НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Лабораторна робота №8

з дисципліни **«**Системне програмування**»**

Виконав:

студент 3 курсу

ФІОТ гр. ІО-21

Кузьменко Володимир

Київ – 2014 р.

Мета: одержання навичок побудови програм автоматичної генерації програмних модулів об’єктних кодів для компіляторів з використанням угод базової мови програмування про міжмодульні зв’язки для генерації кодів вставок на мові Асемблер.

Завдання:

1. Визначити варіант завдання для основних задач. Визначити приклади лексем через файл в папці spLb9 модуля тестування spLb9.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 spLb9 для побудови елементу індексу таблиць лексем.
5. Підготувати програмний модуль контрольної задачі, який виконує заданий варіант і дозволяє перевірити коректність виконання програм. Для цього організувати пошук в таблиці відповідності типів результатів типам операндів для семантичного аналізу.
6. Побудувати програмний проект, ввівши програмні модулі у відповідні файли проекту і налагодити синтаксис.
7. Побудувати виконуваний модуль тестової програми і налагодити змістовне виконання програми для перевірки результатів контрольних прикладів.
8. Одержати результати виконання у формі вихідного об’єктного файлу spLb9с.cpp в проекті spLb9с.dsp, проаналізувати виконання цього файлу в режимі налагодження і зробити висновки.
9. Продивившись в режимі дизасемблювання фрагменти операцій, що виконуються в замінити оператори мови С/C++ операторами вставок на мові Асемблер, приклад яких наведено в інтерпретації множення з фіксованою крапкою.
10. Продемонструвати результати викладачам.

|  |  |  |
| --- | --- | --- |
| № вар. | Вираз, який відтворюється в графі внутрішнього подання | Мова відтворення |
| 12 | float b, a[3]; unsigned n=1,d=5; b:=2\*a[n]; b:=d; | С |

Лістинг програми

#include "stdafx.h"

#include "..\Lab2\token.h"

#include "..\Lab2\visgrp.h"

#include "..\Lab3\tables.h"

#include "..\Lab3\lexan.h"

#include "..\Lab4\syntaxP.h"

#include "..\Lab3\langio.h"

#include "..\Lab6\seman.h"

// #include "..\Lab7\interp.h"

#include "codgen.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(); // лексичний аналіз

}

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

prLaTxt(nodes, nn); // реконструкція лексичного аналізу

printf("\n");

SxAnInit('C'); // синтаксичний аналіз

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");

PrCdGen(nodes + nr, "..\\spLb9c\\spLb9c.cpp"); // file\_name);

SmCdGen(nodes + nr, 1);

EpCdGen(nodes + nr);

return 0;

}

*codgen.h*void PrCdGen(struct lxNode \* nd, char\*fn); // вказівник на корінь дерева вузлів);

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

int incR); // кількість повторень

void EpCdGen(struct lxNode \* nd); // вказівник на корінь дерева вузлів);

*codgen.cpp*

#include "stdafx.h"

// #include "interp.h"

#include "..\Lab2\token.h"

#include "..\Lab2\visgrp.h"

#include "..\Lab3\index.h"

#include "..\Lab3\langio.h"

#include "..\Lab4\syntaxP.h"

#include "seman.h"

#include "codgen.h"

#include <stdio.h>

#include <string.h>

extern struct lxNode nodes[MAX\_NODES];

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 пам'яті з кореню дерева, але можна зайти з таблиці

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

}

\*/

union gnDat acc32;

int cc, jmpNmb = 0;

union gnDat stk[32];

char sCom[128] = "";

unsigned char sPtr = 0;

void push

Nd->ndOp >= \_void && nd->prvNd->ndOp < \_fork) WriteCod(";\n");

if (nd->pstNd) dclCdGen(nd->pstNd);

}

}

void PrCdGen(struct lxNode \* nd, char\*fn) // вказівник на корінь дерева вузлів;

{

opFlsC(fn);

WriteCod("// spLb9c.cpp: Визначає вхідну точку для консольної прикладної програми.\n");

WriteCod("//\n\n#include \"stdafx.h\"\n\n\n");

WriteCod("int main(int argc, char\* argv[])\n{");

dclCdGen(nd);

WriteCod(";\n\_asm{");

}

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

int incR) // кількість повторень

{

union gnDat \*vp1, \*vp2;

char // \*name,

\*sOp1 = "", \*sOp2 = "", lbl0[6] = "\_0000", lbl1[6] = "\_0000", lbl2[6] = "\_0000";

int lcLbl;

struct lxNode \*nt;

// struct indStrUS \*pRtNdx;

if (nd->ndOp >= \_void && nd->ndOp <= \_fork) return &acc32; // термінали описів

else

if (nd->ndOp > \_cnst && nd->ndOp != \_remL) // >= \_EOS

{

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

if (dfnFlg != 0 && nd->prvNd && nd->ndOp == \_ass)

{

vp1 = SmCdGen(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 = SmCdGen(nd->pstNd, 1);

sOp2 = (char\*)(nd->prvNd->prvNd);

