(nd->ndOp==\_EOS)

{nd->dataType=\_v; nd->resLength=0;

}else if(nd->ndOp==\_brkz)

{if(nd->prvNd==0)

{nd->dataType=nd->pstNd->dataType; nd->resLength=nd->pstNd->resLength;

tPrv=(enum datType)nd->dataType;}

else

{nd->dataType=nd->prvNd->dataType;nd->resLength=nd->prvNd->resLength;

tPrv=(enum datType)nd->dataType;}

}

else

{ftImp.oprd1=tPrv; if(tPrv!=\_v)ftImp.ln1=lnPrv;else ftImp.ln1=lnPrv=0;

ftImp.oprd2=tPst; if(tPst!=\_v)ftImp.ln2=lnPst;else ftImp.ln2=lnPst=0;

ftImp.oprtn=nd->ndOp;

struct recrdSMA\*

pftImp = selBin(&ftImp, ftTbl, 179);

if(pftImp)

{nd->dataType=pftImp->res;tPrv=(enum datType)pftImp->res;

nd->resLength=pftImp->lnRes;

}}

}

return tPrv;

}

*semanT.cpp*

#include "stdafx.h"

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

#include "..\spLb7\seman.h"

#include "interpF.h"

union gnDat \*\_pd;

\_fop \*\_paddf=&\_fadd;

