МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

КАФЕДРА МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Бинарное дерево

Студент гр. 7383 _____ Ласковенко Е.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

Оглавление

Цель работы

Познакомиться со структурой и реализацией бинарного дерева, получить навыки программирования функций, использующих бинарное дерево, на языке программирования C++.

Задача

Требуется:

а) для заданной формулы f построить дерево-формулу t;

б) для заданного дерева-формулы t напечатать соответствующую формулу f;

в) с помощью построения дерева-формулы t преобразовать заданную формулу f из префиксной формы в инфиксную;

г) упростить дерево-формулу t, заменяя в нем все поддеревья, соответствующие формулам (f + 0), (0 + f), (f - 0), (f * 1), (1 * f), на поддеревья, соответствующие формуле f, а поддеревья, соответствующие формулам (f * 0) и (0 * f), на узел с 0;

Реализация задачи

Описание функций, классов и переменных. Описание работы алгоритма

Функция **int main()**:

Является головной функцией, в которой происходит считывание данных из файла (название файла вводит пользователь в консоли). Если файла с введенным названием нет или же файл невозможно открыть на чтение, то в консоль выводится ошибка, а программа снова требует ввести название файла на открытие. Далее объявляются необходимые переменные, описанные ниже, происходит вывод на экран содержимого файла. Если файл пуст, то выводится соответствующее предупреждение, если файл не пустой, то происходит вызов функций построения бинарного дерева, вывода дерева в консоль, перевода из префиксной формы выражения в инфиксную, упрощения выражения. Все вызовы находятся в цикле, выходом из которого является условие ввода пользователем пустой строки в консоли. Также в функции реализована обработка исключений — проверка на валидность входных данных, проверка наличия файла с данными и прочее.

Объявляемые переменные:

- string filename — строковая переменная, хранящая имя файла, с которого необходимо считать данные.
- string temp_str — строковая переменная, хранящая входные данные.
- filebuf file — файловый буфер, используемый при считывании входных данных с файла.
- stringbuf str_buf — строковый буфер, используемый при считывании входных данных с консоли.
- istream is_str — входной поток, в который помещаются входные данные.
- auto size — переменная, хранящая размер (количество символов) файла с входными данными.

Функция **Bool isSign(char ch):**

Функция, принимающая на вход символ и проверяющая, является ли входной символ знаком операции.

Функция **Bool isTerm(char ch):**

Функция, принимающая на вход символ и проверяющая, является ли входной символ термом в соответствии с заданием.

Функция **Void buildBT_RLR(istream& is_str, BinTree* root=new BinTree, bool flag=false)**:

Рекурсивная функция, принимающая на вход входной поток, с которого происходит считывание символа. Если символ — знак операции, то значению текущего узла присваивается этот символ, затем происходит два рекурсивных вызова, в которые подаются адреса левого и правого листа узла. Если символ — терма, тогда значению текущего узла присваивается этот символ и текущий вызов завершается. В функции происходит проверка на валидность входных данных с помощью исключений.

Функция **void printBT(BinTree* root, unsigned int i=0):**

Рекурсивная функция, принимающая на вход корень бинарного дерева. Выводит на экран представление бинарного дерева, проходя по всем узлам. Обход — ПКЛ. В каждой строке выводятся знаки табуляций, количество которых равняется уровню узла.

Функция **void prefix_to_infix(BinTree* root, bool rt_flag=false):**

Рекурсивная функция, принимающая на вход корень бинарного дерева. Выводит на экран выражение в инфиксной форме, проходя по всем узлам бинарного дерева. Обход — ЛКП.

Функция **void fixBT(BinTree* root, BinTree* pv_rt=nullptr):**

Рекурсивная функция, принимающая на вход корень бинарного дерева. Изменяет бинарное дерево в соответствии с задачей. Происходят рекурсивные вызовы до самого левого и нижнего узла бинарного дерева, затем до самого правого нижнего. Далее на каждом вызове происходит проверка значения узла, и, при необходимости, дерево перестраивается так, как обозначено в задании.

