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: Иерархические списки

Студент гр. 8304 _____ Ивченко А.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

Цель работы.

Ознакомиться с нелинейными конструкциями обработки информации, изучить иерархические структуры данных на языке C/C++.

Задание(вариант 7).

Подсчитать число атомов в иерархическом списке; сформировать линейный список атомов, соответствующий порядку подсчёта;

Описание работы программы.

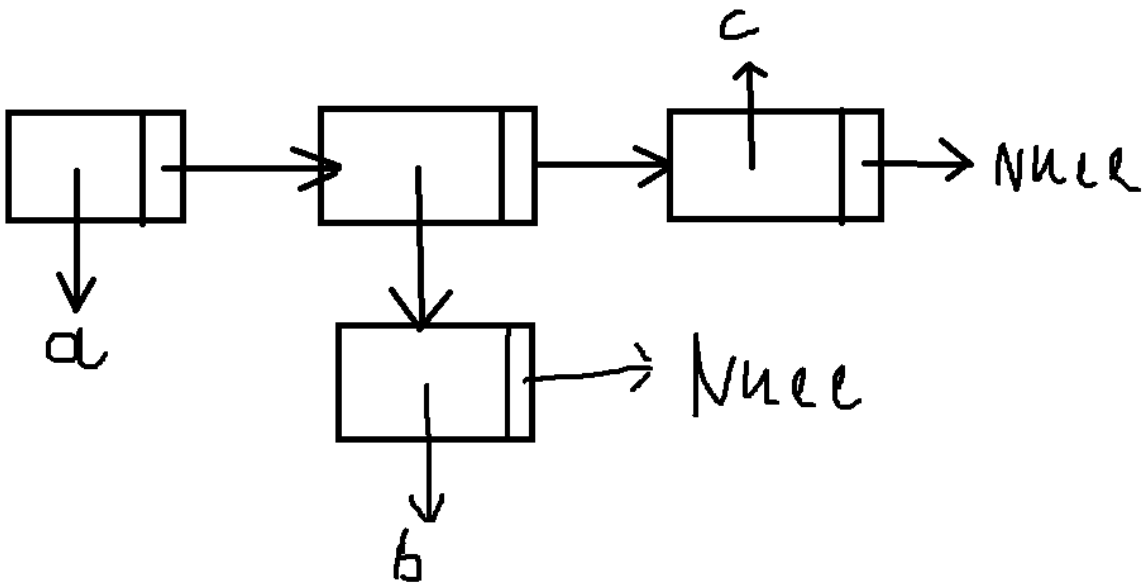
Создается структура s-выражения, которое включает в себя поля: тэг, определящий атомарность узла, и сам узел, содержащий указатель на следующий элемент списка, указатель на хвост, а также сам элемент(атом). Определяются следующие функции класса s_expr, где hlist — иерархический список или указатель на s-выражение:

- *HLIST TAIL(CONST HLIST S) - функция, возвращающая указатель на указатель на следующий элемент.*

  HLIST HEAD (CONST HLIST S) - функция, вовзращающая указатель на вложенный список.
- *HLIST CONS (CONST HLIST H, CONST HLIST T) — конструктор иерархического списка, создает экземпляр класса s_expr.*
- *HLIST MAKE_ATOM (CONST BASE X) — функция создания атома*
- *BOOL ISATOM (CONST HLIST S) — функция проверяет атомарность узла*
- *BOOL ISNULL (CONST HLIST S) — функция проверяет ненулевой список или нет.*
- *VOID DESTROY (HLIST S) — рекурсивная функция, удаляющая все элементы списка.*

- *VOID READ_HLIST (HLIST& Y, STD::STRINGSTREAM& S, INT *);VOID READ_S_EXPR (BASE PREV, HLIST& Y, STD::STRINGSTREAM& S, INT *); VOID READ_SEQ (HLIST& Y, STD::STRINGSTREAM& S, INT *) - функции, выполняющие считывание строки из потока данных, которые также подсчитывают число атомов в иерархическом списке и выводят на экран линейный список атомов, соответствующий порядку подсчёта.*

Графическое представление иерархического списка.



(a(b)c)

Тестирование.

```
Иерархический список: (a(b(c)))
Линейный список атомов:abc
Количество атомов в иерархическом списке:3
test№2
Иерархический список:(a(b))
Линейный список атомов:ab
Количество атомов в иерархическом списке:2
test№3
Иерархический список:((c))
Линейный список атомов:c
Количество атомов в иерархическом списке:1
test№4
Иерархический список:((((cd)
Линейный список атомов:cd ! List.Error 3
test№5
Иерархический список:(a(b(cdef)sfd(eww)))
Линейный список атомов:abcdefsfdeww
Количество атомов в иерархическом списке:12
test№6
Иерархический список:(d(ds(()()()d)))
Линейный список атомов:ddsd
Количество атомов в иерархическом списке:4
test№7
Иерархический список:(((SDF)SDF))
Линейный список атомов:SDFSDF
Количество атомов в иерархическом списке:6
test№8
Иерархический список:(a(ab(cd())(d()(fgs)())))
Линейный список атомов:aabcddfgs
Количество атомов в иерархическом списке:9
test№9
Иерархический список:(a(b(((C)))defsgs()))
Линейный список атомов:abCdefsgs
Количество атомов в иерархическом списке:9
test№10
Иерархический список:)a(bc(((de))er)()df))
Линейный список атомов: ! List.Error 1
```

Выводы.

В результате лабораторной работы был изучен принцип работы иерархических списков и s-выражений, а также были получены навыки по представлению данных конструкций на языке C++.

