МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

отчет

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Программирование рекурсивных алгоритмов

Студентка гр. 9303 _____ Зарезина Е.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

Цель работы.

Ознакомление со структурой иерархического списка, способом реализации и разработка базовых функций для работы со списком и решения рекурсивной задачи.

Основные теоретические положения.

Линейный однонаправленный (односвязный) список - список, каждый элемент которого хранит помимо значения указатель на следующий элемент. В последнем элементе указатель на следующий элемент равен NULL (константа нулевого указателя).

Иерархический список – способ представления списка смежности, т.е. для каждой вершины указан список смежных с нею вершин Иерархические списки отличаются от линейных тем, что элементы такого списка могут представлять из себя как значения(*атомы*), так и другие иерархические списки. Из данного определения очевидно что обработка подобных списков наиболее удобна при помощи рекурсивных алгоритмов. Представлять иерархические списки в программе удобнее в виде структуры из двух указателей – элемент(атом/список) и на другую структуру списка.

Выполнение работы.

int main() – Пользвателю предлагается выбор: ввод последовательности через консоль, либо считывание из файла. Если пользователь выбрал ввод с консоли, то вызывается функция read_lisp() для считывания иерархического списка с консоли. Если было выбрано считывание из файла, то в функции считывается имя необходимого файла и выводится сообщение об ошибке, если файла не существует, либо его невозможно открыть. Если не возникло ошибки при открытии файла, то выводится строка, считанная из файла, а затем она передаётся в функцию read_lisp_file() для обработки иерархического списка из файла. Далее запускается функция count_lisp(), производящая подсчёт элементов в иерархическом списке, функция make_list_lisp(), составляющая

линейный список из элементов иерархического списка и функция write_list(), выводящая получившийся линейный список.

Тестирование.

1)

1 - console

2 - file

Your choice:

1

Enter your list :

(kate)

(- It's not Atom

k - it's Atom

count_1 = 1

Head sum = 1

a - it's Atom

count_1 = 1

Head sum = 1

t - it's Atom

count_1 = 1

Head sum = 1

e - it's Atom

count_1 = 1

Head sum = 1

Tail = 0 !!!

Equal to head and tail's sum

Head = 1

Tail = 1

Result sum = 1

Equal to head and tail's sum

Head = 1

Tail = 1

Result sum = 3

Equal to head and tail's sum

Head = 1

Tail = 1

Result sum = 4

count_2 = 4

Head sum = 4

(- It's not Atom

List consist of head and tail elements

1 : k(107)

List consist of head and tail elements

1 : a(97)

List consist of head and tail elements

1 : t(116)

List consist of head and tail elements

1 : e(101)

No elements in tail!

1 : e(101)

1 : t(116)

2 : e(101)

1 : a(97)

2 : t(116)

3 : e(101)

1 : k(107)

2 : a(97)

3 : t(116)

4 : e(101)

Number of elements in list = 4
Linear list: 1 : k(107)

2 : a(97)

3 : t(116)

4 : e(101)

Ввод:	**Вывод:**
a	Number of elements in list = 1 Linear list: 1 : a(97)
()	Number of elements in list = 0 Linear list:

hfusifuh	Number of elements in list = 1 Linear list: 1 : h(104)
(hope	! List.Error 2
)	! List.Error 1
(po(doc(pi)()()kap)tee)	Number of elements in list = 13 Linear list: 1 : p(112) 2 : o(111) 3 : d(100) 4 : o(111) 5 : c(99) 6 : p(112) 7 : i(105) 8 : k(107) 9 : a(97) 10 : p(112) 11 : t(116) 12 : e(101) 13 : e(101)
(pet(ri()ci)a)	Number of elements in list = 8 Linear list: 1 : p(112) 2 : e(101) 3 : t(116) 4 : r(114) 5 : i(105) 6 : c(99) 7 : i(105) 8 : a(97)

Выводы.

