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: Рекурсивная обработка иерархических списков

Студент гр. 7383 _____ Тян Е.

Преподаватель _____ Размочаева Н. В.

Санкт-Петербург
2018

СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ

Цель работы: ознакомиться с одной из часто используемых на практике нелинейных конструкций, способами её организации и рекурсивной обработки. Получить навыки решения задач обработки иерархических списков с использованием базовых функций их рекурсивной обработки.

Формулировка задачи: реализовать проверку синтаксической корректности арифметического выражения, представленного в виде иерархического списка, в постфиксной форме, и простое деление на 0. В выражение входят константы и переменные, которые являются атомами списка. Операции представляются как: <аргументы> <операция>. Аргументов может быть 1, 2 и более.

Входные данные: арифметическое выражение.

2. РЕАЛИЗАЦИЯ ЗАДАЧИ

В данной работе используется главная функция (main()) и дополнительные функции (void errors(int err), lisp cons(const lisp h, const lisp t), lisp make_char_atom(const base x), int read_s_expr(string array, int* ptr), void read_lisp(string array, lisp* head, int* ptr), bool write_lisp(lisp head, int i, int j, bool* flag), bool write(lisp head, int i, int j, bool* flag), void destroy(lisp s)).

Параметры используемые в функции void read_lisp(string array, lisp* head, int* ptr):

- array - строка с введенными данными;
- head – голова списка;
- ptr – указатель на данный элемент в строке данных.

Параметры используемые в функции int read_s_expr(string array, int* ptr):

- array - строка с введенными данными;
- ptr – указатель на данный элемент в строке данных.

Параметры используемые в функции lisp make_char_atom(const base x):

- x – значение, которое необходимо сделать атомом списка.

Параметры используемые в функции lisp cons(const lisp h, const lisp t):

- h – голова списка;
- t – хвост списка.

Параметры используемые в функции bool write_lisp(lisp head, int i, int j, bool* flag):

- head – голова списка;
- i – счетчик для более удобного вывода головы списка;
- j – счетчик для более удобного вывода хвоста списка;
- flag – переменная, меняющая значение в зависимости от того, встретилось деление на нуль или нет.

Параметры используемые в функции bool write(lisp head, int i, int j, bool* flag):

- head – голова списка;

• i – счетчик для более удобного вывода головы списка;

• j – счетчик для более удобного вывода хвоста списка;

• flag – переменная, меняющая значение в зависимости от того, встретилось деление на нуль или нет.

Параметры используемые в функции void errors(int err):

• err – номер ошибки.

Параметры используемые в функции void destroy(lisp s):

• s – элемент, который необходимо удалить.

В функции main выводится меню на консоль, где можно выбрать число, соответствующее выполняемой операции. Считывается целое число и, при помощи switch(), выбирается необходимая опция. При выборе «1», пользователь вводит выражение. При выборе «2», программа считывает выражение из файла «test1.txt». При выборе «3» программа завершает работу. При выборе другого значения, программа выводит сообщение: «Проверьте введенные данные и повторите попытку» и ожидает дальнейших действий. Программа завершает свою работу только при выборе «3», в противном случае - программа будет ожидать дальнейших указаний. Считанные данные из файла как и данные с консоли записываются в строку. После считывания данных с файла как и с консоли вызывается функция void read_lisp(string array, lisp* head, int* ptr), которая начинает «проходить» по элементам строки, при просмотре данных строки вызывается функция int read_s_expr(string array, int* ptr), которая проверяет, чем является данный элемент, функция int read_s_expr(string array, int* ptr) проверяет каждое значение строки array до пробельного символа или пока не дойдет до конца строки. Если данный элемент является константой или переменной – функция возвращает значение символа, если знак операции – значение -2, в противном случае функция возвращает значение -1. Функция void read_lisp(string array, lisp* head, int* ptr) просматривает строку на три символа вперед, если они существуют. Таким образом, если первый элемент – цифра или константа, то второй должен быть цифрой или константой, а третий знаком операции, т.к.