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

{

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

{

sOp1 = "eax";

if (nd->pstNd->dataType <= \_cnst) sOp1 = (char\*)(nd->pstNd->pstNd);

sprintf(sCom, "\n\tpush\t%s\n\tfild\tdword ptr[esp]", sOp1);

WriteCod(sCom);

sprintf(sCom, "\n\tadd\tesp, 4\n\tfstp\t%s", sOp2);

}

else

sprintf(sCom, "\n\tfstp\t%s", sOp2);

}

else

{

if (nd->pstNd->dataType<\_f) sprintf(sCom, "\n\tmov\t%s, eax", sOp2);

else sprintf(sCom, "\n\tfistp\t%s", sOp2);

}

}

WriteCod(sCom);

prAss(nd);

\*/

return &acc32;

}

// sOp2 = "";

if (nd->prvNd != 0 && nd->ndOp != \_ixbz && ((nd->ndOp & (~\_lVlu)) > \_ass ||

(nd->ndOp & (~\_lVlu)) <= \_qmrk))

{

if (nd->prvNd->ndOp <= \_cnst) sOp2 = (char\*)(nd->prvNd->prvNd);

else sOp2 = ""; // if (sOp2 == 0) sOp2 = "";

if (nd->ndOp == \_whileP)

{

nBgCkl[nbNst++] = jmpNmb;

if (nd->prvNd->ndOp == \_repeat)

{

// cnvLb(lbl0, jmpNmb);

nd->ndOp = \_whileN;

}

else

{

cnvLb(lbl0, jmpNmb++); //,

sOp1 = lbl0;

sprintf(sCom, "\n%s:", sOp1);

WriteCod(sCom);

}

}

else

if (nd->ndOp == \_for)

{

nBgCkl[nbNst++] = jmpNmb;

cnvLb(lbl0, jmpNmb++); //,

lblForHd = lbl0;

}

else sOp1 = "";

if (nd->prvNd->ndOp > \_cnst)

{

vp1 = SmCdGen(nd->prvNd, 1);

if (\*lblForHd != 0 && nd->ndOp == \_EOS && (nd->prvNd == 0 ||

nd->prvNd->ndOp >= \_comma) && (nd->pstNd == 0 || nd->pstNd->ndOp

>= \_comma))

{

sprintf(sCom, "\n%s:", lblForHd);

WriteCod(sCom);

}

}

else

if (nd->pstNd != 0 && (nd->ndOp & \_lVlu) == 0)

{

if ((nd->prvNd->dataType & (cdCns - 1)) < \_f ||

nd->prvNd->dataType >= cdPtr)

{

if ((nd->pstNd->dataType & (cdCns - 1)) < \_f || nd->ndOp < \_cln)

{

if (nd->prvNd->dataType & cdArr)

sprintf(sCom,"\n\tlea\teax, %s", sOp2);

else sprintf(sCom,"\n\tmov\teax, %s", sOp2);

if (nd->ndOp != \_div && nd->ndOp != \_mul && nd->ndOp != \_mod)

sOp1 = "eax, ";

}

else

sprintf(sCom, "\n\tfild\t%s", sOp2);

}

else

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

sprintf(sCom, "\n\tfld\t%s", sOp2);

WriteCod(sCom);

sOp2 = "";

}

}

if (nd->ndOp == \_repeat)

{

cnvLb(lbl0, jmpNmb++); //,

sOp1 = lbl0;

sprintf(sCom, "\n%s:", sOp1);

WriteCod(sCom);

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

nodes[nodes[nd->prnNd].prnNd].pstNd->ndOp = \_whileN;

}

if (nd->pstNd)

{

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

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

{

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

{

sprintf(sCom, "\n\tpush\teax");

WriteCod(sCom);

if ((nd->pstNd->dataType & msStp) >= \_f)

{

sprintf(sCom, "\n\tfild\t[esp]\n\tadd esp, 4");

WriteCod(sCom);

}

}

}

// push(acc32);

if (nd->pstNd != 0 && (nd->ndOp > \_fork && nd->ndOp != \_qmrk && nd->ndOp != \_cln

/\* || nd->ndOp == \_whileN \*/)

/\* && nd->ndOp != \_if && nd->ndOp != \_else \*/)

{

if (nd->pstNd->ndOp > \_cnst)

{

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

if (\*lblForHd != 0 && nd->ndOp == \_EOS && (nd->prvNd == 0 ||

nd->prvNd->ndOp >= \_comma) && (nd->pstNd == 0 ||

nd->pstNd->ndOp >= \_comma))

{

cnvLb(lbl0, nBgCkl[nbNst - 1] + 1);

sprintf(sCom, "\n\ttest\teax, eax\n\tjz\t%s", lbl0);

WriteCod(sCom);

lblForHd = "";

}

if ((nd->ndOp & (~\_lVlu)) <= \_ass && (nd->ndOp & (~\_lVlu)) > \_qmrk

&& (((nd->prvNd->dataType & msStp) >= \_f && (nd->pstNd->dataType

& msStp) < \_f) || ((nd->prvNd->ndOp & \_lVlu) != 0) &&

nd->prvNd->pstNd != 0 && nd->prvNd->pstNd->ndOp > \_ass))

