**Отчет по лабораторной работе №24** по курсу Алгоритмы и структуры данных

Студент группы М8О-107Б-2022 Калуцкий Максим Витальевич, № по списку 13

Контакты e-mail max.kalutskij17@gmail.com

Работа выполнена: «14» .05.2023г.

Преподаватель: Аносова Наталья Павловна каф. 806

Входной контроль знаний с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отчет сдан « » \_\_\_\_\_\_\_\_\_202 \_\_ г., итоговая оценка \_\_\_\_\_

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. **Тема:** Динамические структуры данных. Обработка деревьев.

1. **Цель работы:**  Составить программу выполнения заданных преобразований арифметических выражений с применением деревьев.
2. **Задание:** Упростить выражения, убрав из него все произведения, в которых в качестве сомножителя используется нуль.
3. **Оборудование** (лабораторное):

*Оборудование ПЭВМ студента, если использовалось:*

Процессор Intel Core i3-4005U CPU с ОП 8096 Мб, НМД \_ 131072\_ Мб. Монитор lenovo

Другие устройства \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Программное обеспечение (лабораторное):**

*Программное обеспечение ЭВМ студента, если использовалось:*

Операционная система семейства \_\_\_Unix\_\_\_\_, наименование \_\_ Ubuntu \_\_\_\_\_\_\_\_\_\_\_\_ версия 4.4.2

интерпретатор команд \_\_\_\_bash\_\_\_\_\_\_\_\_ версия \_\_5.0.17\_\_\_\_.

Система программирования \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_версия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Редактор текстов \_\_\_\_\_nano\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ версия \_25.2.2 \_\_\_\_\_\_\_\_\_

Утилиты операционной системы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Прикладные системы и программы **\_\_\_**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Местонахождение и имена файлов программ и данных на домашнем компьютере \_\_\_\_\_home/jekich228\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Идея, метод, алгоритм** решения задачи(в формах:словесной,псевдокода,графической[блок-схема,диаграмма,рисунок,таблица] или формальные спецификации с пред- и постусловиями)

Для выполнения поставленной задачи была создана структура узла дерева, в которой находится поле значения в виде строки, поле с указателем на правого ребенка и поле с указателем на левого ребенка.

При считывании выражения из терминала программа разбивает введенное выражение на элементы, каждый из которых является либо оператором, либо операндом. Далее по считанному выражению строится постфиксная запись и по ней строится дерево выражения, после чего срабатывает блок программы, который выполняет задание моего варианта – рекурсивно пробегается вглубь дерева и ищет поддеревья, в которых есть умножение на нуль. Такие поддеревья обрезаются и дерево приводится к стандартному виду, далее по упрощенному дереву строится постфиксная запись выражения, а уже потом по ней строится обычная, инфиксная запись.

1. **Сценарий выполнения работы** [план работы,первоначальный текст программы в черновике(можно на отдельном листе)итесты либо соображения по тестированию].

#include <bits/stdc++.h>

#include <Windows.h>

using namespace std;

// structure for creating tree nodes

struct elem\_tree{

    string value;

    elem\_tree \* right;

    elem\_tree \* left;

};

queue<string> infz;  //queue for infix entry

queue<string> postfz;  //queue for postfix recording

stack<string> st;   //auxiliary stack for postfix record formation

stack<elem\_tree\*> pst; // stack for tree formation

map<string,int>q = {{"(", 0}, {")", 1}, {"^", 2}, {"\*", 3}, {"/", 3}, {"+", 4}, {"-", 4}}; //dictionary for setting the operation priority

queue<string> pz;  //a queue for a postfix entry but already for a simplified expression

// a function that creates a new tree node

elem\_tree\* new\_node(string value){

    elem\_tree \* node = new elem\_tree;

    node->value = value;

    node->right = NULL;

    node->left = NULL;

    return node;

}

// // draw our tree

void printTree(elem\_tree\* root, int indent = 0) {

    if (root != NULL) {

        printTree(root->right, indent + 4);

        cout << string(indent, ' ') << root->value << endl;

        printTree(root->left, indent + 4);