Был реализован иерархический список и базовые функции для работы с ним. В отчёте представлены результаты тестирования, разработанный код программы представлен в Приложении А.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#INCLUDE <IOSTREAM>
#INCLUDE <FSTREAM>
#INCLUDE <IOMANIP>
#INCLUDE <CSTDLIB>
#INCLUDE <IOS>
#INCLUDE <STREAMBUF>

USING NAMESPACE STD;

TYPEDEF CHAR BASE;   // ÁÀÇÎÂÛÉ ÒÈÏ ÝËÀÌÅÍÒÎÂ (ÀÒÎÌÎÂ)

STRUCT NODE {
     BASE *HD;
     NODE *TL;
     // CONSTRUCTOR
     NODE (){
          HD = NULL;
          TL = NULL;
     }
};

TYPEDEF NODE *LIST;
STRUCT S_EXPR;
STRUCT  TWO_PTR{
     S_EXPR *HD;
     S_EXPR *TL;
} ;  //END TWO_PTR;

STRUCT S_EXPR {
     BOOL TAG; // TRUE: ATOM, FALSE: PAIR
     UNION{
          BASE ATOM;
          TWO_PTR PAIR;
     } NODE;         //END UNION NODE
};             //END S_EXPR

TYPEDEF S_EXPR *LISP;

// ÔÓÍÊÖÈÈ
VOID CREATE_TABS(INT LEVEL);
VOID PRINT_S_EXPR( LISP S );
LISP HEAD (CONST LISP S);
LISP TAIL (CONST LISP S);
LISP CONS (CONST LISP H, CONST LISP T);
LISP MAKE_ATOM (CONST BASE X);
BOOL ISATOM (CONST LISP S);
BOOL ISNULL (CONST LISP S);
VOID DESTROY (LISP S);
BASE GETATOM (CONST LISP S);
LIST CONS (BASE X, LIST T);
INT COUNT_SEQ(CONST LISP X, INT COUNT, INT LEVEL);
INT COUNT_LISP(CONST LISP X, INT COUNT, INT LEVEL);
LIST MAKE_LIST_LISP(CONST LISP X, INT LEVEL, LIST P);
LIST MAKE_LIST_SEQ(CONST LISP X, INT LEVEL, LIST P);
```

```
     // ÔÓÍÊÖÈÈ ÂÂÎÄÀ:
VOID READ_LISP ( LISP& Y);
VOID READ_S_EXPR (BASE PREV, LISP& Y);
VOID READ_SEQ ( LISP& Y);
VOID READ_LISP_FILE (IFSTREAM &FILE, LISP& Y);
VOID READ_S_EXPR_FILE (IFSTREAM &FILE, BASE PREV, LISP& Y);
VOID READ_SEQ_FILE (IFSTREAM &FILE, LISP& Y);



     // ÔÓÍÊÖÈÈ ÂÛÂÎÄÀ:
VOID WRITE_LISP (CONST LISP X);
VOID WRITE_SEQ (CONST LISP X);



VOID CREATE_TABS(INT LEVEL){
     FOR(INT I= 0; I<LEVEL; I++){
          COUT<<"\T";
     }
}

LISP TAIL (CONST LISP S){
     IF (S != NULLPTR){
          IF (!ISATOM(S)){
               RETURN S->NODE.PAIR.TL;
          } ELSE {
               CERR << "ERROR: TAIL(ATOM) \N";
               EXIT(1);
          }
     } ELSE {
          CERR << "ERROR: TAIL(NIL) \N";
          EXIT(1);
     }
}

LISP HEAD (CONST LISP S){
     IF (S != NULLPTR){
          IF (!ISATOM(S)){
               RETURN S->NODE.PAIR.HD;
          } ELSE {
               CERR << "ERROR: HEAD(ATOM) \N";
               EXIT(1);
          }
     } ELSE {
          CERR << "ERROR: HEAD(NULLPTR) \N";
          EXIT(1);
     }
}

BOOL ISATOM (CONST LISP S){
     IF(S == NULLPTR) {
          RETURN FALSE;
     } ELSE {
          RETURN (S -> TAG);
```

```
     }
}

BOOL ISNULL (CONST LISP S){
     RETURN S==NULLPTR;
}

LISP CONS (CONST LISP H, CONST LISP T){
     LISP P;
     IF (ISATOM(T)){
          CERR << "ERROR: TAIL(NIL) \N";
          EXIT(1);
     } ELSE {
          P = NEW S_EXPR;
          IF (P == NULLPTR){
               CERR << "MEMORY NOT ENOUGH\N";
               EXIT(1);
          } ELSE {
               P->TAG = FALSE;
               P->NODE.PAIR.HD = H;
               P->NODE.PAIR.TL = T;
               RETURN P;
          }
     }
}