{

sprintf(sCom, "\n\tpush\teax", sOp2);

// \n\tfild\t[esp]\n\tadd\tesp, 4

WriteCod(sCom);

sOp2 = "[esp]";

}

}

else

if ((nd->ndOp & (~\_lVlu)) <= \_ass && (nd->ndOp & (~\_lVlu)) > \_qmrk)

{

sOp2 = (char\*)(nd->pstNd->prvNd);

if ((nd->pstNd->dataType & msStp) < \_f)

{

if ((nd->prvNd->dataType & msStp) < \_f)

{

sprintf(sCom, "\n\tmov\teax, %s", sOp2);

if (nd->ndOp != \_div && nd->ndOp != \_mul && nd->ndOp != \_mod)

sOp1 = "eax, ";

}

/\*

else -11.07.07

sprintf(sCom, "\n\tfild\t%s", sOp2);

\*/

}

/\*

else

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

sprintf(sCom, "\n\tfld\t%s", sOp2);

WriteCod(sCom);

sOp2 = "";

\*/

}

}

}

else

if (nd->ndOp == \_brkz) // || nd->ndOp == \_whileN)

{

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

return &acc32;

}

if (/\* (nd->ndOp & (~\_lVlu)) == \_ixbz || \*/

(nd->ndOp & \_lVlu) != 0 && nd->prvNd->ndOp > \_cnst)

{

vp1 = SmCdGen(nd->prvNd, 1);

// 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;

\*sCom = 0;

/\* if (nd->prvNd && nd->prvNd->dataType <= \_cnst) sOp2 = (char\*)(nd->prvNd->prvNd); \*/

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

<\_f && nd->pstNd != 0 && (nd->pstNd->dataType & msStp) >= \_f && (nd->ndOp

& (~\_lVlu)) > \_ass)

{

sOp1 = "eax";

if (nd->prvNd->dataType <= \_cnst) sOp1 = sOp2;

sprintf(sCom, "\n\tpush\t%s\n\tfild\t[esp]\n\tadd\tesp, 4", sOp1);

WriteCod(sCom);

sOp1 = "";

}

else

/\*

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

(nd->pstNd->dataType & msStp) < \_f && (nd->ndOp & (~\_lVlu))

>= \_asOr && (nd->ndOp & (~\_lVlu)) <= \_ass) // Shl)

{

// sOp1 = "eax"; // 10.07.07

if (nd->prvNd->dataType <= \_cnst)

{

sOp1 = sOp2;

sprintf(sCom, "\n\tpush\t%s", sOp1);

WriteCod(sCom);

sOp1 = "[esp]";

}

}

else

\*/

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

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

{

if ((nd->pstNd->dataType & msStp) < \_f)

{

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

{

sOp2 = ", eax";

}

if ((nd->prvNd->dataType & msStp) >= \_f) WriteCod("\n\tpush\teax");

else

if (nd->ndOp == \_div || nd->ndOp == \_mul || nd->ndOp == \_mod)

{

WriteCod("\n\txchg\teax, [esp]");

sOp2 = "";

}

sOp1 = "[esp]";

}

}

else

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

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

(nd->pstNd->dataType & msStp) < \_f && (nd->ndOp & \_lVlu) == 0)

{

if (nd->ndOp != \_div && nd->ndOp != \_mul && nd->ndOp != \_mod &&

nd->ndOp != \_shRgt && nd->ndOp != \_shLft)

sOp1 = "eax, ";

else sOp1 = "";

} // \t

if ((nd->ndOp > \_cnst && nd->ndOp < \_with && nd->ndOp != \_whileN)

|| nd->ndOp == \_qmrk || nd->ndOp == \_cln)

{

lcLbl = jmpNmb++;

cnvLb(lbl1, lcLbl); //,

sOp1 = lbl1;

}

if (nd->ndOp == \_break)

{

cnvLb(lbl0, nBgCkl[nbNst - 1] + 1);

sOp1 = lbl0;

}

if (nd->ndOp == \_continue)

{

cnvLb(lbl0, nBgCkl[nbNst - 1]);

sOp1 = lbl0;

}

if (nd->ndOp == \_for || nd->ndOp == \_whileN)

{

cnvLb(lbl1, nBgCkl[nbNst - 1]);

sOp1 = lbl1;

cnvLb(lbl0, nBgCkl[nbNst - 1] + 1);

sOp2 = lbl0;

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

nbNst--;

}

// сюди вставляти потрібні операції

ftImp.oprd1 = 0;

ftImp.ln1 = 0;

ftImp.oprd2 = 0;

ftImp.ln2 = 0;

if (nd->prvNd)

{

ftImp.oprd1 = nd->prvNd->dataType & (~cdCns);

ftImp.ln1 = nd->prvNd->resLength;

}

if (nd->pstNd)

{

ftImp.oprd2 = nd->pstNd->dataType & (~cdCns);

ftImp.ln2 = nd->pstNd->resLength;

}

ftImp.oprtn = nd->ndOp;

struct recrdSMA\* pftImp = selBin(&ftImp, ftTbl, 619);

if (pftImp)

{

// sOp2 = "";