    }

}

// a function that determines whether a string is a numberчиѝлом

bool is\_digit(string a){

    if(a == "-") return 0;

    else{

        if(a[0]=='-') a = a.substr(1,a.size()-1);

    for(char i:a) if(!isdigit(i))return 0;

    }

    return 1;

}

//  a function that converts an infix form to a postfix form, it is stored in a postfix queue

void create\_que(string s){

    string t = "";

    for(char i:s){

        if(i==' ') continue;

        else{

            if(isdigit(i))t+=string(1,i);

            else{

                if(i=='-' and t.size()==0){

                    if((!infz.empty()) and (is\_digit(infz.back()) or infz.back() ==")")){

                        infz.push(string(1,i));

                    }else t = string(1,i);

                }

                else if(i=='-' and t.size()!=0){

                    infz.push(t);

                    infz.push(string(1,i));

                    t = "";

                }

                else{

                    if(t.size()!=0){

                        infz.push(t);

                        infz.push(string(1,i));

                        t = "";

                    }

                    else infz.push(string(1,i));

                }

                }

        }

        }

        if(t.size()!=0)infz.push(t);

        cout<<"The read expression has the form:"<<endl;

        int x = infz.size();

        for(int i = 0;i<x;++i){

            cout<<infz.front()<<" ";

            infz.push(infz.front());

            infz.pop();

        }

        for(int i = 0;i<3;++i)cout<<endl;

}

// a function that translates an infix form into a postfix form, it is stored in a postfix queue

void pstrec(){

    while(!infz.empty()){

        string s = infz.front();

        infz.pop();

        if(is\_digit(s)) postfz.push(s);

        else{

            if(s=="(") st.push(s);

            else if(s==")"){

                while(st.top()!="("){

                    postfz.push(st.top());

                    st.pop();

                }

                st.pop();

            }

            else{

                if(st.empty()) {

                   st.push(s);

                }

                else{

                    while((!st.empty()) and ((q[st.top()] <= q[s]) and q[st.top()] != 0)){

                        postfz.push(st.top());

                        st.pop();

                    }

                    st.push(s);

                }

            }

        }

    }

    if(!st.empty()){

        while(!st.empty()){

            postfz.push(st.top());

            st.pop();

        }

    }

    cout<<"The read expression has the form in POSTFIX NOTATION:"<<endl;

    int x = postfz.size();

        for(int i = 0;i<x;++i){

            cout<<postfz.front()<<" ";

            postfz.push(postfz.front());

            postfz.pop();

        }

        for(int i = 0;i<3;++i)cout<<endl;

}

 // a function that forms a tree from a postfix entry, that is, from the infz queue

elem\_tree\* create\_free(){

    while(!postfz.empty()){

        string t = postfz.front();

        postfz.pop();

        if(is\_digit(t)){

           pst.push(new\_node(t));

        }

        else{

            elem\_tree\* keke = new\_node(t);

            keke->right = pst.top();

            pst.pop();

            keke->left = pst.top();

            pst.pop();

            pst.push(keke);

        }

    }

    return pst.top();

}

// this function recursively traverses the tree in width and replaces integer expressions that include multiplication by zero element by zero

int simplify(elem\_tree\* &head, int a = 0){

    if(head ->left == NULL and head ->right == NULL) return stoi(head->value);

    if(a == 1){

        if(head->value == "+") return simplify(head->left,1)+simplify(head->right,1);

        else if(head->value == "-") return simplify(head->left,1)-simplify(head->right,1);

        else if(head->value == "^") return simplify(head->left,1)^simplify(head->right,1);

        else if(head->value == "/") return simplify(head->left,1)/simplify(head->right,1);

   }

   if(head -> value == "\*"){

        if(simplify(head->left,1)==0 or simplify(head->right,1)==0){

            if(head->left != NULL) delete head->left;

            if(head->right != NULL) delete head->right;

            head->value = "0";

            head->right = NULL;

            head->left = NULL;

            return 0;

        }

        else {

            return 1;

