|  |
| --- |
| Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования |
| Московский Государственный Технический Университет имени Н.Э. Баумана |
|  |

Факультет              ФУНДАМЕНТАЛЬНЫЕ НАУКИ

Кафедра             МАТЕМАТИКА И МАТЕМАТИЧЕСКАЯ ФИЗИКА

**Отчет по домашнему заданию № 2.3**

**по курсу «Информатика»**

Студента           Борисов Иван Дмитриевич ­­­­­­­­­­­

(фамилия, имя, отчество)

Группа                                          ФН11-22Б

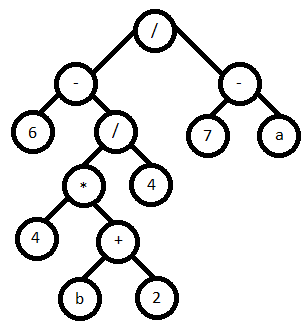
Преподаватель \_   доцент, к.т.н. Ничушкина Т. Н.

Должность, ФИО, подпись

2019

**Использовать бинарное дерево для вычисления выражения (6-4\*(b+2)/4)/(7-a).**

**Графическое изображение дерева**

****

**Количество узлов:13; Количество листов:7; Глубина дерева:6**

**Дерево-это структура данных, представляющая собой совокупность элементов и отношений, образующий иерархическую структуру этих элементов.**

**Потомки – все вершины, в которые входят ветви, исходящие из одной общей вершины, а сама вершина для них-предком.**

**Высота(глубина) дерева определяется количеством уровней, на которых располагаются его вершины. Высота пустого дерева 0, высота из одного корня 1.**

**Степень вершины-количество дуг , исходящих из нее.   
 Степень дерева – максимальная степень вершины (двоичные деревья-степень не более 2; сильноветвящаяся-степень произвольная).  
 Бинарное дерево-дерево, каждая вершина которого имеет не более 2 потомков.**

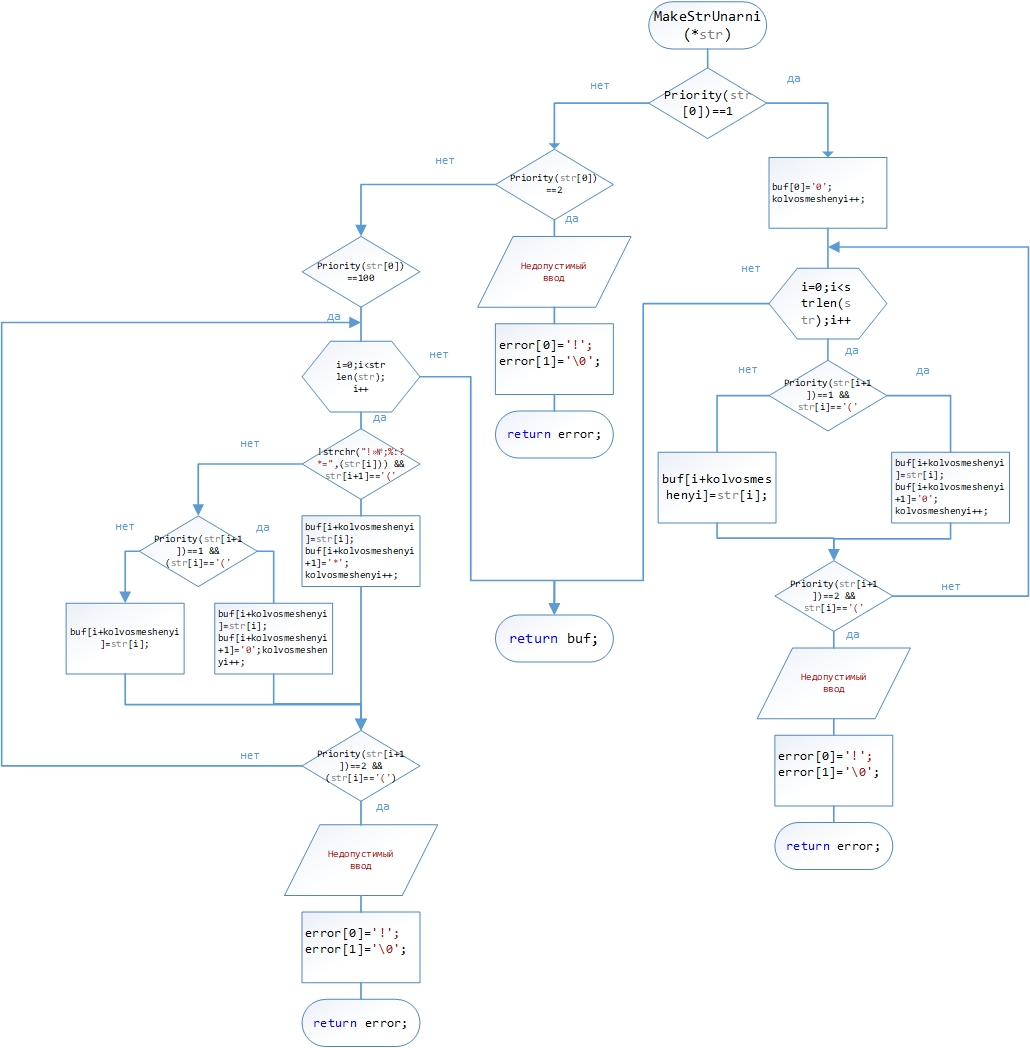
**В общем случае на К-ом уровне может быть до 2^(k-1) вершин.**