struct recrdSMA ftTbl[179]= // таблиця припустимості типів для операцій

{{\_if,\_ui,32,\_ui,32,\_v,0},

{\_if,\_ui,32,\_si,32,\_v,0},

{\_if,\_ui,32,\_f,32,\_v,0},

{\_if,\_ui,32,\_d,64,\_v,0},

{\_if,\_si,32,\_ui,32,\_v,0},

{\_if,\_si,32,\_si,32,\_v,0},

{\_if,\_si,32,\_f,32,\_v,0},

{\_if,\_si,32,\_d,64,\_v,0},

{\_if,\_f,32,\_ui,32,\_v,0},

{\_if,\_f,32,\_si,32,\_v,0},

{\_if,\_f,32,\_f,32,\_v,0},

{\_if,\_f,32,\_d,64,\_v,0},

{\_if,\_d,64,\_ui,32,\_v,0},

{\_if,\_d,64,\_si,32,\_v,0},

{\_if,\_d,64,\_f,32,\_v,0},

{\_if,\_d,64,\_d,64,\_v,0},

{\_else,\_v,0,\_ui,32,\_v,0},

{\_else,\_v,0,\_si,32,\_v,0},

{\_else,\_v,0,\_f,32,\_v,0},

{\_else,\_v,0,\_d,64,\_v,0},

{\_cln,\_ui,32,\_ui,32,\_ui,32},

{\_cln,\_ui,32,\_si,32,\_si,32},

{\_cln,\_ui,32,\_f,32,\_f,32},

{\_cln,\_ui,32,\_d,64,\_d,64},

{\_cln,\_si,32,\_ui,32,\_si,32},

{\_cln,\_si,32,\_si,32,\_si,32},

{\_cln,\_si,32,\_f,32,\_f,32},

{\_cln,\_si,32,\_d,64,\_d,64},

{\_cln,\_f,32,\_ui,32,\_f,32},

{\_cln,\_f,32,\_si,32,\_f,32},

{\_cln,\_f,32,\_f,32,\_f,32},

{\_cln,\_f,32,\_d,64,\_d,64},

{\_cln,\_d,64,\_ui,32,\_d,64},

{\_cln,\_d,64,\_si,32,\_d,64},

{\_cln,\_d,64,\_f,32,\_d,64},

{\_cln,\_d,64,\_d,64,\_d,64},

{\_qmrk,\_ui,32,\_ui,32,\_ui,32},

{\_qmrk,\_ui,32,\_si,32,\_si,32},

{\_qmrk,\_ui,32,\_f,32,\_f,32},

{\_qmrk,\_ui,32,\_d,64,\_d,64},

{\_qmrk,\_si,32,\_ui,32,\_ui,32},

{\_qmrk,\_si,32,\_si,32,\_si,32},

{\_qmrk,\_si,32,\_f,32,\_f,32},

{\_qmrk,\_si,32,\_d,64,\_d,64},

{\_qmrk,\_f,32,\_ui,32,\_ui,32},

{\_qmrk,\_f,32,\_si,32,\_si,32},

{\_qmrk,\_f,32,\_f,32,\_f,32},

{\_qmrk,\_f,32,\_d,64,\_d,64},

{\_qmrk,\_d,64,\_ui,32,\_ui,32},

{\_qmrk,\_d,64,\_si,32,\_si,32},

{\_qmrk,\_d,64,\_f,32,\_f,32},

{\_qmrk,\_d,64,\_d,64,\_d,64},

{\_asAdd,\_ui,32,\_ui,32,\_ui,32},

{\_asAdd,\_ui,32,\_si,32,\_ui,32},

{\_asAdd,\_si,32,\_si,32,\_si,32},

{\_asAdd,\_f,32,\_ui,32,\_f,32},

{\_asAdd,\_f,32,\_si,32,\_f,32},

{\_asAdd,\_f,32,\_f,32,\_f,32},

{\_asAdd,\_f,32,\_d,64,\_f,32},

{\_asAdd,\_d,64,\_ui,32,\_d,32},

{\_asAdd,\_d,64,\_si,32,\_d,64},

{\_asAdd,\_d,64,\_f,32,\_d,64},

{\_asAdd,\_d,64,\_d,64,\_d,64},

{\_ass,\_ui,32,\_ui,32,\_ui,32},

{\_ass,\_ui,32,\_si,32,\_ui,32},

{\_ass,\_si,32,\_si,32,\_si,32},

{\_ass,\_f,32,\_ui,32,\_f,32},

{\_ass,\_f,32,\_si,32,\_f,32},

{\_ass,\_f,32,\_f,32,\_f,32},

{\_ass,\_f,32,\_d,64,\_f,32},

{\_ass,\_d,64,\_ui,32,\_d,32},

{\_ass,\_d,64,\_si,32,\_d,64},

{\_ass,\_d,64,\_f,32,\_d,64},

{\_ass,\_d,64,\_d,64,\_d,64},

{\_dcr,\_v,0,\_ui,32,\_ui,32},

{\_dcr,\_v,0,\_si,32,\_si,32},

{\_dcr,\_ui,32,\_v,0,\_ui,32},

{\_dcr,\_si,32,\_v,0,\_si,32},

{\_inr,\_v,0,\_ui,32,\_ui,32},

{\_inr,\_v,0,\_si,32,\_si,32},

{\_inr,\_ui,32,\_v,0,\_ui,32},

{\_inr,\_si,32,\_v,0,\_si,32},

{\_lt,\_f,32,\_f,32,\_ui,32},

{\_le,\_f,32,\_f,32,\_ui,32},

{\_eq,\_f,32,\_f,32,\_ui,32},

{\_ne,\_ui,32,\_ui,32,\_ui,32},

{\_ne,\_f,32,\_f,32,\_ui,32},

{\_ge,\_f,32,\_f,32,\_ui,32},

{\_gt,\_f,32,\_f,32,\_ui,32},

{\_add,\_ui,32,\_ui,32,\_ui,32},

{\_add,\_ui,32,\_si,32,\_si,32},

{\_add,\_ui,32,\_f,32,\_f,32},

{\_add,\_ui,32,\_d,32,\_d,64},

{\_add,\_si,32,\_ui,32,\_si,32},

{\_add,\_si,32,\_si,32,\_si,32},

{\_add,\_si,32,\_f,32,\_f,32},

{\_add,\_si,32,\_d,32,\_d,64},

{\_add,\_f,32,\_ui,32,\_f,32},

{\_add,\_f,32,\_si,32,\_f,32},

{\_add,\_f,32,\_f,32,\_f,32, \_paddf},

{\_add,\_f,32,\_d,32,\_d,64},

{\_add,\_d,32,\_ui,32,\_d,64},

{\_add,\_d,32,\_si,32,\_d,64},

{\_add,\_d,32,\_f,32,\_d,64},

{\_add,\_d,32,\_d,32,\_d,64},

{\_sub,\_ui,32,\_ui,32,\_ui,32},

{\_sub,\_ui,32,\_si,32,\_si,32},

{\_sub,\_ui,32,\_f,32,\_f,32},

{\_sub,\_ui,32,\_d,32,\_d,64},

{\_sub,\_si,32,\_ui,32,\_si,32},

{\_sub,\_si,32,\_si,32,\_si,32},

{\_sub,\_si,32,\_f,32,\_f,32},

{\_sub,\_si,32,\_d,32,\_d,64},

{\_sub,\_f,32,\_ui,32,\_f,32},

{\_sub,\_f,32,\_si,32,\_f,32},

{\_sub,\_f,32,\_f,32,\_f,32},

{\_sub,\_f,32,\_d,32,\_d,64},

{\_sub,\_d,32,\_ui,32,\_d,64},

{\_sub,\_d,32,\_si,32,\_d,64},

{\_sub,\_d,32,\_f,32,\_d,64},

{\_sub,\_d,32,\_d,32,\_d,64},

{\_mul,\_ui,32,\_ui,32,\_ui,32},

{\_mul,\_ui,32,\_si,32,\_si,32},

{\_mul,\_ui,32,\_f,32,\_f,32},

{\_mul,\_ui,32,\_d,32,\_d,64},

{\_mul,\_si,32,\_ui,32,\_si,32},

{\_mul,\_si,32,\_si,32,\_si,32},

{\_mul,\_si,32,\_f,32,\_f,32},

{\_mul,\_si,32,\_d,32,\_d,64},

{\_mul,\_f,32,\_ui,32,\_f,32},

{\_mul,\_f,32,\_si,32,\_f,32},

{\_mul,\_f,32,\_f,32,\_f,32},

{\_mul,\_f,32,\_d,32,\_d,64},

{\_mul,\_d,32,\_ui,32,\_d,64},

{\_mul,\_d,32,\_si,32,\_d,64},

{\_mul,\_d,32,\_f,32,\_d,64},

{\_mul,\_d,32,\_d,32,\_d,64},

{\_div,\_ui,32,\_ui,32,\_ui,32},

{\_div,\_ui,32,\_si,32,\_si,32},

{\_div,\_ui,32,\_f,32,\_f,32},

{\_div,\_ui,32,\_d,32,\_d,64},

{\_div,\_si,32,\_ui,32,\_si,32},

{\_div,\_si,32,\_si,32,\_si,32},

{\_div,\_si,32,\_f,32,\_f,32},

{\_div,\_si,32,\_d,32,\_d,64},