LISP MAKE_ATOM (CONST BASE X){
     LISP S;
     S = NEW S_EXPR;
     S -> TAG = TRUE;
     S->NODE.ATOM = X;
     RETURN S;
}

VOID DESTROY (LISP S) {
     IF (S != NULLPTR) {
          IF (!ISATOM(S)) {
               DESTROY ( HEAD (S));
               DESTROY ( TAIL(S));
          }
          DELETE S;
     }
}

BASE GETATOM (CONST LISP S){
     IF (!ISATOM(S)){
          CERR << "ERROR: GETATOM(S) FOR !ISATOM(S) \N";
          EXIT(1);
     }ELSE{
          RETURN (S->NODE.ATOM);
     }
}

// ÂÂÎÄ ÑÏÈÑÊÀ Ñ ÊÎÍÑÎËÈ
VOID READ_LISP ( LISP& Y){
     BASE X;
     DO{
          CIN >> X;
```

```
     }WHILE (X==' ');
     READ_S_EXPR ( X, Y);
}

VOID READ_S_EXPR (BASE PREV, LISP& Y){ //PREV - ÐÀÍÅÅ ÏÐÎ÷ÈÒÀÍÍÛÉ
ÑÈÌÂÎË
     IF ( PREV == ')' ) {
          CERR << " ! LIST.ERROR 1 " << ENDL;
          EXIT(1);
     } ELSE IF ( PREV != '(' ) {
          Y = MAKE_ATOM (PREV);
     } ELSE {
          READ_SEQ (Y);
     }
}

VOID READ_SEQ ( LISP& Y) {
     BASE X;
     LISP P1, P2;
     CIN>>NOSKIPWS;
     CIN >> X;
     IF (X == '\N') {
          CERR << " ! LIST.ERROR 2 " << ENDL;
          EXIT(1);
     } ELSE {
          WHILE  ( X==' ' ) {
               CIN >> X;
          }
          IF ( X == ')' ){
               Y = NULLPTR;
          } ELSE {
               READ_S_EXPR ( X, P1);
               READ_SEQ ( P2);
               Y = CONS (P1, P2);
          }
     }
}

VOID READ_LISP_FILE (IFSTREAM &FILE, LISP& Y){
     BASE X;
     DO{
          FILE >> X;
     }WHILE (X==' ');
     READ_S_EXPR_FILE (FILE,X,Y);
}

VOID READ_S_EXPR_FILE (IFSTREAM &FILE, BASE PREV, LISP& Y){ //PREV -
ÐÀÍÅÅ ÏÐÎ÷ÈÒÀÍÍÛÉ ÑÈÌÂÎË
     IF ( PREV == ')' ) {
          CERR << " ! LIST.ERROR 1 " << ENDL;
          EXIT(1);
     } ELSE IF ( PREV != '(' ) {
          Y = MAKE_ATOM (PREV);
     } ELSE {
          READ_SEQ_FILE(FILE,Y);
     }
}
```

```
VOID READ_SEQ_FILE (IFSTREAM &FILE, LISP& Y) {
     BASE X;
     LISP P1, P2;
     IF (!(FILE >> X)) {
          CERR << " ! LIST.ERROR 2 " << ENDL;
          EXIT(1);
     } ELSE {
          WHILE  ( X==' ' ) {
               FILE >> X;
          }
          IF ( X == ')' ){
               Y = NULLPTR;
          } ELSE {
               READ_S_EXPR_FILE (FILE, X, P1);
               READ_SEQ_FILE (FILE, P2);
               Y = CONS (P1, P2);
          }
     }
}

VOID  WRITE_LISP  (CONST  LISP  X){//  ÏÐÎÖÅÄÓÐÀ  ÂÛÂÎÄÀ  ÑÏÈÑÊÀ  Ñ
ÎÁÐÀÌËŸÞÙÈÌÈ ÅÃÎ ÑÊÎÁÊÀÌÈ,
                                        // ÏÓÑÒÎÉ ÑÏÈÑÎÊ ÂÛÂÎÄÈÒÑŸ ÊÀÊ
()
     IF (ISNULL(X)){
          COUT << " ()";
     } ELSE IF (ISATOM(X)){
          COUT << ' ' << X->NODE.ATOM;
     } ELSE  {                    // ÍÅÏÓÑÒÎÉ ÑÏÈÑÎÊ}
          COUT << " (" ;
          WRITE_SEQ(X);
          COUT << " )";
     }
}