**Бинарные деревья делятся по степени вершин на: строгие (у листьев 0, а у узлов 2), нестрогие (у листьев 0, а у узлов 1 или 2), полные (только полностью заполненные уровни), неполные(содержит не полностью заполненные уровни).**

**Сбалансированное дерево(полное)-длины всех путей от корня к внешним вершинам равны между собой.  
 Почти сбалансированное-длины всевозможных путей от корня к внешним вершинам отличаются не более, чем на единицу.**

**Скорость поиска О(n) = C \* log2(n) (в худшем O(n)=n)**

**Схема алгоритма**

****

**Код программы**

#include "stdafx.h"

#include <conio.h>

#include <locale.h>

#include <ctime>

#include < string.h >

#include <ctype.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

#include <sstream>

#include <chrono>

using namespace std;

struct element // тип элемента

{

char value[80]; // информационное поле 1

element \*Left; // адресное поле ≪левый≫

element \*Right; // адресное поле ≪правый≫

};

void AddElement(element \*\*MyTree, element \*TreeElement) //рекурсивная функция создания/добавления вершины дерева

{

element \* current;

current = \*MyTree; //вспомогательный указатель на указатель дерева

if (current == NULL)

\*MyTree = TreeElement; //если узла нет, создаем узел

else

//если значение текущего узла поддерева меньше значения узла дерева,

if (strcmp(TreeElement->value,current->value)>0)

//то зависываем значение узла в левое поддерево

AddElement(&current->Left, TreeElement);

//иначе зависываем значение узла в правое поддерево

else

AddElement(&current->Right, TreeElement);

}

void PreOrder\_ShowTree(element \*MyTree) //рекурсивная функция печати бинарного дерева

{

if (MyTree != NULL) //пока не встретится пустой узел

{

printf("%4s", MyTree->value); //печать узла дерева

PreOrder\_ShowTree (MyTree->Left); //рекурсивный вызов для левого поддерева

PreOrder\_ShowTree (MyTree->Right); //рекурсивный вызов для правого поддерева

}

}

void SymmetricOrder\_ShowTree(element \*MyTree) //рекурсивная функция печати бинарного дерева

{

if (MyTree != NULL) //пока не встретится пустой узел

{

SymmetricOrder\_ShowTree (MyTree->Left); //рекурсивный вызов для левого поддерева

printf("%4s", MyTree->value); //печать узла дерева

SymmetricOrder\_ShowTree (MyTree->Right); //рекурсивный вызов для правого поддерева

}

}

void PostOrder\_ShowTree(element \*MyTree) //рекурсивная функция печати бинарного дерева

{

if (MyTree != NULL) //пока не встретится пустой узел

{

PostOrder\_ShowTree (MyTree->Left); //рекурсивный вызов для левого поддерева

PostOrder\_ShowTree (MyTree->Right); //рекурсивный вызов для правого поддерева

printf("%4s", MyTree->value); //печать узла дерева

}

}

void RemoveTree(element\*\*MyTree) // очистка памяти

{

element \*current;

current = \*MyTree;

if (\*MyTree != NULL){

RemoveTree(&(\*MyTree)->Left);