{\_div,\_f,32,\_ui,32,\_f,32},

{\_div,\_f,32,\_si,32,\_f,32},

{\_div,\_f,32,\_f,32,\_f,32},

{\_div,\_f,32,\_d,32,\_d,64},

{\_div,\_d,32,\_ui,32,\_d,64},

{\_div,\_d,32,\_si,32,\_d,64},

{\_div,\_d,32,\_f,32,\_d,64},

{\_div,\_d,32,\_d,32,\_d,64},

{\_mod,\_ui,32,\_ui,32,\_ui,32},

{\_mod,\_ui,32,\_si,32,\_si,32},

{\_mod,\_si,32,\_ui,32,\_si,32},

{\_mod,\_si,32,\_si,32,\_si,32},

{\_orB,\_ui,32,\_ui,32,\_ui,32},

{\_orB,\_ui,32,\_si,32,\_si,32},

{\_orB,\_si,32,\_ui,32,\_si,32},

{\_orB,\_si,32,\_si,32,\_si,32},

{\_andB,\_ui,32,\_ui,32,\_ui,32},

{\_andB,\_ui,32,\_si,32,\_si,32},

{\_andB,\_si,32,\_ui,32,\_si,32},

{\_andB,\_si,32,\_si,32,\_si,32},

{\_xorB,\_ui,32,\_ui,32,\_ui,32},

{\_xorB,\_ui,32,\_si,32,\_si,32},

{\_xorB,\_si,32,\_ui,32,\_si,32},

{\_xorB,\_si,32,\_si,32,\_si,32},

{\_or,\_ui,32,\_ui,32,\_ui,32},

{\_and,\_ui,32,\_ui,32,\_ui,32},

{\_ixbz,\_ui+cdPtr|cdCns,32,\_ui,32,\_ui,32},

{\_ixbz,\_ui+cdPtr|cdCns,32,\_si,32,\_ui,32},

{\_ixbz,\_si+cdPtr|cdCns,32,\_ui,32,\_si,32},

{\_ixbz,\_si+cdPtr|cdCns,32,\_si,32,\_si,32},

{\_ixbz,\_f+cdPtr|cdCns,32,\_ui,32,\_f,32},

{\_ixbz,\_f+cdPtr|cdCns,32,\_si,32,\_f,32},

{\_ixbz,\_d+cdPtr|cdCns,64,\_ui,32,\_d,32},

{\_ixbz,\_d+cdPtr|cdCns,64,\_si,32,\_d,32},

};

int lnCod[]= // вектор довжин типів

{0, 0, 0, 0,

8,16,32,64,

8,16,32,64,//

32,64,80,48,

\_lbl,//

\_enm,\_str,\_unn,

};

enum datType tpLx[]= // масив кодів типів констант від типів лексем

{\_v, //0 Eu - Некласифікований об'єкт

\_v, //1 S0 - Роздільник

\_v, //2 S1g - Знак числової константи

\_ui, //3 S1c - Ціле число

\_d, //4 S2c - Число з точкою

\_v, //5 S1e - Літера "e" або "E"

\_v, //6 S1q - Знак "-" або "+"

\_f, //7 S1p - Десяткові цифри порядку

\_v, //8 S1n - Елементи імені

\_v, //9 S1s - Літери рядка або символьної константи

\_v, //10 S1t - Елементи констант, які перетворюються

\_strn, //11 S2s - Ознака закінчення константи

\_v, //12 S2 - Початковий елемент групового роздільника

\_v, //13 S3 - Наступний елемент групового роздільника

\_ui, //14?S3c - Ціле число з недесятковою основою

\_v, //15?S0p - Ознака типу константи

\_v, //16 Soc- Вісімковий код

\_v, //17 Scr- Коментар-рядок

\_v, //18 Scl- Обмежений коментар

\_v, //19 Ec - Неправильна константа

\_v, //20 Ep - Неправильна константа з точкою

\_v, // Eq - Неправильна константа з порядком

\_v, // En - Неправильне ім'я

\_v // Eo - Неприпустиме сполучення операцій

};

struct recrdTMD tpLxMd[]=

// масив кодів та ознак ключових слів типів

{{\_v, 0, 0}, //0 \_void

{\_v, 0, 0}, //1 \_extern

{\_v, 0, 0}, //2 \_var

{\_v,cdCns,0}, //3 \_const

{\_enm, 0,32}, //4 \_enum

{\_str, 0, 0}, //5 \_struct/\*\_record\*/

{\_unn, 0, 0}, //6 \_union

{\_v,cdReg,0}, //7 \_register

{\_ui,0,32}, //8 \_unsigned

{\_si,0,32}, //9 \_signed

{\_si,0, 8}, //10 \_char

{\_si,0,16}, //11 \_short

{\_si,0,32}, //12 \_int

{\_si,0,32}, //13 \_long

{\_si,0,64}, //14 \_sint64

{\_ui,0,64}, //15 \_uint64

{\_f, 0,32}, //16 \_float

{\_d, 0,64}, //17 \_double

};

struct recrdTPD tpTbl[]= // таблиця модифікованих типів

{{{\_void,\_void,\_void},\_v,0},

{{\_enum,\_void,\_void},\_enm,32},

{{\_struct,\_void,\_void},\_str,0},

{{\_union,\_void,\_void},\_unn,0},

{{\_unsigned,\_void,\_void},\_ui,32},

{{\_signed,\_void,\_void},\_si,32},

{{\_char,\_unsigned,\_void},\_uc,8},

{{\_char,\_signed,\_void},\_sc,8},//4

{{\_char,\_void,\_void},\_sc,8},

{{\_short,\_void,\_void},\_si,16},

{{\_short,\_unsigned,\_void},\_ui,16},

{{\_short,\_signed,\_void},\_si,16},

{{\_int,\_void,\_void},\_si,32},//9

{{\_int,\_unsigned,\_void},\_ui,32},

{{\_int,\_signed,\_void},\_si,32},

{{\_int,\_long,\_void},\_si,32},

{{\_long,\_void,\_void},\_si,32},

{{\_float,\_void,\_void},\_f,32},//14

{{\_double,\_void,\_void},\_d,64},

{{\_double,\_long,\_void},\_ld,80},

{{\_class,\_void,\_void},\_cls,0},

};

*syntax.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;

}

*tables.h*

struct recrdKWD // структура рядка таблиці ключових слыв ы роздыльникыв

{char\* key;// примірник структури ключа

enum tokType func;// примірник функціональної частини

int versn; // номер первинноъ версыъ застосування

};

//---!!! функції етапу виконання!!!---

// порівняння рядків

int cmpStr(unsigned char\* s1, unsigned char\* s2);

// вибірка за двійковим пошуком

struct recrdKWD\*selBin(char\* kArg, struct recrdKWD\*tb, int ln);

//---!!! функції етапів проектування і настроювання!!!---

// сортування для двійкового пошуку

struct recrdKWD\*srtBin(struct recrdKWD\*tb, int ln);

// включення за двійковим пошуком

struct recrdKWD\*insBin(struct recrd\*pElm,

struct recrdKWD\*tb, int\*pQnElm);

// вилучення за двійковим пошуком

struct recrdKWD\*delBin(struct recrdKWD\*pElm,

struct recrdKWD\*tb, int\*pQnElm);