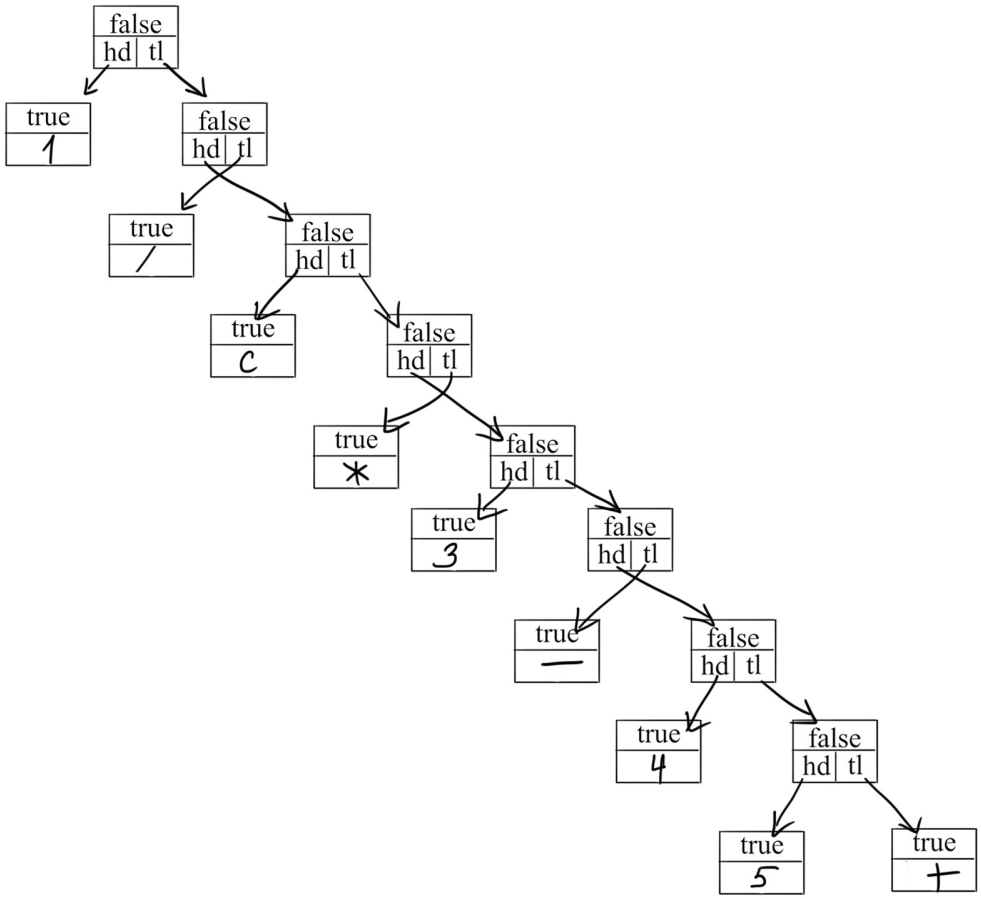
любое постфиксное выражение можно можно рассматривать как выражение приведенное на рис. 1.

Рисунок 1 – Рекрсивное представление постфиксного выражения

Если в ходе прохода по строке данных возникает ошибка, то следует обращение к функции void errors(int err), и на консоль выводится сообщение об ошибке в связи с видом ошибки. Если все выполняется корректно, то программа вызывает функцию make_char_atom(const base x) для создания атома списка, далее для двух атомов списка вызывается функция lisp cons(const lisp h, const lisp t) для их «объединения». Если не были выявлены ошибки во время создания иерархического списка, то вызывается функция bool write(lisp head, int i, int j, bool* flag), которая в свою очередь вызывает функцию bool write_lisp(lisp head, int i, int j, bool* flag) для вывода на консоль самого списка и проверки простого деления на нуль. Если в ходе работы программы обнаруживается операция деления на нуль, то программа выдает ошибку и прекращает работу. Для более наглядного представления иерархического списка при выводе используются отступы разной длины. После успешного выполнения программы вызывается функция void destroy(lisp s) для удаления списка.

Для более понятного описания реализации задачи рассмотрим два случая: введем корректное выражение и выражение с делением на нуль.

В первом случае: 1 c 3 4 5 + - * /, иерархический список данного выражения представлен на рис. 2.



Можно заметить, что выражение «4 5 +» само по себе представляет аргумент, следовательно, как было изложено ранее теперь рассматривается выражение 1 с 3 <аргумент> - * / и т.д.

Во втором случае: а 0 /. При записи списка не будет выявлено ошибок, но при прохождению по списку будет выявлено деление на нуль. И

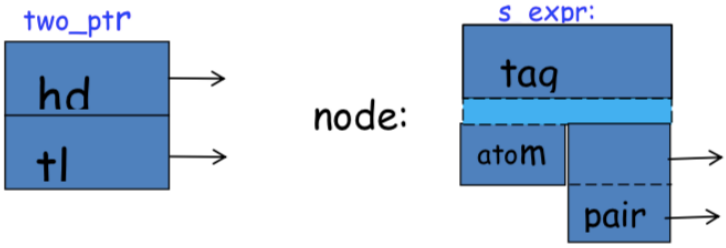


Рисунок 3 – Второй случай

программа выведет ошибку.