        }

   }

   else{

        simplify(head->left,0);

        simplify(head->right,0);

        return 0;

   }

}

//a function that builds a postfix entry on a tree

void expr\_from\_tree(elem\_tree\* root) {

    if (root == nullptr) {

        return;

    }

     pz.push(root->value);

    expr\_from\_tree(root->right);

    expr\_from\_tree(root->left);

    }

int main(){

    SetConsoleCP(CP\_UTF8);

    SetConsoleOutputCP(CP\_UTF8);

    string a;

    getline(cin,a); //reading an expression into a string

    create\_que(a); //I put the expression in the lol queue and immediately output it

    pstrec(); //I create a queue with a postfix entry and immediately output it

    elem\_tree\* head = create\_free(); // creating a tree

    cout<<"Expression tree before simplification:"<<endl;

    printTree(head); // bringing out a tree

    for(int i = 0;i<3;++i)cout<<endl;

    simplify(head); //simplifying the expression through a tree

    cout<<"Expression tree after simplification:"<<endl;

    printTree(head); // output the tree of the modified expression

   for(int i = 0;i<3;++i)cout<<endl;

    expr\_from\_tree(head);

    string s = "";

    int x = pz.size();

        for(int i = 0;i<x;++i){

            s = " "+pz.front()+s;

                // output a postfix entry of a simplified expression

            pz.push(pz.front());

            pz.pop();

        }

    cout<<"SIMPLIFIED postfix entry:"<<endl;

    cout<<s<<endl;

    for(int i = 0;i<3;++i)cout<<endl;

    stack<string> z1;

    stack<string> z2;

    while(!pz.empty()){

        z1.push(pz.front());

        pz.pop();

    }

    while(!z1.empty()){

        string h = z1.top();

        z1.pop();

        if(is\_digit(h)){

            z2.push(h);

        }

        else{

            string s1,s2;

            s1 = z2.top();

            z2.pop();

            s2 = z2.top();

            z2.pop();

            z2.push("("+s2+h+s1+")");

        }

    }

    cout<<"A normalized (infix) entry of A SIMPLIFIED FORM:"<<endl;

    cout<<z2.top();

}

1. **Распечатка протокола** (подклеить листинг окончательного варианта программы с тестовыми примерами,подписанныйпреподавателем).

PS C:\Users\heleu\Desktop\laba\_2\_sem\_kek> & 'c:\Users\heleu\.vscode\extensions\ms-vscode.cpptools-1.15.4-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-crtl10kv.5yz' '--stdout=Microsoft-MIEngine-Out-qs5zfr2g.4te' '--stderr=Microsoft-MIEngine-Error-keqsb1vk.lpa' '--pid=Microsoft-MIEngine-Pid-fqag03vc.u1e' '--dbgExe=C:\MinGW\bin\gdb.exe' '--interpreter=mi'

221+(32\*3)-32+32\*(54-(27+27))

The read expression has the form:

221 + ( 32 \* 3 ) - 32 + 32 \* ( 54 - ( 27 + 27 ) )

The read expression has the form in POSTFIX NOTATION:

221 32 3 \* + 32 - 32 54 27 27 + - \* +

Expression tree before simplification:

27

+

27

-

54

\*

32

+

32

-

3

\*

32

+

221

Expression tree after simplification:

0

+

32

-

3

\*

32

+

221

SIMPLIFIED postfix entry:

221 32 3 \* + 32 - 0 +

A normalized (infix) entry of A SIMPLIFIED FORM:

(((221+(32\*3))-32)+0)

PS C:\Users\heleu\Desktop\laba\_2\_sem\_kek>

1. **Дневник отладки** должен содержать дату и время сеансов отладки, и основные события(ошибки в сценарии и программе,нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Лаб. | Дата | Время | Событие | Действие по исправлению | Примечание |
|  | или |  |  |  |  |  |
|  | дом. |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. **Замечания автора** по существу работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Выводы**: Проделав данную работу, я научился составлять программы выполнения заданных преобразований арифметических выражений с применением деревьев.

Недочёты при выполнении задания могут быть устранены следующим образом: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_