НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Лабораторна робота №7

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

Виконав:

студент 3 курсу

ФІОТ гр. ІО-21

Кузьменко Володимир

Київ – 2014 р.

**Завдання на роботу**

Завдання на підготовку до роботи на комп’ютері:

1. Визначити варіант завдання для основних задач за табли­цею 8.1. 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 для прискорення пошуку і визначити функції доступу до імен за індексом.

5. Підготувати програмний модуль контрольної задачі, який виконує заданий варіант з таблиці 7.1 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 виконуваний 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 інтерпретаторі, замінити оператори мови С/C++ операторами асемблерних вставок, приклад яких наведено в інтерпретації множення з фіксованою точкою.

10. Продемонструвати результати викладачам.

Варіант завдання:

|  |  |  |
| --- | --- | --- |
| 12 | float **b, a[3];** unsigned **n,d; b:=2\*a[n];** **b:=d;** | Pascal |

**Лістинг:**

*Lab7.cpp*

#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 "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;

}

*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 "..\Lab2\token.h"

#include "..\Lab6\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;

}

*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 "..\Lab2\token.h"

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

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

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

#include "..\Lab6\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 пам'яті 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) acc32.\_fd = vp1->\_fd\*vp2->\_fd;

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

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

#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 ім'я

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;

}

*semanT.cpp*

#include "stdafx.h"

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

#include "..\Lab6\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

\_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 використання вставок на мові Асемблер для побудови та оптимізації абстрактної машини інтерпретації комп’ютерної мови. Мною були вивчені угоди про зв’язки для створення процедур і функцій інтерпретації операцій і операторів комп’ютерних мов і звертання до них за допомогою операторів мови С з використанням функціонального типу даних.