Не стоит забывать, что в программе используется структура, приведенная на рис. 3, т.е. при вводе числа, а не цифры, будет выведена ошибка, т.к. цифра – тоже значение символьного типа.

3. ТЕСТИРОВАНИЕ

Программа была собрана в компиляторе GCC в OS Linux Ubuntu 12.04. Программа может быть скомпилирована с помощью команды:

g++ <имя_файла>.cpp

Тестовые случаи представлены в Приложении А.

Исходя из тестовых случаев можно увидеть, что в одиннадцатом тесте программа ведет себя неверно, поэтому была исправлена программа: добавлена проверка на корректность введенных данных – если после последнего знака операции есть символы, то программа должна выводить ошибку. Вывод исправленной программы указан в тестовых случаях №12, 13. После, тестовые случаи не выявили неверного поведения программы, что говорит о том, что по результатам тестирования было показано: поставленная задача была выполнена.

4. ВЫВОД

В ходе работы ознакомились с одной из часто используемых на практике нелинейных конструкций, способами её организации и рекурсивной обработки. Были получены навыки решения задач обработки иерархических списков на языке C++. Была реализована программа выполняющая синтаксическую проверку корректности арифметического выражения, представленного в постфиксной форме, и простое деление на 0.

Была использована работа с файлами. Разработано меню циклического взаимодействия с пользователем.

ПРИЛОЖЕНИЕ А. ТЕСТОВЫЕ СЛУЧАИ

№	Ввод	Вывод
1	1 2 3 +	2 3 + Корректный ввод.
2	2	1 2 - Корректный ввод.
3	1 1 2 3 4 5 - + /	Неверный ввод.
4	1	Неверный ввод.
5	1 2 1 03 wrong 30 9438 7923 in put	Неверный ввод.
6	1 s f +	s f + Корректный ввод.
7	1 1 a -	1 a - Корректный ввод.
8	1 a 5 *	a 5 * Корректный ввод.

9	1 1 0 /	1 0 Деление на нуль.
10	1 0 1 /	0 1 / Корректный ввод.
11	1 1 2 - 9	1 2 - Корректный ввод.
12	1 1 2 - 339 dje 9j id	Неверный ввод.
13	1 - - - - - 9	Неверный ввод.