/\*

if (nd->pstNd != 0 && nd->pstNd->ndOp <= \_cnst)

sOp2 = (char\*)(nd->pstNd->prvNd);

\*/

strcpy(sCom, pftImp->assCd);

if ((nd->ndOp & (~\_lVlu)) > \_qmrk && (nd->ndOp & (~\_lVlu)) <= \_ass)

{

if(!strcmp(sOp1, "[esp]")) sOp2 = sOp1;

else

if ((nd->pstNd->dataType & msStp) < \_f &&

(nd->dataType & msStp) < \_f) sOp2 = "eax";

sOp1 = (char\*)(nd->prvNd->prvNd); // ???

if ((nd->prvNd->dataType & msStp) < \_f && (nd->pstNd->dataType

& msStp) < \_f && (nd->ndOp == \_asDiv || nd->ndOp == \_asMod))

sOp2 = sOp1;

if (nd->ndOp & \_lVlu) sOp1 = "[ebx]";

}

else

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

{

sOp2 = (char\*)(nd->prvNd->prvNd);

if (nd->pstNd->ndOp <= \_cnst) sOp1 = (char\*)(nd->pstNd->prvNd);

else sOp1 = "eax";

}

else

if ((nd->ndOp >= \_dcr && nd->ndOp <= \_inrN) || nd->ndOp ==

\_shRgt || nd->ndOp == \_shLft) sOp1 = sOp2;

sprintf(sCom, pftImp->assCd, sOp1, sOp2, sOp1);

WriteCod(sCom);

if (nd->ndOp <= \_ass && nd->ndOp > \_qmrk) prAss(nd);

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

if (nd->pstNd != 0 && nd->ndOp > \_ass && nd->ndOp <= \_frkz &&

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

\_cnst && (nd->dataType & msStp) < \_f && nd->ndOp != \_div &&

nd->ndOp != \_mul && nd->ndOp != \_mod) WriteCod("\n\tpop\teax");

else

if (!strcmp(sOp1, "[esp]") || !strcmp(sOp2, "[esp]"))

{

sprintf(sCom, "\n\tadd\tesp, 4");

WriteCod(sCom);

}

}

if (nd->ndOp == \_if || nd->ndOp == \_cln || nd->ndOp == \_else

|| nd->ndOp == \_qmrk || nd->ndOp == \_whileP || nd->ndOp == \_repeat)

// && cc != 0)

{

if (nd->pstNd != 0)

{

if (nd->pstNd->ndOp > \_cnst) vp2 = SmCdGen(nd->pstNd, 1);

else

{

if ((nd->pstNd->dataType & msStp) < \_f)

sprintf(sCom, "\n\tmov\teax, %s", (char\*)(nd->pstNd->prvNd));

else sprintf(sCom, "\n\tfld\t%s", (char\*)(nd->pstNd->prvNd));

WriteCod(sCom);

}

}

if (nd->ndOp == \_whileP)

{

sprintf(sCom,"\n\tjmp\t%s", lbl0);

WriteCod(sCom);

}

sOp1 = "";

lbl2[0] = 0;

if ((nd->ndOp == \_if && nodes[nd->prnNd].ndOp == \_else) || nd->ndOp == \_qmrk)

{

sOp1 = "\n\tjmp\t";

lbl2[0] = '\_';

cnvLb(lbl2, lcLbl + 1);

}

if (nd->ndOp != \_repeat)

{

sprintf(sCom, "%s%s\n%s:", sOp1, lbl2, lbl1);

lbl2[0] = '\_';

WriteCod(sCom);

}

}

// cnvLb(lbl0, ++jmpNmb); //,

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 = SmCdGen(nd->prvNd->pstNd, 1);

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

else return vp1;

}

while(vp1->\_id);

}

\*/

return &acc32;

}

*seman.h*

/\*

enum lngType

{

\_V, \_EX, \_VR, \_CNS, \_ENM, \_STR /\*\_record/, \_UNN, \_REG, //

\_UI, \_SG, \_SC, \_HI, \_SI, \_LI, \_SI64, \_UI64, //

\_F, \_D, \_L, \_LBL, \_AU, \_ST, \_VL, \_TP, \_SZOF, //

\_REAL, \_ARR, \_SET, \_FILE, \_OBJ, \_STRN, \_CU, //

};

\*/

// модифікатори

#define cdPtr 0x20000 // код покажчика 1-го рівня

#define cdCns 0x8000 // код константного типу даних

#define cdArr 0x10000 // код даних типу масиву

#define cdCna 0x18000 // код константного масиву

#define cdReg 0x10000000 // код регістрового типу даних

#define cdExt 0x20000000 // код зовнішнього типу даних

#define cdStt 0x30000000 // код статичного типу даних

#define cdAut 0x40000000 // код автоматичного типу даних

#define msMod 0xffff8000 // маска модифікаторів типів

#define msStp 0x00007fff // маска стандартних типів

enum datType

{

\_v, // порожній тип даних

\_uc = 4, \_us, \_ui, \_ui64, // стандартні цілі без знака

\_sc = 8, \_ss, \_si, \_si64, // стандартні цілі зі знаком

\_geq = 0x0ffe, // загальний тип для рівності

\_gen = 0x0fff, // загальний (довільний) тип

\_enm = 0x1000, // перенумеровані типи enum

\_f = 0x1ff0, \_d, \_ld, \_rea, // дані з плаваючою точкою

\_str = 0x2000, // структурні типи /\* \_record \*/,

\_unn = 0x3000, // типи об’єднань union

\_cls = 0x4000, // типи класів

\_obj = 0x5000, // типи об’єктів

\_fun = 0x6000, // функціональні типи

\_ctp = 0x7000, // умовні типи мови Pascal

\_lbl, \_strn, // мітки

// інші стандартні типи

\_fl, \_tp, \_vl, \_vr, //

};

