ММИНОБРНАУКИ 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: Рекурсивная обработка иерархических списков.

Студент гр. 9384 _____ Мосин К.К.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

Создать иерархический список. Проверить созданный ранее список на синтаксическую корректность.

Задание.

ВАРИАНТ 11.

Проверить синтаксическую корректность арифметического выражения, в которое могут входить любые 2 из операций +, -, *, /.

Выполнение работы.

Обработка арифметического выражения предполагается в префиксном или постфиксном варианте. При запуске программы пользователю предложено ввести вариант обработки. Выполняется проверка на количество операций функцией

bool OprList(const list x),

при успехе которой сразу проверяется синтаксис выражения функцией

bool isCorrectList(const list x, const int op).

Концепция проста - устраивают положения операция_переменная или переменная_переменная с учетом присутствия операции. Если пользователь выбрал постфиксную запись, достаточно перевернуть иерархический лист для прогона функции проверки на префиксную запись. Этим занимаются

list reverse(const list s)

list rev(const list s, const list z)

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

Входные данные	Выходные данные
1 ()	() Flase
1 (+a(*abc)d(*ef))	(+ a (* a b c) d (* e f)) True
2 (a(bc/)d+(ef*))	(a (b c /) d + (e f *)) False

Выводы.

В ходе выполнения лабораторный работы был создан иерархический список. Также были использованы алгоритмы для обхода иерархического списка и его обработка.

ПРИЛОЖЕНИЕ
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>

#include "list.h"

using namespace std;
using namespace h_list;

bool isCorrectList(const list x, const int op = 0);
bool isOperation(const base x);
bool OprList(const list x);
void OprSeq(const list x);
list reverse(const list s);
list rev(const list s, const list z);

int main(){
   list s;
   int a;
   cout << "Type 1 for prefix form\n";
   cout << "Type 2 for postfix form\n";
   cout << "Command: ";
   cin >> a;
   if(a != 1 && a != 2){
      cout << "Error: type 1 or 2\n";
      return -1;
   }
   cout << "Type list: ";
   read_list(s);
   write_list(s);
   if(a == 2)
      s = reverse(s);
   if(OprList(s) && isCorrectList(s))
      cout << " True" << endl;
   else
      cout << " False" << endl;
   destroy(s);
   return 0;
}

bool isCorrectList(const list x, const int op){
   if(!isNull(x) && !isAtom(head(x)))
      return isCorrectList(head(x), 0);
   else if(!isNull(x) && !isNull(tail(x)) && isOperation(head(x)->node.atom) && !isOperation(head(tail(x))-
>node.atom))
      return isCorrectList(tail(x), 1);
   else if(!isNull(x) && !isNull(tail(x)) && !isOperation(head(x)->node.atom) && !isOperation(head(tail(x))-
>node.atom))
      return isCorrectList(tail(x), op);
   else if(!isNull(x) && isNull(tail(x)) && op && !isOperation(x->node.atom))
      return true;
   else{
      return false;
   }
}
```

```
bool isOperation(const base x){
   if(x == '+' || x == '-' || x == '*' || x == '/')
      return true;
   return false;
}

bool OprList(const list x){
   static bool add, sub, mul, div = false;
   if(isNull(x))
      return false;
   else if(isAtom(x)){
      if(x->node.atom == '+') add = true;
      else if(x->node.atom == '-') sub = true;
      else if(x->node.atom == '*') mul = true;
      else if(x->node.atom == '/') div = true;
   }
   else
      OprSeq(x);

   if((add && sub && mul) || (add && sub && div) || (add && mul && div) || (sub && mul && div))
      return false;
   return true;
}

void OprSeq(const list x){
   if(!isNull(x)){
      OprList(head(x));
      OprSeq(tail(x));
   }
}

list reverse(const list s){
   return (rev(s, NULL));
}

list rev(const list s, const list z){
   if(isNull(s))
      return (z);
   else if(isAtom(head(s)))
      return (rev(tail(s), cons(head(s), z)));
   else
      return (rev(tail(s), cons(rev(head(s), NULL), z)));
}
```

Название файла: list.h

```
namespace h_list{
   typedef char base;

   struct exp;
   struct ptrs{
      exp *head;
      exp *tail;
   };
   struct exp{
      bool tag;
      union{
         base atom;
         ptrs pair;
```

```
        }node;
    };

    typedef exp *list;

    list head(const list s);
    list tail(const list s);
    list cons(const list h, const list t);
    list makeAtom(const base x);
    bool isAtom(const list s);
    bool isNull(const list s);
    void destroy(list s);

    void read_list(list& y);
    void read_exp(base prev, list& y);
    void read_seq(list& y);
    void write_list(const list x);
    void write_seq(const list x);
}
```

Название файла: list.cpp

```
#include <iostream>
#include <cstdlib>

using namespace std;

#include "list.h"

namespace h_list{
    list head(const list s){
        if(s){
            if(!isAtom(s)){
                return s->node.pair.head;
            }
            else{
                cerr << "Error: Head(atom)\n";
                exit(1);
            }
        }
        else{
            cerr << "Error: Heid(NULL)\n";
            exit(1);
        }
    }

    list tail(const list s){
        if(s){
            if(!isAtom(s)){
                return s->node.pair.tail;
            }
            else{
                cerr << "Error: Tail(atom)\n";
                exit(1);
            }
        }
        else{
            cerr << "Error: Tail(NULL)\n";
            exit(1);
```

```
    }
}

list cons(const list h, const list t){
    list p;
    if (isAtom(t)){
        cerr << "Error: Cons(*, atom)\n";
        exit(1);
    }
    else{
        p = new exp;
        if (p == NULL){
            cerr << "Error: Memory not enough\n";
            exit(1);
        }
        else{
            p->tag = false;
            p->node.pair.head = h;
            p->node.pair.tail = t;
            return p;
        }
    }
}

list make_atom(const base x){
    list s = new exp;
    s->tag = true;
    s->node.atom = x;
    return s;
}

bool isAtom(const list s){
    if (s == NULL)
        return false;
    else
        return (s->tag);
}

bool isNull(const list s){
    return s == NULL;
}

void destroy(list s){
    if(s){
        if(!isAtom(s)){
            destroy(head(s));
            destroy(tail(s));
        }
        delete s;
    }
}

void read_list(list& y){
    base x;
    do
        cin >> x;
    while(x == ' ');
    read_exp(x, y);
```

```
    }

    void read_exp(base prev, list& y){
        if(prev == ')'){
            cerr << " ! List.Error 1 " << endl;
            exit(1);
        }
        else if(prev != '(')
            y = make_atom(prev);
        else
            read_seq(y);
    }

    void read_seq(list& y){
        base x;
        list p1, p2;

        if(!(cin >> x)){
            cerr << " ! List.Error 2 " << endl;
            exit(1);
        }
        else{
            while(x == ' ')
                cin >> x;
            if(x == ')')
                y = NULL;
            else{
                read_exp(x, p1);
                read_seq(p2);
                y = cons(p1, p2);
            }
        }
    }

    void write_list(const list x){
        if(isNull(x))
            cout << " ()";
        else if(isAtom(x))
            cout << ' ' << x->node.atom;
        else{
            cout << " (";
            write_seq(x);
            cout << " )";
        }
    }

    void write_seq(const list x){
        if (!isNull(x)){
            write_list(head(x));
            write_seq(tail(x));
        }
    }
}
```