// корекція за двійковим пошуком

struct recrdKWD\*updBin(struct keyStr, struct recrdKWD\*pElm,

struct recrdKWD\*tb, int\*pQnElm);

//------------------------------------------------------

*tables.cpp*

#include "stdafx.h"

#include "tables.h"

#include <string.h>

// порівняння рядків

int cmpStr(unsigned char\* s1, unsigned char\* s2)

{unsigned n=0;

while(s1[n]==s2[n]&&s1[n]!=0)n++;

return s1[n]-s2[n];

}

// порівняння за відношенням порядку

int cmpKys(char \*k0, char \*kArg)

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

(unsigned char\*)kArg);

//if(i)

return i;

}

//------------------------------------------------------

// вибірка за двійковим пошуком

struct recrdKWD\*selBin(char\* kArg, struct recrdKWD\*tb, int ln)

{int i, nD=-1, nU=ln, n=(nD+nU)>>1;

while(i=cmpKys(tb[n].key,kArg))

{if(i>0)nU=n;else nD=n;

n=(nD+nU)>>1;

if(n==nD)return NULL;

}

return &tb[n];

}

// сортування для двійкового пошуку

struct recrdKWD\*srtBin(struct recrdKWD\*tb, int ln)

{int n=0, n1;

struct recrdKWD el;

for(;n<ln;n++)for(n1=n+1;n1<ln;n1++)

if(cmpKys(tb[n].key,tb[n1].key)>0)

{el=tb[n];tb[n]=tb[n1];tb[n1]=el;}

return tb;

}

*visgrp.cpp*

#include "stdafx.h"

#include "visgrp.h"

#include "token.h"

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

\*cprC[], \*cprP[], \*cprV[], \*\*oprtr, \*\*cpr,

modeP,//=1, // тип роздільника операторних дужок для Паскаля

modeC,//=0, // тип роздільника операторних дужок для С

modeL;//=modeC;

char mode=0; // mode=0 режим генерації роздільника

// mode=1 режим генерації слова

// mode=2 режим генерації одного вкладеного оператора в С

char fpr[]= {0,0x4f,0x4f,0x4f,0x46,0x11,6,0x12,

0x13,0x4e,0x12,0x1, 0x4e,0x4e,0x4e,0x4e,

0x4e,0x4e,0x13,0x4e, 0x4e, 0x9, 0x9, 0x9,

0x4e,0x4e,0x4e,0x4e,

0x4e,0x4e,0x4e,0x4e, 0x4e,0x4e,0x4e,0x4e,

0x4e,0x4e,0x4e,0x4e, 0x4e,0x4e,0x4e,0x4e,

0x4e,0x4e,0x4e,0x4e, 0x4e,0x4e,0x4e,0x4e,

0x4e,0x4e,0x4e,0x4e, 0x4e,0x4e,0x4e,0x4e, 0x4e,0x4e,

0x4f,0x4f,0x4f,0x4f, //Verilog+4

0x4f,0x4f,0x4f,0x4f, //Verilog+8

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+15

0x4f,0x4f,0x4f,0x4f, //Verilog+19

0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+24

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+31

0x4f,0x4f,0x4f,0x4f, //Verilog+35

0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+40

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+46

0x4e,0x4e, 0x4e,0x4e,0x4e,0x4e,

0x4f,0x4f,0x4f, //Verilog|SQL+3

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //P++9

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //C++15

0x4f,0x4f,0x4f,0x4f,0x4f/\*raise\*/, //C++20

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+26

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+32

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+38

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+44

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+50

0x4f,0x4f,0x4f,0x4f, //Verilog+54

0x4f,0x4f,0x4f,0x4f, //Verilog+58

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+64

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+70

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+76

0x42, 0x2, 0x42,2,0x43,3, 0x44,4,0x45,5,0x46,6,

0x1, 0x1, 0x1, 0x2, 0x2, 0x43,

0x10,0x10,0x10,0x10, 0x10,0x10,0x10,0x10,

0x10,0x10,0x10, 0x3C,0x3C,

0x3C,0x3C, 0x3E,0x3E,0x3E,0x3E,

0x1A,0x1A,0x1C,0x1C,0x1A,0x1A,

0x20, 0x20, 0x30, 0x30,

0x3E,0x3E, 0x3A,0x1E,0x1E,0x1C, 0x1C,

0x3C,0x3C,0x3C,0x3C,

0x11, 0x1C, 0x1C,0x15, // PV+4 .. \_lmts,\_eqar,\_astar,\_trasand,

0x3C,0x3C,0x3C,0x3C,0x3C,0x3C,0x3C, //V+11 \_orR,\_andR,\_xorR,\_norR,\_nandR,\_nxorR,\_xornR

0x3C,0x3C,0x3C, //V+14 \_delay,\_event,\_events,

0x16,0x18,0x17,0x17, 0x3C,

0x1,0x1, 0x30,0x16,0x18,0x17, // %(mod) |(or) &(and) ^(xor)

0x1E,0x1E,0x13,0x15, //<<(shl) >>(shr) ||(or) &&(and)