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД ПРОГРАММЫ

main.cpp:

```
#include <iostream>
#include <cctype>
#include <cstdlib>
#include <cstring>
#include <fstream>
#include <cstring>

using namespace std;

typedef char base;
struct s_expr;
struct two_ptr{
    s_expr *hd;
    s_expr* tl;
};
struct s_expr{
    bool tag;
    union{
        base atom;
        two_ptr pair;
    }node;
};
typedef s_expr* lisp;

void errors(int err) {
    switch(err) {
        case 1:
            cerr<<"Неверный ввод."<<endl;
            break;
        case 2:
            cerr<<"Деление на нуль."<<endl;
            break;
        case 3:
           cerr<<"Не хвататет значений, констант или знаков. Проверьте введенные данные и
повторите попытку."<<endl;
            break;
        case 4:
            cerr << "Неверный ввод."<<endl;
            break;
        default:
            break;
    }
}

lisp cons (const lisp h, const lisp t){
    lisp p;
    p = new s_expr;
    if ( p == NULL) {
        errors(4);
        return NULL;
```

```
    }else {
            p->tag = false;
            p->node.pair.hd = h;
            p->node.pair.tl = t;
            return p;
        }
}

lisp make_char_atom (const base x) {
    if(x=='+' || x=='-' || x=='*' || x=='/' || isalpha(x) || isdigit(x)){
        lisp s;
        s = new s_expr;
        s -> tag = true;
        s->node.atom = x;
        return s;
    }else{
        errors(3);
        return NULL;
    }
}

int read_s_expr(string array,int*ptr);

void read_lisp(string array,lisp* head,int* ptr){
    int num=0;
    num=read_s_expr(array,ptr);
    if(num!=-1 && num!=-2){
        lisp p11;
        p11=make_char_atom(array[(*ptr)-2]);
        int num2,num3;
        num2=read_s_expr(array,ptr);
        num3=read_s_expr(array,ptr);
        lisp p12;
        if(num3==-1 || num2==-1){
            errors(1);
            return;
        }
        if(num3!=-1 && num3!=-2){
            lisp p1,p2;
            (*ptr)=(*ptr)-4;
            read_lisp(array,&p1,ptr);
            read_lisp(array,&p2,ptr);
            p12=cons(p1,p2);
        }
        if(num3==-2){
            if(num2!=-2 && num2!=-1){
                lisp p1,p2;
                p1=make_char_atom(array[(*ptr)-4]);
                p2=make_char_atom(array[(*ptr)-2]);
                p12=cons(p1,p2);
            }
        }
        (*head)=cons(p11,p12);
    }else if(num==-2 && (*ptr)!= 0){
```

```
        (*head)=make_char_atom(array[(*ptr)-2]);
    }else{
        errors(1);
        return ;
    }
}

int read_s_expr(string array,int* ptr){
    int num=0;
    while(!isspace(array[(*ptr)]) && (*ptr)< array.length()){
        (*ptr)++;
    }
    (*ptr)++;
    if((*ptr) > 2 && isspace(array[(*ptr)-3])){
        if(isdigit(array[(*ptr)-2]) || isalpha(array[(*ptr)-2]))
            return array[(*ptr)-2];
        else if(array[(*ptr)-2] == '+' || array[(*ptr)-2]=='-' || array[(*ptr)-2]=='*' || array[(*ptr)-2]=='/')
            return -2;
        else
            return -1;
    }else if((*ptr) <= 2){
        if(isdigit(array[(*ptr)-2]) || isalpha(array[(*ptr)-2]))
            return array[(*ptr)-2];
        else if(array[(*ptr)-2] == '+' || array[(*ptr)-2]=='-' || array[(*ptr)-2]=='*' || array[(*ptr)-2]=='/')
            return -2;
        else
            return -1;
    }else
        return -1;
}

bool write_lisp(lisp head, int i,int j,bool* flag);

bool write(lisp head,int i,int j,bool* flag){
    if(head==NULL){
        cerr<< "Неверный ввод."<<endl;
        return false;
    }else if(head->tag==true){
        cerr<<"Не хвататет значений, констант или знаков. Проверьте введенные данные и
повторите попытку."<<endl;
        return false;
    }else{
        i++;
        write_lisp(head->node.pair.hd,i,j,flag);
        j++;
        write_lisp(head->node.pair.tl,i,j,flag);
        return true;
    }
}

bool write_lisp(lisp head, int i,int j,bool* flag){
    if(head==NULL){
        cerr<<"Не хвататет значений, констант или знаков. Проверьте введенные данные и
повторите попытку."<<endl;
```

```
        return false;
    }
    if(head->tag==true){
        for(int j=0;j< i;j++)
            cout<<'\t';
        if(head->node.atom=='+' || head->node.atom=='-' || head->node.atom=='*' || head->node.atom=='/' ||
(isalpha(head->node.atom))){
            if(head->node.atom=='/' && (*flag)==true){
                errors(2);
                return false;
            }
            cout<<head->node.atom<<endl;
        }else{
            cout<<head->node.atom<<endl;
            if(head->node.atom == '0')
                (*flag)=true;
            else
                (*flag)=false;
        }
        return true;
    }else{
        write(head,i,j,flag);
    }
}

void destroy (lisp s) {
    if ( s != NULL) {
        if (s->tag == false){
            destroy ( s->node.pair.hd);
            destroy ( s->node.pair.tl);
        }
        delete s;
    };
}

int main(){
    int num=0;
    while(num != 3){
        cout << "Выберите дальнейшие действия и введите цифру:"<<endl;
        cout << "1. Ввести выражение вручную."<<endl;
        cout << "2. Считать выражение из файла test1.txt."<<endl;
        cout << "3. Завершить работу."<<endl;
        cin >> num;
        switch(num){
            case 1:{
                getchar();
                string array;
                getline(cin,array);
                lisp head=NULL;
                bool flag;
                flag=false;
                int ptr=0,i=0,j=0;
                read_lisp(array,&head,&ptr);
                if(head!=NULL && ptr==array.length()+1)
```

```
            if(write(head,i,j,&flag) && flag==false)
                cout<< "Корректный ввод."<<endl;
            if(head!=NULL && ptr< array.length()+1)
                cerr<<"Неверный ввод."<<endl;;
            destroy(head);
            break;
        }
        case 2:{
            string array;
            ifstream infile("test1.txt");
            getline(infile,array);
            array=array+'\n';
            lisp head=NULL;
            bool flag;
            int ptr=0,i=0,j=0;
            read_lisp(array,&head,&ptr);
            if(head==NULL)
                cerr<<"Файл пуст."<<endl;
            if(write(head,i,j,&flag) && flag==false)
                        cout<< "Корректный ввод."<<endl;
            destroy(head);
            break;
        }
        case 3:
            return 0;
        default:
            cerr << "Проверьте введенные данные и повторите попытку." << endl;
            break;
      }
   }
   return 0;
}
```