struct recrdTMD // структура рядка таблиці базових типів

{

enum datType tpLx; // примірник структури ключа

unsigned md; // модифікатор

unsigned ln; // базова або гранична довжина даних типу

};

struct recrdTPD // структура рядка таблиці модифікованих типів

{

enum tokType kTp[3]; // примірник структури ключа

enum datType dTp; // примірник функціональної частини

int ln; // базова довжина даних типу

};

// вибірка за двійковим пошуком

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

// порівняння рядків

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

union gnDat

{

int \_id;

short \_sd;

char \_cd;

float \_fd;

double \_dd;

\_\_int64 \_i8;

};

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

struct recrdSMA // структура рядка таблиці припустимості типів для операцій

{

enum tokType oprtn;

int oprd1, ln1;

int oprd2, ln2;

int res, lnRes;

\_fop \*pintf;

char \*assCd;

};

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

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

// функції етапів проектування і настроювання!

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

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

int nR); // номер кореневого вузла

*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 ім'я

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 "..\Lab2\token.h"

#include "seman.h"

#include "..\Lab7\interpF.h"

union gnDat \*\_pd;

// \_fop \*\_paddf =& \_fadd;

struct recrdSMA ftTbl[619] = // таблиця припустимості типів для операцій

{

{\_if, \_ui, 32, \_v, 0, \_v, 0, "\n\tcmp\teax, 0\n\tjz\t%s"},

{\_if, \_ui, 32, \_ui, 32, \_v, 0, "\n\tcmp\teax, 0\n\tjz\t%s"},

{\_if, \_ui, 32, \_si, 32, \_v, 0, "\n\tcmp\teax, 0\n\tjz\t%s"},

{\_if, \_ui, 32, \_f, 32, \_v, 0, "\n\tcmp\teax, 0\n\tjz\t%s"},

{\_if, \_ui, 32, \_d, 64, \_v, 0, "\n\tcmp\teax, 0\n\tjz\t%s"},

{\_if, \_si, 32, \_v, 0, \_v, 0, "\n\tcmp\teax, 0\n\tjz\t%s"},

{\_if, \_si, 32, \_ui, 32, \_v, 0, "\n\tcmp\teax, 0\n\tjz\t%s"},

{\_if, \_si, 32, \_si, 32, \_v, 0, "\n\tcmp\teax, 0\n\tjz\t%s"},

{\_if, \_si, 32, \_f, 32, \_v, 0, "\n\tcmp\teax, 0\n\tjz\t%s"},

{\_if, \_si, 32, \_d, 64, \_v, 0, "\n\tcmp\teax, 0\n\tjz\t%s"},

{\_if, \_f, 32, \_v, 0, \_v, 0, "\n\tftst\n\tfstsw\tax\n\tsahf\n\tfincstp\n\tjz\t%s"},

{\_if, \_f, 32, \_ui, 32, \_v, 0, "\n\tftst\n\tfstsw\tax\n\tsahf\n\tfincstp\n\tjz\t%s"},

{\_if, \_f, 32, \_si, 32, \_v, 0, "\n\tftst\n\tfstsw\tax\n\tsahf\n\tfincstp\n\tjz\t%s"},

{\_if, \_f, 32, \_f, 32, \_v, 0, "\n\tftst\n\tfstsw\tax\n\tsahf\n\tfincstp\n\tjz\t%s"},

{\_if, \_f, 32, \_d, 64, \_v, 0, "\n\tftst\n\tfstsw\tax\n\tsahf\n\tfincstp\n\tjz\t%s"},

{\_if, \_d, 64, \_v, 0, \_v, 0, "\n\tftst\n\tfstsw\tax\n\tsahf\n\tfincstp\n\tjz\t%s"},

{\_if, \_d, 64, \_ui, 32, \_v, 0, "\n\tftst\n\tfstsw\tax\n\tsahf\n\tfincstp\n\tjz\t%s"},

{\_if, \_d, 64, \_si, 32, \_v, 0, "\n\tftst\n\tfstsw\tax\n\tsahf\n\tfincstp\n\tjz\t%s"},

{\_if, \_d, 64, \_f, 32, \_v, 0, "\n\tftst\n\tfstsw\tax\n\tsahf\n\tfincstp\n\tjz\t%s"},