0x4e,0x4e,0x30,0x12,

0x4e,0x4e,0x15,

0x42,0x42,0x43,0x44,0x45,0x46

},

gpr[]= {0, 0x4f, 0x4f, 0x4f, 6,0x12,6,0x12,

0x13,0x4e,0x12,0x1, 0x12,0x1,0x4e,0x4e,

0x11,0x1,0x1,0x11, 0x11, 0x9, 0x9, 0x9,

0x11,0x1,0x1,0x11,

0x01,0x01,0x01,0x01, 0x01,0x01,0x01,0x01,

0x01,0x01,0x01,0x01, 0x01,0x01,0x01,0x01,

0x01,0x01,0x01,0x01, 0x01,0x01,0x01,0x01,

0x01,0x01,0x01,0x01, 0x01,0x01,0x01,0x01, 0x01,0x01,

0x4f,0x4f,0x4f,0x4f, //Verilog+4

0x4f,0x4f,0x4f,0x4f, //Verilog+8

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+15

0x4f,0x4f,0x4f,0x4f, //Verilog+19

0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+24

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+31

0x4f,0x4f,0x4f,0x4f, //Verilog+35

0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+40

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+46

0x01,0x01, 0x01,0x01,0x01,0x01,

0x4f,0x4f,0x4f, //Verilog|SQL+3

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //P++9

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //C++15

0x4f,0x4f,0x4f,0x4f,0x4f/\*raise\*/, //C++20

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+26

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+32

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+38

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+44

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+50

0x4f,0x4f,0x4f,0x4f, //Verilog+54

0x4f,0x4f,0x4f,0x4f, //Verilog+58

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+64

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+70

0x4f,0x4f,0x4f,0x4f,0x4f,0x4f, //Verilog+76

0x2, 0x2, 2,2,0x3,3, 0x4,4,0x5,5,0x6,6,

0x1, 0x1, 0x1, 0x2, 0x2, 0x43,

0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10,

0x10,0x10,0x10, 0x3C,0x3C,

0x3C,0x3C, 0x3E,0x3E,0x3E,0x3E,

0x1A,0x1A,0x1C,0x1C,0x1A,0x1A,

0x20, 0x20, 0x30, 0x30,

0x3E,0x3E, 0x34,0x1E,0x1E,0x1C, 0x1C,

0x3C,0x3C,0x3C,0x3C,

0x11, 0x1C, 0x1C,0x15, // PV+4 .. \_lmts,\_eqar,\_astar,\_trasand,

0x3C,0x3C,0x3C,0x3C,0x3C,0x3C,0x3C, //V+11 \_orR,\_andR,\_xorR,\_norR,\_nandR,\_nxorR,\_xornR

0x3C,0x3C,0x3C, //V+14 \_delay,\_event,\_events,

0x16,0x18,0x17,0x17, 0x3C,

0x1,0x1, 0x30,0x16,0x18,0x17, // %(mod) |(or) &(and) ^(xor)

0x1E,0x1E,0x13,0x15, //<<(shl) >>(shr) ||(or) &&(and)

0x4e,0x4e,0x30,0x12,

0x4e,0x4e,0x15,

0x2,0x2,0x3,0x4,0x5,0x6

};

void prOpBr(struct lxNode\*rt)//відкриті дужки

{if(rt->ndOp>\_inr||rt->ndOp<\_if

||(rt->ndOp>=\_to&&rt->ndOp<=\_step))

{printf("%s",oprtr[\_brkt]); mode=0;}

else if(mode!=0){printf(" "); mode=0;}

/\* else\*/ if(!(rt&&rt->ndOp!=\_EOS))

{printf("%s",oprtr[\_opbr]); mode=modeL;}

}

void prClBr(struct lxNode\*rt)//закриті дужки

{if(rt->ndOp>\_inr||rt->ndOp<\_if

||(rt->ndOp>=\_to&&rt->ndOp<=\_step))

{printf("%s",oprtr[\_bckt]); mode=0;}

else if(rt&&rt->ndOp!=\_EOS)

{;/\*if(rt->ndOp<\_opbr||(rt->ndOp>=\_asOr&&rt->ndOp>=\_asOr))

{printf("%s",oprtr[\_EOS]); mode=0;}\*/}

else printf("%s%s",oprtr[\_EOS],oprtr[\_ocbr]);

}

// виведення піддерва лексем

void prLxTxt(struct lxNode\*rt) //корінь піддерева

{static int mdCnt=0; // лічильники входжень

struct lxNode\* rt0; // робочий вказівник

char n=0, c, bC=0, opCnt=0;

if(rt->ndOp<=\_cnst)

{if(rt->ndOp!=\_nil)// обробка термінального операнда

{if(mode==1&&rt->ndOp<begOprtr-8)printf(" ");

printf("%s",rt->prvNd);

mode=1;

} } // вихід з рекурсії

else if(cpr[rt->ndOp&(~\_lVlu)][0]!=0)

{while((c=cpr[rt->ndOp&(~\_lVlu)][n])!=0)// цикл прогляду шаблона

{n++; // просування по шаблону

switch(c) // аналіз літери шаблона

{case 7:

if(mdCnt==0)mdCnt=-1;

else while(c!=-1)c=cpr[rt->ndOp&(~\_lVlu)][n++];// пропуск шаблона

c=0; break;

case 6: opCnt--;// повернення до першого аргумента

case -1: break;

case -2: mdCnt=0; break;

case 1: /\*case 'f':\*/ if(mode!=0&&rt->ndOp>=\_if

&&rt->ndOp<begOprtr-8)printf(" ");

printf("%s",oprtr[rt->ndOp&(~\_lVlu)]);

if(rt->ndOp>=begOprtr-8)mode=0;

else mode=1; break;

case 'x': case 'y': case 5: case 4:

if(opCnt)rt0=rt->pstNd; // вибір аргумента

else rt0=rt->prvNd; // перевірка потреби обрамлення

if(rt0)

{if(c=='y')bC=fpr[rt->ndOp]>fpr[rt0->ndOp]&&rt0

&&rt0->ndOp!=\_opbz;

else if(c==5){if(rt0&&rt0->ndOp!=\_brkz)bC=1;}

else if(c==4)bC=0;

else bC=fpr[rt->ndOp&(~\_lVlu)]>=fpr[rt0->ndOp&(~\_lVlu)]&&rt0;

if(bC)prOpBr(rt0); // обрамлення дужками

prLxTxt(rt0); // рекурсивний виклик відтворення

if(bC)prClBr(rt0); // обрамлення дужками

}

case 'z':opCnt++; // підготовка наступного аргумента

break;

default :printf("%c",c);

}

// mdC=mdCnt;

if(c==-1&&mdCnt!=0)break;

}}

else{rt0=rt->prvNd;

if(rt0!=NULL&&rt->ndOp!=\_remL&&rt->ndOp!=\_rem)

{bC=fpr[rt->ndOp&(~\_lVlu)]>fpr[rt0->ndOp&(~\_lVlu)];

if(bC)prOpBr(rt0);

prLxTxt(rt0);

if(bC)prClBr(rt0);