ИСХОДНЫЙ КОД

```
#INCLUDE <IOSTREAM>
#INCLUDE <CSTDLIB>
#INCLUDE <SSTREAM>

TYPEDEF CHAR BASE;

        STRUCT S_EXPR {
                BOOL TAG;
                UNION{
                        BASE ATOM;
                        S_EXPR *NEXT;
                        S_EXPR *LIST;
                } NODE;
        };

        TYPEDEF S_EXPR *HLIST;

        VOID PRINT_S_EXPR( HLIST S );

        HLIST HEAD (CONST HLIST S);
        HLIST TAIL (CONST HLIST S);
        HLIST CONS (CONST HLIST H, CONST HLIST T);
        HLIST MAKE_ATOM (CONST BASE X);
        BOOL ISATOM (CONST HLIST S);
        BOOL ISNULL (CONST HLIST S);
        VOID DESTROY (HLIST S);

        BASE GETATOM (CONST HLIST S);



        VOID READ_HLIST ( HLIST& Y, STD::STRINGSTREAM& S, INT *);
        VOID READ_S_EXPR (BASE PREV, HLIST& Y, STD::STRINGSTREAM& S, INT *);
```

```
        VOID READ_SEQ ( HLIST& Y, STD::STRINGSTREAM& S, INT *);


HLIST HEAD (CONST HLIST S){
                IF (S != NULL) {
                        IF (!ISATOM(S))         RETURN S->NODE.NEXT;
                        ELSE { STD::CERR << "ERROR: HEAD(ATOM) \N"; EXIT(1); }
                }ELSE {
                        STD::CERR << "ERROR: HEAD(NIL) \N";
                        EXIT(1);
                }
}

BOOL ISATOM (CONST HLIST S){
        IF(S == NULL) RETURN FALSE;
        ELSE RETURN (S -> TAG);
}

BOOL ISNULL (CONST HLIST S){
        RETURN S == NULL;

}

HLIST TAIL (CONST HLIST S)
{
                IF (S != NULL) {
                        IF (!ISATOM(S))         RETURN S->NODE.LIST;
                        ELSE { STD::CERR << "ERROR: TAIL(ATOM) \N"; EXIT(1); }
                }ELSE { STD::CERR << "ERROR: TAIL(NIL) \N";
                        EXIT(1);
                }
}

HLIST CONS (CONST HLIST H, CONST HLIST T)

        {HLIST P;
        IF (ISATOM(T)) {
                STD::CERR << "ERROR: CONS(*, ATOM)\N";
                EXIT(1);
        }
```

```
        ELSE {
                P = NEW S_EXPR;
                IF ( P == NULL)         {
                        STD::CERR << "MEMORY NOT ENOUGH\N";
                        EXIT(1);
                }
                ELSE {
                        P->TAG = FALSE;
                        P->NODE.NEXT = H;
                        P->NODE.LIST = T;
                        RETURN P;
                }
        }
}

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



VOID DESTROY (HLIST S)
{
        IF ( S != NULL) {
                IF (!ISATOM(S)) {
                        DESTROY ( HEAD (S));
                        DESTROY ( TAIL (S));
                }
                DELETE S;

        };
        }

BASE GETATOM (CONST HLIST S)
        {
                IF (!ISATOM(S)) {
                        STD::CERR << "ERROR: GETATOM(S) FOR !ISATOM(S) \N";
```

```
                EXIT(1);
        }
        ELSE RETURN (S->NODE.ATOM);
}



VOID READ_HLIST ( HLIST& Y,  STD::STRINGSTREAM& S, INT *C)
{       BASE X;
        DO S >> X;
        WHILE (X == ' ');
        IF (X != '(')    {
                STD::CERR << " ! LIST.ERROR 1 " << STD::ENDL;
                EXIT(1);
        }
        READ_S_EXPR ( X, Y, S, C);


}
VOID READ_S_EXPR (BASE PREV, HLIST& Y,  STD::STRINGSTREAM& S, INT *C)
{
        IF ( PREV == ')' ) {
                STD::CERR << " ! LIST.ERROR 2 " << STD::ENDL;
                EXIT(1);
        }
        ELSE IF ( PREV != '(' ) {
                Y = MAKE_ATOM (PREV);
                STD::COUT<<PREV;
                (*C)++;
        }
        ELSE READ_SEQ (Y, S, C);
}
VOID READ_SEQ ( HLIST& Y,  STD::STRINGSTREAM& S, INT *C) {
        BASE X;
        HLIST P1, P2;


        IF (!(S >> X)) {
                STD::CERR << " ! LIST.ERROR 3 " << STD::ENDL;
                EXIT(1);
        }
        ELSE{
```

```
                WHILE (X == ' ')
                        S >> X;



                IF ( X == ')' ) Y = NULLPTR;
                ELSE {
                        READ_S_EXPR ( X, P1, S, C);
                        READ_SEQ ( P2, S, C);
                        Y = CONS (P1, P2);
                }
        }
}

INT MAIN(INT ARGC, CHAR* ARGV[]){
        IF (ARGC == 2){
                STD::STRINGSTREAM STR;
                STR << ARGV[1];
                STD::COUT << "ИЕРАРХИЧЕСКИЙ СПИСОК:" << ARGV[1] << STD::ENDL;
                INT C = 0;
                S_EXPR* HL;
                STD::COUT << "ЛИНЕЙНЫЙ СПИСОК АТОМОВ:";
                READ_HLIST(HL, STR, &C);
                STD::COUT << STD::ENDL;
                STD::COUT <<"КОЛИЧЕСТВО АТОМОВ В ИЕРАРХИЧЕСКОМ СПИСКЕ:"<< C <<
STD::ENDL;
                DESTROY(HL);


        }
        RETURN 0;

}
```