{\_if, \_d, 64, \_d, 64, \_v, 0, "\n\tftst\n\tfstsw\tax\n\tsahf\n\tfincstp\n\tjz\t%s"},

i, 32, \_ui, 32, "\n\tficomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tsetne\tal\n\tmovsx\teax, al"},

{\_ne, \_d, 64, \_f, 32, \_ui, 32, "\n\tfcomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tsetne\tal\n\tmovsx\teax, al"},

{\_ne, \_d, 64, \_d, 64, \_ui, 32, "\n\tfcomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tsetne\tal\n\tmovsx\teax, al"},

{\_ge, \_ui, 32, \_ui, 32, \_ui, 32, "\n\tcmp\t%s%s\n\tsetae\tal\n\tmovsx\teax, al"},

{\_ge, \_ui, 32, \_si, 32, \_ui, 32, "\n\tcmp\t%s%s\n\tsetge\tal\n\tmovsx\teax, al"},

{\_ge, \_ui, 32, \_f, 32, \_ui, 32, "\n\tfcomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tsetae\tal\n\tmovsx\teax, al"},

{\_ge, \_ui, 32, \_d, 64, \_ui, 32, "\n\tfcomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tsetae\tal\n\t\tmovsx\teax, al"},

{\_ge, \_si, 32, \_ui, 32, \_ui, 32, "\n\tcmp\t%s%s\n\tsetge\tal\n\tmovsx\teax, al"},

{\_ge, \_si, 32, \_si, 32, \_ui, 32, "\n\tcmp\t%s%s\n\tsetge\tal\n\tmovsx\teax, al"},

{\_ge, \_si, 32, \_f, 64, \_ui, 32, "\n\tfcomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tsetae\tal\n\tmovsx\teax, al"},

{\_ge, \_si, 32, \_d, 64, \_ui, 32, "\n\tfcomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tsetae\tal\n\tmovsx\teax, al"},

{\_ge, \_f, 32, \_ui, 32, \_ui, 32, "\n\tficomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tsetae\tal\n\tmovsx\teax, al"},

{\_ge, \_f, 32, \_si, 32, \_ui, 32, "\n\tficomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tsetae\tal\n\tmovsx\teax, al"},

{\_ge, \_f, 32, \_f, 32, \_ui, 32, "\n\tfcomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tsetae\tal\n\tmovsx\teax, al"},

{\_ge, \_f, 32, \_d, 64, \_ui, 32, "\n\tfcomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tsetae\tal\n\tmovsx\teax, al"},

{\_ge, \_d, 64, \_ui, 32, \_ui, 32, "\n\tficomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tsetae\tal\n\tmovsx\teax, al"},

{\_ge, \_d, 64, \_si, 32, \_ui, 32, "\n\tficomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tsetae\tal\n\tmovsx\teax, al"},

{\_ge, \_d, 64, \_f, 32, \_ui, 32, "\n\tfcomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tsetae\tal\n\tmovsx\teax, al"},

{\_ge, \_d, 64, \_d, 64, \_ui, 32, "\n\tfcomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tsetae\tal\n\tmovsx\teax, al"},

{\_gt, \_ui, 32, \_ui, 32, \_ui, 32, "\n\tcmp\t%s%s\n\tseta\tal\n\tmovsx\teax, al"},

{\_gt, \_ui, 32, \_si, 32, \_ui, 32, "\n\tcmp\t%s%s\n\tsetg\tal\n\tmovsx\teax, al"},

{\_gt, \_ui, 32, \_f, 32, \_ui, 32, "\n\tfcomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tseta\tal\n\tmovsx\teax, al"},

{\_gt, \_ui, 32, \_d, 64, \_ui, 32, "\n\tfcomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tseta\tal\n\t\tmovsx\teax, al"},

{\_gt, \_si, 32, \_ui, 32, \_ui, 32, "\n\tcmp\t%s%s\n\tsetg\tal\n\tmovsx\teax, al"},

{\_gt, \_si, 32, \_si, 32, \_ui, 32, "\n\tcmp\t%s%s\n\tsetg\tal\n\tmovsx\teax, al"},

{\_gt, \_si, 32, \_f, 32, \_ui, 32, "\n\tfcomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tseta\tal\n\tmovsx\teax, al"},

{\_gt, \_si, 32, \_d, 64, \_ui, 32, "\n\tfcomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tseta\tal\n\tmovsx\teax, al"},

{\_gt, \_f, 32, \_ui, 32, \_ui, 32, "\n\tficomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tseta\tal\n\tmovsx\teax, al"},

{\_gt, \_f, 32, \_si, 32, \_ui, 32, "\n\tficomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tseta\tal\n\tmovsx\teax, al"},

{\_gt, \_f, 32, \_f, 32, \_ui, 32, "\n\tfcomp\t%s%s\n\tfstsw\tax\n\tsahf\n\tseta\tal\n\tmovsx\teax, al"},

{\_gt, \_f, 32

n\tpop\teax"},

{\_sub, \_ui + cdPtr, 32, \_si, 32, \_ui + cdPtr, 32, "\n\tpush\teax\n\tmov\t%s%s\n\tmov\tcl, 2\n\tshl\teax, cl\n\tsub\t[esp], eax\n\tpop\teax"},