}

if(mode==1&&(rt->ndOp&(~\_lVlu))<begOprtr-8)printf(" ");

// if(rt->ndOp!=\_EOS||rt->prvNd->ndOp!=\_EOS||rt->pstNd==NULL)

printf("%s",oprtr[rt->ndOp&(~\_lVlu)]);

if((rt->ndOp&(~\_lVlu))>begOprtr-8)mode=0; else mode=1;

if(rt->ndOp==\_remL||rt->ndOp==\_rem)

printf("%s\n",rt->prvNd);

/\* else\*/{rt0=rt->pstNd;

if(rt0!=NULL)

{bC=gpr[rt->ndOp&(~\_lVlu)]>=gpr[rt0->ndOp&(~\_lVlu)]

&&(rt->ndOp&(~\_lVlu))<\_frkz;//&&rt0->ndOp<\_frkz;//0

if(bC)prOpBr(rt0);

prLxTxt(rt0);

if(bC)prClBr(rt0);

}}

if((rt->ndOp&(~\_lVlu))>=\_opbr&&(rt->ndOp&(~\_lVlu))<=\_bckt)

{printf("%s",oprtr[(rt->ndOp&(~\_lVlu))+1]); mode=0;}

if((rt->ndOp&(~\_lVlu))>=\_opbz&&(rt->ndOp&(~\_lVlu))<=\_tdbz)

{printf("%s",oprtr[((rt->ndOp&(~\_lVlu))-\_frkz)\*2+\_fork+1]); mode=0;}

} return;

}

void prOpBr(struct lxNode\*rt,FILE\*f)//відкриті дужки

{if(rt->ndOp>\_inr||rt->ndOp<\_if

||(rt->ndOp>=\_to&&rt->ndOp<=\_step))

{fprintf(f,"%s",oprtr[\_brkt]); mode=0;}

else if(mode!=0){fprintf(f," "); mode=0;}

/\* else\*/ if(!(rt&&rt->ndOp!=\_EOS))

{fprintf(f,"%s",oprtr[\_opbr]); mode=modeL;}

}

void prClBr(struct lxNode\*rt,FILE\*f)//закриті дужки

{if(rt->ndOp>\_inr||rt->ndOp<\_if

||(rt->ndOp>=\_to&&rt->ndOp<=\_step))

{fprintf(f,"%s",oprtr[\_bckt]); mode=0;}

else if(rt&&rt->ndOp!=\_EOS)

{;/\*if(rt->ndOp<\_opbr||(rt->ndOp>=\_asOr&&rt->ndOp>=\_asOr))

{printf("%s",oprtr[\_EOS]); mode=0;}\*/}

else fprintf(f,"%s%s",oprtr[\_EOS],oprtr[\_ocbr]);

}

void prLxTxt(struct lxNode\*rt,FILE\*f) //корінь піддерева

{static int mdCnt=0; // лічильники входжень

struct lxNode\* rt0; // робочий вказівник

char n=0, c, bC=0, opCnt=0;

if(rt->ndOp<=\_cnst)

{if(rt->ndOp!=\_nil)// обробка термінального операнда

{if(mode==1&&rt->ndOp<begOprtr-8)fprintf(f," ");

fprintf(f,"%s",rt->prvNd);

mode=1;

} } // вихід з рекурсіъ

else if(cpr[rt->ndOp][0]!=0)

{while((c=cpr[rt->ndOp][n])!=0)// цикл прогляду шаблона

{n++; // просування по шаблону

switch(c) // аналіз літери шаблона

{case 7:

if(mdCnt==0)mdCnt=-1;

else while(c!=-1)c=cpr[rt->ndOp][n++];// пропуск шаблона

c=0; break;

case 6: opCnt--;// повернення до першого аргумента

case -1: break;

case -2: mdCnt=0; break;

case 1: /\*case 'f':\*/ if(mode!=0&&rt->ndOp>=\_if

&&rt->ndOp<begOprtr-8)fprintf(f," ");

fprintf(f,"%s",oprtr[rt->ndOp]);

if(rt->ndOp>=begOprtr-8)mode=0;

else mode=1; break;

case 'x': case 'y': case 5: case 4:

if(opCnt)rt0=rt->pstNd; // вибір аргумента

else rt0=rt->prvNd; // перевірка потреби обрамлення

if(rt0)

{if(c=='y')bC=fpr[rt->ndOp]>fpr[rt0->ndOp]&&rt0

&&rt0->ndOp!=\_opbz;

else if(c==5){if(rt0&&rt0->ndOp!=\_brkz)bC=1;}

else if(c==4)bC=0;

else bC=fpr[rt->ndOp]>=fpr[rt0->ndOp]&&rt0;

if(bC)prOpBr(rt0,f); // обрамлення дужками

prLxTxt(rt0,f); // рекурсивний виклик відтворення

if(bC)prClBr(rt0,f); // обрамлення дужками

}

case 'z':opCnt++; // підготовка наступного аргумента

break;

default :fprintf(f,"%c",c);

}

// mdC=mdCnt;

if(c==-1&&mdCnt!=0)break;

}}

else{rt0=rt->prvNd;

if(rt0!=NULL&&rt->ndOp!=\_remL&&rt->ndOp!=\_rem)

{bC=fpr[rt->ndOp]>fpr[rt0->ndOp];

if(bC)prOpBr(rt0,f);

prLxTxt(rt0,f);

if(bC)prClBr(rt0,f);

}

if(mode==1&&rt->ndOp<begOprtr-8)fprintf(f," ");

// if(rt->ndOp!=\_EOS||rt->prvNd->ndOp!=\_EOS||rt->pstNd==NULL)

fprintf(f,"%s",oprtr[rt->ndOp]);

if(rt->ndOp>begOprtr-8)mode=0; else mode=1;

if(rt->ndOp==\_remL||rt->ndOp==\_rem)

fprintf(f,"%s\n",rt->prvNd);

/\* else\*/{rt0=rt->pstNd;

if(rt0!=NULL)

{bC=gpr[rt->ndOp]>=gpr[rt0->ndOp]