RemoveTree(&(\*MyTree)->Right);

delete \*MyTree;

}

}

int Priority(char c) //1-унарные 2-не унарные 3-не операция

{

switch(c)

{

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

}

return 100;//это не операция,пропускаем ее

}

element \*MakeTree (char \*str,int first,int last)

{

char c;

int MinPtr,i,k,ptr;

int nest=0;

char \* Term = new char [100];

element \*Tree = new element;

int key=0;

if (first==last)

{

Term[0]=str[first];

Term[1]='\0';

strcpy(Tree->value,Term);

Tree->Left=NULL;

Tree->Right=NULL;

return Tree;

}

for (int i=first;i<last+1 && key==0;i++)

if (strchr("\*+/-()",str[i]))

key=1;

if (key==0)

{

for (int i=first;i<last+1;i++)

{

Term[i-first]=str[i];

Term[i-first+1]='\0';

}

strcpy(Tree->value,Term);

Tree->Left=NULL;

Tree->Right=NULL;

return Tree;

}

MinPtr = 100;

for (i=first;i<=last;i++)

{

c=str[i];

if (c=='(')

{

nest ++;

continue;

}

if (c==')')

{

nest --;

continue;

}

if (nest>0)

continue;

ptr = Priority(c);

if (ptr<=MinPtr)// ищем последнюю операцию с наименьшим приоритетом (<=)

{

MinPtr=ptr;

k=i;

}

}

//все выражение взято в скобки

if (MinPtr == 100 && str[first]=='(' && str[last]==')')

return MakeTree(str,first+1,last-1);

Term[0]=str[k];

Term[1]='\0';

strcpy(Tree->value,Term);

Tree->Left=MakeTree(str,first,k-1);

Tree->Right=MakeTree(str,k+1,last);

return Tree;

}

int IsNumber(element \*Tree)

{

int i = 0;

if (!Tree)

return 0; //пустое дерево

while (Tree->value[i]) // пока не дошли до конца строки

if (!strchr("-.,0123456789",Tree->value[i++]))

return 0;//если нашли не цифру-выход

return 1;

}

void Calculate(element \*Tree)

{

double a,b,c=0;

//проверка: можно ли вычислить

if (!Tree || !IsNumber(Tree->Left) || !IsNumber(Tree->Right))

return ;

a=atof(Tree->Left->value);//получить данные от сыновей

b=atof(Tree->Right->value);

switch (Tree->value[0])

{

case'+':

c=a+b;

break;

case '-':

c=a-b;

break;

case '\*':

c=a\*b;

break;

case '/':

if (b!=0)

{

c=a/b;

break;

}

else

{

cout<<"Деление на ноль невозможно! Перезапустите программу!"<<endl;

system("pause");

exit(0);

}

}

//удалить ненужные поддеревья

delete Tree->Left;

delete Tree->Right;

//обновить вершину

std::ostringstream strs;

strs << c;

std::string str = strs.str();

sprintf\_s((Tree)->value, 20, "%f", c);

Tree->Left=NULL;

Tree->Right=NULL;

}

void extrCalculate(element \*\*MyTree)

{

element \*current;

current = \*MyTree;

if (\*MyTree != NULL){

extrCalculate(&(\*MyTree)->Left);

extrCalculate(&(\*MyTree)->Right);

Calculate(\*MyTree);

}

}

char \*MakeStrUnarni(char \*str)

{

int i=0;

int kolvosmeshenyi=0;

char \* buf = new char [150];

char \* error = new char [10];

if (Priority(str[0])==1)

{

buf[0]='0';

kolvosmeshenyi++;

for (i=0;i<strlen(str);i++)

{

if(Priority(str[i+1])==1 && (str[i]=='('))

{

buf[i+kolvosmeshenyi]=str[i];

buf[i+kolvosmeshenyi+1]='0';

kolvosmeshenyi++;

}

else

buf[i+kolvosmeshenyi]=str[i];

if(Priority(str[i+1])==2 && (str[i]=='('))//если это недопустимый ввод /\*

{

cout<<"Недопустимый ввод (не существует унарного умножения и деления)"<<endl;

error[0]='!';

error[1]='\0';

return error;