VOID  WRITE_SEQ  (CONST  LISP  X)    {//ÏÐÎÖÅÄÓÐÀ  ÂÛÂÎÄÀ  ÑÏÈÑÊÀ  ÁÅÇ
ÎÁÐÀÌËŸÞÙÈÕ ÅÃÎ ÑÊÎÁÎÊ
     IF (!ISNULL(X)) {
          WRITE_LISP(HEAD (X));
          WRITE_SEQ(TAIL (X));
     }
}


VOID CONC2 (LIST &Y, CONST LIST Z){ // ÏÐÎÖÅÄÓÐÀ CONC2 (Y := Y*Z)
     IF (Y==NULLPTR) {
          Y = Z;
     } ELSE {
          CONC2(Y->TL, Z);
     }
}

VOID WRITE_LIST ( LIST S ){
     LIST P = S;
     SHORT I = 0;
     WHILE (P != NULLPTR) {
          I++;
```

```
          COUT << I << " : " << *P->HD << "(" << INT (*P->HD) << ")
"<< ENDL;
          P = P->TL;
     }
}

LIST CONS (BASE X, LIST T){
     LIST P = NEW NODE;
     IF (P == NULLPTR){
          CERR << "MEMORY NOT ENOUGH\N";
          EXIT(1);
     } ELSE {
          P->HD = NEW CHAR;
          *P->HD = X;
          P->TL = T;
          RETURN P;
     }
}

//ÏÎÄÑ÷ŽÒ ÝËÅÌÅÍÒÎÂ ÏÎÑËÄÎÂÀÒÅËÜÍÎÑÒÈ ÑÏÈÑÊÀ Ñ ÎÁÐÀÌËŸÞÙÈÌÈ ÑÊÎÁÊÀÌÈ
INT COUNT_LISP(CONST LISP X, INT COUNT, INT LEVEL){
     LEVEL++;
     IF(ISNULL(X)){
          RETURN COUNT;
     } ELSE IF (ISATOM(X)){
          CREATE_TABS(LEVEL);
          COUT<< X->NODE.ATOM << " - IT'S ATOM" << ENDL;
          COUNT++;
          CREATE_TABS(LEVEL);
          COUT <<"COUNT_1 = "<<COUNT<<ENDL;
     } ELSE {
          CREATE_TABS(LEVEL);
          COUT<< "( - IT'S NOT ATOM" << ENDL;
          COUNT+=COUNT_SEQ(X,COUNT,LEVEL);
          CREATE_TABS(LEVEL);
          COUT <<"COUNT_2 = "<<COUNT<<ENDL;
     }
     CREATE_TABS(LEVEL);
     COUT<< "HEAD SUM = "<<COUNT<<ENDL;
     RETURN COUNT;
}

//ÏÎÄÑ÷ŽÒ ÝËÅÌÅÍÒÎÂ ÏÎÑËÄÎÂÀÒÅËÜÍÎÑÒÈ ÑÏÈÑÊÀ ÁÅÇ ÎÁÐÀÌËŸÞÙÈÕ ÑÊÎÁÎÊ
INT COUNT_SEQ(CONST LISP X, INT COUNT, INT LEVEL){
     LEVEL++;
     IF(!ISNULL(X)){
          INT  HEAD1  =  COUNT_LISP(HEAD(X),  COUNT,  LEVEL);//ÑÓÌÌÀ
ÝËÅÌÅÍÒÎÂ Â ÃÎËÎÂÅ ÑÏÈÑÊÀ
          INT  TAIL1  =  COUNT_SEQ(TAIL(X),  COUNT,  LEVEL);//ÑÓÌÌÀ
ÝËÅÌÅÍÒÎÂ Â ÕÂÎÑÒÅ ÑÏÈÑÊÀ
          COUNT = HEAD1+TAIL1;
          CREATE_TABS(LEVEL);
          COUT<< "EQUAL TO HEAD AND TAIL'S SUM " << ENDL;
          CREATE_TABS(LEVEL);
          COUT<< "HEAD = "<<HEAD<<ENDL;
          CREATE_TABS(LEVEL);
          COUT<< "TAIL = "<<TAIL<<ENDL;
          CREATE_TABS(LEVEL);
```