&&rt->ndOp<\_frkz&&rt0->ndOp<\_frkz;//0

if(bC)prOpBr(rt0,f);

prLxTxt(rt0,f);

if(bC)prClBr(rt0,f);

}}

if(rt->ndOp>=\_opbr&&rt->ndOp<=\_bckt)

{fprintf(f,"%s",oprtr[rt->ndOp+1]); mode=0;}

if(rt->ndOp>=\_opbz&&rt->ndOp<=\_tdbz)

{fprintf(f,"%s",oprtr[(rt->ndOp-\_frkz)\*2+\_fork+1]); mode=0;}

} return;

}

// виведення масиву лексем

void prLaTxt(struct lxNode\*ar, unsigned n) //початок масиву лексем

{unsigned i=0;

do

{if(ar[i].ndOp<=\_cnst||ar[i].ndOp==\_remL)

{if(ar[i].ndOp!=\_nil)// обробка термінального операнда

{if(ar[i].ndOp==\_remL)printf("%s",oprtr[ar[i].ndOp]);

if(mode==1&&ar[i].ndOp<begOprtr-8)printf(" ");

if(ar[i].ndOp==\_remL)printf("%s\n",ar[i].prvNd);

else printf("%s",ar[i].prvNd);

mode=1;

} } //

else

{if(mode==1&&ar[i].ndOp<begOprtr-8)printf(" ");

printf("%s",oprtr[ar[i].ndOp]);

if(ar[i].ndOp>begOprtr-8)mode=0; else mode=1;

}

}while(++i<n);

}

*token.h*

#define begOprtr 0x50+46+78 // зміщення початку виконавчих операторів

#define \_lVlu 0x1000 // lValue

enum tokType

{\_nil, \_nam, //0 зовнішнє подання

\_srcn, \_cnst, //2 вхідне і внутрішнє кодування константи

\_if,\_then,\_else,\_elseif, //4 if then else elseif

\_case, \_switch, \_default, \_endcase,//8 case switch defualt endcase

\_break, \_return, \_whileP, \_whileN, //12 break return while do

\_continue, \_repeat, \_untilN, \_endloop, //16 continue repeat until

\_for, \_to, \_downto, \_step,// for to downto step

\_untilP, \_loop, \_with, \_endif,

\_void,\_extern,\_var,\_const,\_enum,\_struct/\*\_record\*/,\_union,\_register,//

\_unsigned,\_signed,\_char,\_short,\_int,\_long,\_sint64,\_uint64,//

\_float,\_double, \_label,\_auto,\_static,\_volatile,\_typedef,\_sizeof,//

\_real,\_array,\_set,\_file,\_object, \_string,\_goto,

\_program,\_function,\_procedure /\*task V\*/,

\_macromodule,\_primitive,\_specify,\_table, //Verilog

\_generate,\_config,\_liblist,\_library, //Verilog

\_incdir,\_include,\_design,\_defaultS,\_instance,\_cell,\_use, //Verilog

\_automatic,\_endmodule,\_endfunction,\_endtask, //Verilog

\_endprimitive,\_endspecify,\_endtable,\_endgenerate,\_endconfig, //Verilog

\_endcaseV,\_casex,\_casez,\_wait,\_forever,\_disable,\_ifnone, //Verilog

\_pulsestyle\_onevent,\_pulsestyle\_ondetect,\_showcanceled,\_noshowcanceled, //Verilog

\_vectored,\_scalared,\_small,\_medium,\_large, //Verilog

\_genvar,\_parameter,\_localparam,\_defparam,\_specparam,\_PATHPULSE$, //Verilog

\_inlineF,\_forward,\_interrupt,\_exportF,\_extrn,\_asmb,

\_input,\_output,\_inout, //Verilog|SQL+3

\_objectP,\_constructor,\_desctructor,\_property,\_resP,\_abstract, //P++9

\_class,\_public,\_private,\_protected,\_virtual,\_friend, //C++16

\_new,\_delete,\_tryC,\_catch,\_throw/\*raise\*/, //C++20

\_initial,\_always,\_assign,\_deassign,\_force,\_release, //Verilog+26

\_reg,\_time,\_realtime,\_event,\_buf,\_not, //Verilog+32

\_andG,\_orG,\_xorG,\_nandG,\_norG,\_xnorG, //Verilog+38

\_tran,\_tranif0,\_tranif1,\_rtran,\_rtranif0,\_rtranif1, //Verilog+44

\_tri,\_trior,\_triand,\_trireg,\_tri0,\_tri1,//Verilog+50

\_wire,\_wand,\_wor,\_wres, //Verilog+54

\_supply0,\_supply1,\_highz0,\_highz1, //Verilog+58

\_strong0,\_strong1,\_pull0,\_pull1,\_weak0,\_weak1, //Verilog+64

\_pulldown,\_pullup,\_bufif0,\_bufif1,\_notif0,\_notif1, //Verilog+70

\_cmos,\_rcmos,\_nmos,\_pmos,\_rnmos,\_rpmos, //Verilog+76

\_fork, \_join, // відкриті і закриті дужки паралельних операторів 2

\_opbr, \_ocbr, // відкриті і закриті дужки операторів 2

\_ctbr, \_fcbr, // відкриті і закриті дужки конкатенацій 3

\_ixbr, \_scbr, // відкриті і закриті дужки індексу 4

\_brkt, \_bckt, // відкриті і закриті дужки порядку і функцій 5

\_tdbr, \_tcbr, // відкриті і закриті дужки даних 6

\_eosP, eosS, // паралельні та послідовні

\_EOS=begOprtr, \_comma, \_cln, \_qmrk,// ; , : ?