}

}

}

else

{

if(Priority(str[0])==2)//если это недопустимый ввод /\*

{

cout<<"Недопустимый ввод (не существует унарного умножения и деления)"<<endl;

error[0]='!';

error[1]='\0';

return error;

}

else

{

if(Priority(str[0])==100)

{

for (i=0;i<strlen(str);i++)

{

if(strchr("1234567890qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM)",(str[i])) && str[i+1]=='(')//2(2+2)

{

buf[i+kolvosmeshenyi]=str[i];

buf[i+kolvosmeshenyi+1]='\*';

kolvosmeshenyi++;

}

else

if(Priority(str[i+1])==1 && (str[i]=='('))

{

buf[i+kolvosmeshenyi]=str[i];

buf[i+kolvosmeshenyi+1]='0';

kolvosmeshenyi++;

}

else

buf[i+kolvosmeshenyi]=str[i];

if(Priority(str[i+1])==2 && (str[i]=='('))//если это недопустимый ввод

{

cout<<"Недопустимый ввод (не существует унарного умножения и деления)"<<endl;

error[0]='!';

error[1]='\0';

return error;

}

}

}

}

}

int kolvoskobok=0;

for (i=0;i<strlen(str);i++)

{

if (str[i]=='(')

{

kolvoskobok++;

}

if (str[i]==')')

{

kolvoskobok--;

}

if (str[i]=='/' && str[i+1]=='0')

{

cout<<"Деление на 0 невозможно!"<<endl;

error[0]='!';

error[1]='\0';

return error;

}

}

if (kolvoskobok!=0)

{

cout<<"Количество скобок не равно"<<endl;

error[0]='!';

error[1]='\0';

return error;

}

buf[strlen(str)+kolvosmeshenyi]='\0';

return buf;

}

int CheckStr(char \*str)

{

for (int i=0;i<strlen(str);i++)

{

if ( !strchr("\*/+()1234567890QWERTYUIOPASDFGHJKLZXCVBNMqwertyuiopasdfghjklzxcvbnm",str[i]))

{

cout<<"Вы ввели строку неправильно,повторите ввод без использования !@#$%^&№;%:=?"<<endl;

return 0;

}

}

return 1;

}

void ZamenaPeremennix(element \*\*MyTree,char \*str,char \*number)

{

char \* buf=new char [80];

element \*current;

current = \*MyTree;

if (\*MyTree != NULL)

{

ZamenaPeremennix(&(\*MyTree)->Left,str,number);

ZamenaPeremennix(&(\*MyTree)->Right,str,number);

if (strcmp((\*MyTree)->value,str)==0)

strcpy((\*MyTree)->value,number);

}

}

void NalichiePeremennix(element \*\*MyTree,int &key)

{

char \* buf=new char [80];

element \*current;

current = \*MyTree;

if (\*MyTree != NULL)

{

NalichiePeremennix(&(\*MyTree)->Left,key);

NalichiePeremennix(&(\*MyTree)->Right,key);

if (!strchr("\*+/-()0123456789",(\*MyTree)->value[0]))

key=1;

}

}

void KolvoPeremennix(element \*\*MyTree,char \*str,int &kolvo)

{

char \* buf=new char [80];

element \*current;

current = \*MyTree;

if (\*MyTree != NULL)

{

KolvoPeremennix(&(\*MyTree)->Left,str,kolvo);

KolvoPeremennix(&(\*MyTree)->Right,str,kolvo);

if (strcmp((\*MyTree)->value,str)==0)

kolvo++;

}

}

void PoiskPeremennix(element \*\*MyTree,int \*kluch,char \*str)

{

element \*current;

current = \*MyTree;

if (\*MyTree != NULL)

{

PoiskPeremennix(&(\*MyTree)->Left,kluch,str);

if (strchr("QWERTYUIOPASDFGHJKLZXCVBNMqwertyuiopasdfghjklzxcvbnm",(\*MyTree)->value[0]))

strcpy(str,(\*MyTree)->value);

else

{

PoiskPeremennix(&(\*MyTree)->Right,kluch,str);

if (strchr("QWERTYUIOPASDFGHJKLZXCVBNMqwertyuiopasdfghjklzxcvbnm",(\*MyTree)->value[0]))

strcpy(str,(\*MyTree)->value);