```
          COUT<< "RESULT SUM = "<<COUNT<<ENDL;
     } ELSE {
          CREATE_TABS(LEVEL);
          COUT<< "TAIL = 0 !!!" << ENDL;
          COUNT = 0;
     }
     RETURN COUNT;
}

//ÑÎÇÄÀÍÈÅ  ËÈÍÅÉÍÎÃÎ  ÑÏÈÑÊÀ  ÈÇ  ÝËÅÌÅÍÒÎÂ  ÑÏÈÑÊÀ  Ñ  ÎÁÐÀÌËŸÞÙÈÌÈ
ÑÊÎÁÊÀÌÈ
LIST MAKE_LIST_LISP(CONST LISP X, INT LEVEL, LIST P){
     LEVEL++;
     IF(ISNULL(X)){
          RETURN P;
     } ELSE IF (ISATOM(X)){
          P = CONS(X->NODE.ATOM, P);
          CREATE_TABS(LEVEL);
          WRITE_LIST(P);
     } ELSE {
          CREATE_TABS(LEVEL);
          COUT<< "( - IT'S NOT ATOM" << ENDL;
          CONC2(P, MAKE_LIST_SEQ(X, LEVEL, P));
     }
     RETURN P;
}

//  ÑÎÇÄÀÍÈÅ  ËÈÍÅÉÍÎÃÎ  ÑÏÈÑÊÀ  ÈÇ  ÝËÅÌÅÍÒÎÂ  ÑÏÈÑÊÀ  ÁÅÇ  ÎÁÐÀÌËŸÞÙÈÕ
ÑÊÎÁÎÊ
LIST MAKE_LIST_SEQ(CONST LISP X, INT LEVEL, LIST P){
     LEVEL++;
     IF(!ISNULL(X)){
          CREATE_TABS(LEVEL);
          COUT<<"LIST CONSIST OF HEAD AND TAIL ELEMENTS"<<ENDL;
          LIST HEAD1 = MAKE_LIST_LISP(HEAD(X), LEVEL, P);
          LIST TAIL1 = MAKE_LIST_SEQ(TAIL(X), LEVEL, P);
          CONC2(HEAD1, TAIL1);
          CREATE_TABS(LEVEL);
          WRITE_LIST(HEAD1);
          RETURN HEAD1;
     }
     CREATE_TABS(LEVEL);
     COUT<<"NO ELEMENTS IN TAIL!"<<ENDL;
     RETURN NULLPTR;
}

INT MAIN(){
     INT WAY;
     COUT<<"1 - CONSOLE\N2 - FILE\NYOUR CHOICE: "<<ENDL;
     CIN>>WAY;
     LISP LISP;
     IF(WAY == 1){
          COUT<<"ENTER YOUR LIST :"<<ENDL;
          READ_LISP(LISP);
     } ELSE IF (WAY == 2){
          STRING FILENAME;
          COUT<<"ENTER FILENAME:"<<ENDL;
          CIN>>FILENAME;
```

```
            IFSTREAM FILE (FILENAME);
            IF(!FILE){
                 CERR<<"FILE CAN'T BE OPENED!"<<ENDL;
                 EXIT(0);
            }
            CHAR LETTER;
            FILE>>LETTER;
            FILE.SEEKG(0,IOS::BEG);
            STRING LINE;
            GETLINE(FILE, LINE);
            COUT<<"FILE CONTENT : "<< LINE<<ENDL;
            FILE.SEEKG(0,IOS::BEG);
            READ_LISP_FILE(FILE, LISP);
            FILE.CLOSE();
            COUT<<"ENTERED LIST :"<<ENDL;
            WRITE_LISP(LISP);
            COUT<<ENDL;
      } ELSE {
            CERR<<"WRONG WAY! TRY AGAIN!"<<ENDL;
            EXIT(0);
      }
      COUT<<ENDL;
      INT COUNTELEM = 0;
      INT RECLEVEL = 0;
      LIST LISTPTR = NULLPTR;
      COUNTELEM = COUNT_LISP(LISP,COUNTELEM,RECLEVEL);
      LISTPTR = MAKE_LIST_LISP(LISP, RECLEVEL, LISTPTR);
      COUT<<"NUMBER OF ELEMENTS IN LIST = "<<COUNTELEM<<ENDL;
      COUT<<"LINEAR LIST: ";
      WRITE_LIST(LISTPTR);
      COUT<<ENDL;
      RETURN 0;
}
```