\_asOr, \_asAnd, \_asXor, \_asAdd, //|= =& =^ =+

\_asSub, \_asMul, \_asDiv, \_asMod, // -= \*= /= %=

\_asShr,\_asShl, \_ass, \_dcr, \_inr, // <<= >>= = -- ++

\_dcrN,\_inrN,\_mcrs,\_dbcln,\_eoCm,\_EOF, //-- ++ // # :: \*/

\_lt,\_le, \_eq, \_ne, \_ge,\_gt, // < <= == != >= >

\_add, \_sub, \_mul, \_div, \_fldDt, \_fldPt,// + - \* / . ->

\_pwr, \_shLfa, \_shRga, \_eqB, \_neB, // \*\* <<< >>> === !==

\_addU,\_subU,\_refU, \_ptrU, // + - \* & унарні

\_lmts,\_eqar,\_astar,\_trasand, // PV+4 .. => \*> &&&

\_orR,\_andR,\_xorR,\_norR,\_nandR,\_nxorR,\_xornR, //V+11 & | ^ ~| ~& ~^

\_delay,\_eventV,\_events, //V+14 # @ @\*

\_norB,\_nandB,\_nxorB,\_xornB,\_addr, //~| ~& ~^ ^~ \_ptr,

\_rem,\_remL, //

\_mod, \_orB, \_andB, \_xorB, // %(mod) |(or) &(and) ^(xor)

\_shLft,\_shRgt, \_or, \_and, //<<(shl) >>(shr) ||(or) &&(and)

\_xmrk,\_invB,\_divI,\_in, //\_not, \_notB, /(div)

\_posedge,\_negedge,\_orE, //Verilog+3

\_frkz, // відкриті і закриті дужки паралельних операторів 2

\_opbz, // відкриті і закриті дужки операторів 2

\_ctbz, // відкриті і закриті дужки конкатенацій 3

\_ixbz, // відкриті і закриті дужки індексу 4

\_brkz, // відкриті і закриті дужки порядку і функцій 5

\_tdbz // відкриті і закриті дужки даних 6

// \_pnil

};

struct lxNode//вузол дерева, САГ або УСГ

{enum tokType ndOp; //код типу лексеми

struct lxNode\* prvNd;// зв'язок з попередником

struct lxNode\* pstNd;// зв'язок з наступником

int dataType; // код типу даних, які повертаються

unsigned resLength; //довжина результату

int x, y, f;//координати розміщення у вхідному файлі

int prnNd;//struct lxNode\* prnNd;//зв'язок з батьківським вузлом

unsigned stkLength;//довжина стека обробки семантики або номер модуля

};

// порівняння за відношенням порядку

int cmpKys(struct lxNode\*, struct lxNode\*);

*token.cpp*

#include "stdafx.h"

#include "token.h"

// порівняння за відношенням порядку

int cmpKys(struct lxNode\*, struct lxNode\*)

{return 0;

}

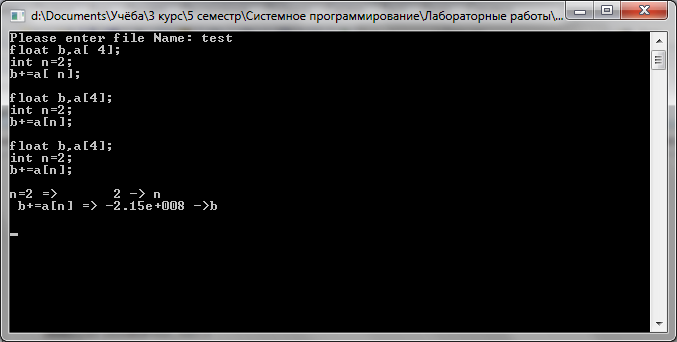
*test.h*

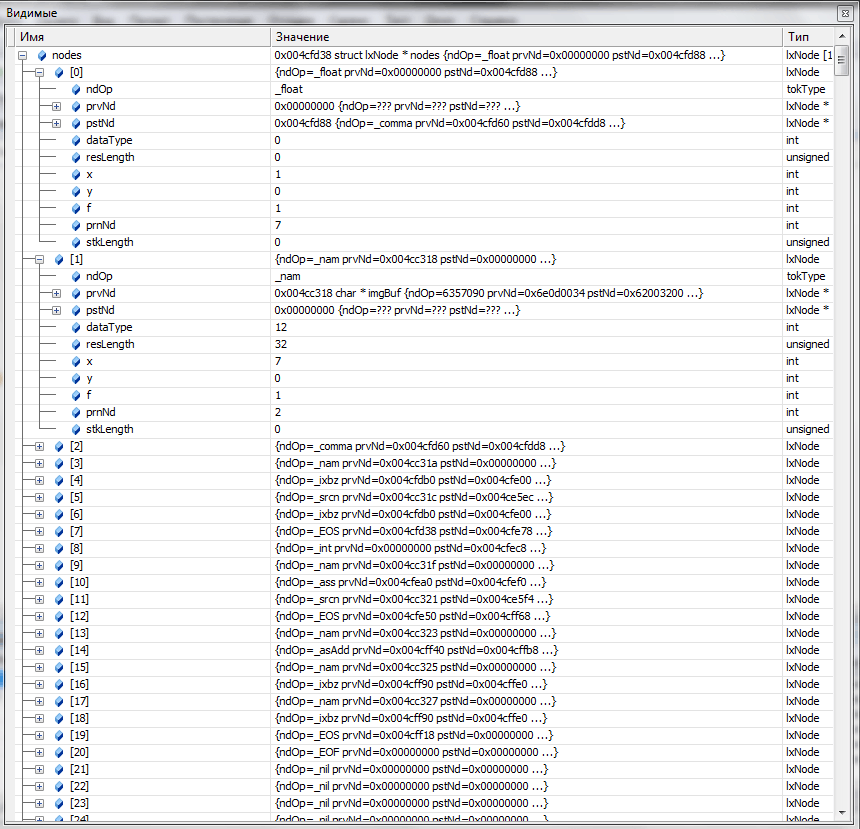
float b, a[4];

int n = 2;

b+=a[n];

***Результати роботи програми***





***Висновки:*** У процесі виконання даної лабораторної роботи ми одержали навички використання асемблерних вставок для оптимізації абстрактної машини інтерпретації комп’ютерної мови, а також вивчили угоди про зв’язки для створення процедур і функцій інтерпретації і звертання до них з операторів мови С.