{\_sub, \_ui + cdPtr, 32, \_ui + cdPtr, 32, \_si, 32, "\n\tsub\t%s%s\n\tmov\tcl, 2\n\tshr\teax, cl"},

{\_sub, \_si + cdPtr, 32, \_ui, 32, \_si + cdPtr, 32, "\n\tpush\teax\n\tmov\t%s%s\n\tmov\tcl, 2\n\tshl\teax, cl\n\tsub\t[esp], eax\n\tpop\teax"},

{\_sub, \_si + cdPtr, 32, \_si, 32, \_si + cdPtr, 32, "\n\tpush\teax\n\tmov\t%s%s\n\tmov\tcl, 2\n\tshl\teax, cl\n\tsub\t[esp], eax\n\tpop\teax"},

{\_sub, \_si + cdPtr, 32, \_si + cdPtr, 32, \_si, 32, "\n\tsub\t%s%s\n\tmov\tcl, 2\n\tshr\teax, cl"},

{\_sub, \_f + cdPtr, 32, \_ui, 32, \_f + cdPtr, 32, "\n\tpush\teax\n\tmov\t%s%s\n\tmov\tcl, 2\n\tshl\teax, cl\n\tsub\t[esp], eax\n\tpop\teax"},

{\_sub, \_f + cdPtr, 32, \_si, 32, \_f + cdPtr, 32, "\n\tpush\teax\n\tmov\t%s%s\n\tmov\tcl, 2\n\tshl\teax, cl\n\tsub\t[esp], eax\n\tpop\teax"},

{\_sub, \_f + cdPtr, 32, \_f + cdPtr, 32, \_si, 32, "\n\tsub\t%s%s\n\tmov\tcl, 2\n\tshr\teax, cl"},

{\_sub, \_d + cdPtr, 64, \_ui, 32, \_d + cdPtr, 64, "\n\tpush\teax\n\tmov\t%s%s\n\tmov\tcl, 3\n\tshl\teax, cl\n\tsub\t[esp], eax\n\tpop\teax"},

{\_sub, \_d + cdPtr, 64, \_si, 32, \_d + cdPtr, 64, "\n\tpush\teax%s\n\tmov\teax, 8\n\tmul\t%s\n\tsub\t[esp], eax\n\tpop\teax"},

{\_sub, \_d + cdPtr, 64, \_d + cdPtr, 64, \_si, 32, "\n\tsub\t%s%s\n\tmov\tcl, 3\n\tshr\teax, cl"},

{\_sub, \_ui + cdPtr + cdArr, 32, \_ui, 32, \_ui + cdPtr, 32, "\n\tpush\teax\n\tmov\t%s%s\n\tmov\tcl, 2\n\tshl\teax, cl\n\tsub\t[esp], eax\n\tpop\teax"},

{\_sub, \_ui + cdPtr + cdArr, 32, \_si, 32, \_ui + cdPtr, 32, "\n\tpush\teax\n\tmov\t%s%s\n\tmov\tcl, 2\n\tshl\teax, cl\n\tsub\t[esp], eax\n\tpop\teax"},

{\_sub, \_ui + cdPtr + cdArr, 32, \_ui + cdPtr, 32, \_si, 32, "\n\tsub\t%s%s\n\tmov\tcl, 2\n\tshr\teax, cl"},

{\_sub, \_si + cdPtr + cdArr, 32, \_ui, 32, \_si + cdPtr, 32, "\n\tpush\teax\n\tmov\t%s%s\n\tmov\tcl, 2\n\tshl\teax, cl\n\tsub\t[esp], eax\n\tpop\teax"},

{\_sub, \_si + cdPtr + cdArr, 32, \_si, 32, \_si + cdPtr, 32, "\n\tpush\teax\n\tmov\t%s%s\n\tmov\tcl, 2\n\tshl\teax, cl\n\tsub\t[esp], eax\n\tpop\teax"},

{\_sub, \_si + cdPtr + cdArr, 32, \_si + cdPtr, 32, \_si, 32, "\n\tsub\t%s%s\n\tmov\tcl, 2\n\tshr\teax, cl"},

{\_sub, \_f + cdPtr + cdArr, 32, \_ui, 32, \_f + cdPtr, 32, "\n\tpush\teax\n\tmov\t%s%s\n\tmov\tcl, 2\n\tshl\teax, cl\n\tsub\t[esp], eax\n\tpop\teax"},

{\_sub, \_f + cdPtr + cdArr, 32, \_si, 32, \_f + cdPtr, 32, "\n\tpush\teax\n\tmov\t%s%s\n\tmov\tcl, 2\n\tshl\teax, cl\n\tsub\t[esp], eax\n\tpop\teax"},

{\_sub, \_f + cdPtr + cdArr, 32, \_f + cdPtr, 32, \_si, 32, "\n\tsub\t%s%s\n\tmov\tcl, 2\n\tshr\teax, cl"},

{\_sub, \_d + cdPtr + cdArr, 32, \_ui, 32, \_d + cdPtr, 32, "\n\tpush\teax\n\tmov\t%s%s\n\tmov\tcl, 3\n\tshl\teax, cl\n\tsub\t[esp], eax\n\tpop\teax"},