}

if(\*kluch==0)

{

if (!strchr("+\*-/()1234567890",(\*MyTree)->value[0])) {

(\*kluch)++;

strcpy(str,(\*MyTree)->value);

}

else

strcpy(str,"!");

}

}

}

int main(int argc, char\* argv[])

{

char \*vvod\_polzovatelya, \*ptr, \*buf, \*proverka;

vvod\_polzovatelya=new char [100];

srand(time(NULL));

int Kate=2;

char ZnachenyePeremennoi[100];

buf = new char [100];

proverka = new char [150];

setlocale(0,"russian");

char key;

cout << "Ввести свою строку - 1; Ввести строку варианта - 2" << endl;

cin>>key;

switch (key)

{

case '1':

cout<<"введите строку"<<endl;;

cin>>vvod\_polzovatelya;

break;

case '2':

strcpy(vvod\_polzovatelya,"(6-4\*(b+2)/4)/(7-a)");

break;

default:

cout << "Вы ввели недопустимое значение,повторите ввод.Ввести свою строку - 1; Ввести строку варианта - 2" << endl;

cin>>key;

break;

}

while (CheckStr(vvod\_polzovatelya)==0 || strcmp(strcpy(proverka,MakeStrUnarni(vvod\_polzovatelya)),"!")==0)

{

cout<<"Повторите ввод"<<endl;

cin>>vvod\_polzovatelya;

}

strcpy(vvod\_polzovatelya,MakeStrUnarni(vvod\_polzovatelya));

cout<<"исходная строка"<<endl;

cout<<vvod\_polzovatelya<<endl;

ptr=vvod\_polzovatelya;

element \* MyTree = MakeTree(vvod\_polzovatelya,0,strlen(vvod\_polzovatelya)-1);

cout<<"Сформированное дерево:"<<endl;

PreOrder\_ShowTree(MyTree);

cout<<" "<<endl;

SymmetricOrder\_ShowTree(MyTree);

cout<<" "<<endl;

PostOrder\_ShowTree(MyTree);

int kluch=0,kolvo=0;

int key2;

while (Kate!=0)

{

NalichiePeremennix(&MyTree,Kate);

PoiskPeremennix(&MyTree,&kluch,buf);

if(!strchr("!",buf[0]))

{

if (Kate!=0 && Kate!=2)

{

cout<<"\nВыберите ввод переменной:самостоятельно - 1; рандомное заполнение - 2"<<endl;

cin>>key2;

switch (key2)

{

case 1:

PoiskPeremennix(&MyTree,&kluch,buf);

if(!strchr("!",buf[0]))

{

cout<<"Введите значение переменной "<<buf<<endl;

cin>>ZnachenyePeremennoi;

KolvoPeremennix(&MyTree,buf,kolvo);

ZamenaPeremennix(&MyTree,buf,ZnachenyePeremennoi);

kolvo=0;

kluch=0;

}

else

{

Kate=0;

}

break;

case 2:

PoiskPeremennix(&MyTree,&kluch,buf);

if(!strchr("!",buf[0]))

{

itoa((rand() % (201) -100),ZnachenyePeremennoi,10);

cout<<"значение переменной "<<buf<<" = "<<ZnachenyePeremennoi<<endl;

KolvoPeremennix(&MyTree,buf,kolvo);

ZamenaPeremennix(&MyTree,buf,ZnachenyePeremennoi);

kolvo=0;

kluch=0;

}

else

{

Kate=0;

}

break;

}

}

else

Kate=0;

}

else

{

Kate=0;

}

}

extrCalculate(&MyTree);

cout<<"\nРезультат вычисления выражения: "<< MyTree->value<<endl;

cout<<" "<<endl;

RemoveTree(&MyTree);

system("pause");

return 0;

} **Таблица тестовых вычислений**

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Вывод программы** |
| **(6-4\*(b+2)/4)/(7-a)** | **~ -1** |  |
| **2-2/3(-2+2)** | **2** |  |
| **14-2/(a-2)-2** | **Ошибка деление на 0** |  |

**Вывод**

Я улучшил свои навыки работы с бинарными деревьями и строками.