{\_sub, \_d + cdPtr + cdArr, 32, \_si, 32, \_d + cdPtr, 32, "\n\tpush\teax%s\n\tmov\teax, 8\n\tmul\t%s\n\tsub\t[esp], eax\n\tpop\teax"},

{\_sub, \_d + cdPtr + cdArr, 32, \_d + cdPtr, 32, \_si, 32, "\n\tsub\t%s%s\n\tmov\tcl, 3\n\tshr\teax, cl"},

{\_mul, \_v, 0, \_ui + cdPtr, 32, \_ui, 32, "\n\tmov\tesi, %s%s\n\tmov\teax, [esi]"},

{\_mul, \_v, 0, \_si + cdPtr, 32, \_si, 32, "\n\tmov\tesi, %s%s\n\tmov\teax, [esi]"},

{\_mul, \_v, 0, \_f + cdPtr, 32, \_f, 32, "\n\tmov\tesi%s%s\n\tfld\tdword ptr[eax]"},

{\_mul, \_v, 0, \_d + cdPtr, 32, \_d, 64, "\n\tmov\tesi%s%s\n\tfld\tqword ptr[eax]"},

{\_mul, \_v, 0, \_ui + cdPtr + cdArr, 32, \_ui, 32, "\n\tlea\tesi, %s%s\n\tmov\teax, [esi]"},

{\_mul, \_v, 0, \_si + cdPtr + cdArr, 32, \_si, 32, "\n\tlea\tesi, %s%s\n\tmov\teax, [esi]"},

{\_mul, \_v, 0, \_f + cdPtr + cdArr, 32, \_f, 32, "\n\tlea\tesi%s%s\n\tfld\tdword ptr[eax]"},

{\_mul, \_v, 0, \_d + cdPtr + cdArr, 32, \_d, 64, "\n\tlea\tesi%s%s\n\tfld\tqword ptr[eax]"},

{\_mul, \_ui, 32, \_ui, 32, \_ui, 32, "\n\tmul\t%s%s"}, // eax

{\_mul, \_ui, 32, \_si, 32, \_si, 32, "\n\timul\t%s%s"}, // eax

{\_mul, \_ui, 32, \_f, 32, \_f, 32, "\n\tfmul\t%s%s"},

{\_mul, \_ui, 32, \_d, 64, \_d, 64, "\n\tfmul\t%s%s"},

{\_mul, \_si, 32, \_ui, 32, \_si, 32, "\n\timul\t%s%s"}, // eax

{\_mul, \_si, 32, \_si, 32, \_si, 32, "\n\timul\t%s%s"}, // eax

{\_mul, \_si, 32, \_f, 32, \_f, 32, "\n\tfmul\t%s%s"},

{\_mul, \_si, 32, \_d, 64, \_d, 64, "\n\tfmul\t%s%s"},

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

%s\n\tmov\tebx, %s\n\tlea\tebx, [ebx][esi\*8]"},

{(enum tokType)(\_ixbz + \_lVlu), \_ui + cdPtr + cdArr, 32, \_ui, 32, \_ui, 32, "\n\tmov\tesi, %s\n\tlea\tebx, %s[esi\*4]"},

{(enum tokType)(\_ixbz + \_lVlu), \_ui + cdPtr + cdArr, 32, \_si, 32, \_ui, 32, "\n\tmov\tesi, %s\n\tlea\tebx, %s[esi\*4]"},

{(enum tokType)(\_ixbz + \_lVlu), \_si + cdPtr + cdArr, 32, \_ui, 32, \_si, 32, "\n\tmov\tesi, %s\n\tlea\tebx, %s[esi\*4]"},

{(enum tokType)(\_ixbz + \_lVlu), \_si + cdPtr + cdArr, 32, \_si, 32, \_si, 32, "\n\tmov\tesi, %s\n\tlea\tebx, %s[esi\*4]"},

{(enum tokType)(\_ixbz + \_lVlu), \_f + cdPtr + cdArr, 32, \_ui, 32, \_f, 32, "\n\tmov\tesi, %s\n\tlea\tebx, %s[esi\*4]"},

{(enum tokType)(\_ixbz + \_lVlu), \_f + cdPtr + cdArr, 32, \_si, 32, \_f, 32, "\n\tmov\tesi, %s\n\tlea\tebx, %s[esi\*4]"},

{(enum tokType)(\_ixbz + \_lVlu), \_d + cdPtr + cdArr, 32, \_ui, 32, \_d, 64, "\n\tmov\tesi, %s\n\tlea\tebx, %s[esi\*8]"},

{(enum tokType)(\_ixbz + \_lVlu), \_d + cdPtr + cdArr, 32, \_si, 32, \_d, 64, "\n\tmov\tesi, %s\n\tlea\tebx, %s[esi\*8]"},

};

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

\_f, // 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},

};

**Висновки:** я одержав навички побудови програм автоматичної генерації програмних модулів об’єктних кодів для компіляторів з використанням угод базової мови програмування про міжмодульні зв’язки для генерації кодів вставок на мові Асемблер.