Класс **class BinTree:**

Класс бинарного дерева. Состоит из следующих полей: значения узла, указателей на левый и правый узел. Для класса были реализованы следующие методы:

- BinTree() - конструктор класса. Инициализирует все поля нулевыми значениями.
- Base RootBT() - функция, возвращающая значение текущего узла.
- BinTree* Left () - функция, возвращающая адрес левого узла.
- BinTree* Right () - функция, возвращающая адрес правого узла.
- void ConsBT(const base x, BinTree* lst=new BinTree, BinTree* rst=new BinTree) - функция, инициализирующая поля объекта класса значениями входных аргументов.
- ~BinTree() - декструктор класса. Рекурсивно освобождает память.

Тестирование программы

Процесс тестирования

Программа собрана в операционной системе Linux Mint 19, с использованием компилятора G++. В других ОС и компиляторах тестирование не проводилось.

Результаты тестирования

Тестовые случаи представлены в Приложении А.

По результатам тестирования было показано, что поставленная задача была выполнена.

Выводы

В ходе лабораторной работы были получены навыки работы с бинарным деревом. В работе был реализован класс, который представляет собой бинарное дерево на базе ссылок. Бинарное дерево удобно использовать, при использовании рекурсивных функций.

ПРИЛОЖЕНИЕ А.

Тестовые случаи

Входные данные	Вывод	Верно?
**+a1+0c-e0	File contains: **+a1+0c-e0 Infix form: ((a+1)*(0+c)*(e-0))	Да
*-+abcd	Infix form: (((a+b)-c)*d)	Да
+a*c+de	Infix form: a+c*(d+e)	Да

ПРИЛОЖЕНИЕ Б.

Код программы

1. Код **main.cpp:**

```
#include <iostream>
#include <istream>
#include <fstream>
#include <sstream>
#include <string>
#include <exception>
#include "btree.h"
#include "api.h"
#include "myexception.h"

using namespace std;

int main()
{
    string filename;
    string temp_str;
    filebuf file;
    stringbuf str_buf;
    istream is_str(&str_buf);

    int int_t;
    string t_str;
    char temp_ch;
    long size;

    while(true)
    {
        try
        {
            cout << "Enter \"1\" to input data from file." << endl
                 << "Enter \"2\" to input data from console."<< endl
                 << "Press ENTER to exit." << endl;
            getline(cin, t_str);
            if(t_str.empty())
                break;
            int_t = stoi(t_str);
            t_str.clear();

            switch(int_t)
            {
            case 0:
                break;
            case 1:
                cout << "Enter a name of the data-file:" << endl;
                getline(cin, filename);
                if(filename.empty())
                    throw new my_ex("Empty filename.");

                if(!file.open(filename, ios::in))
                {
                    file.close();
                    throw new my_ex("Incorrect filename.");
                }

                size = file.in_avail();
                cout << "File contains: ";
```

```
                for(auto c_size = 0; c_size<size; c_size++)
                {
                    temp_ch=file.sbumpc();
                    cout << temp_ch;
                    if(temp_ch != ' ' && temp_ch != '\n')
                        temp_str.append(1, temp_ch);
                }
                cout << endl;
                file.close();
                if(!temp_str.size())
                    throw "Input file is empty.";

                break;
            case 2:
                cout << "Enter an expression:" << endl;
                getline(cin, temp_str);
                if(temp_str.empty())
                    throw new my_ex("Empty.");

                break;
            }

            str_buf.str(temp_str);
            temp_str.clear();
            auto root = new BinTree;
            buildBT_RLR(is_str, root);
            char t_ch = '\0';
            is_str.get(t_ch);
            if(t_ch)
                throw new my_ex("Error: Argument expected. (Odd operation)");
            is_str.clear();

            cout << "Binary tree:" << endl;
            printBT(root);

            cout << "Infix form:" << endl;
            prefix_to_infix(root);
            cout << endl;

            fixBT(root);
            cout << "Fixed binary tree:" << endl;
            printBT(root);
            delete root;
        }
        catch (exception* ex)
        {
            cout << ex->what() << endl;
            temp_str.clear();
            is_str.clear();
        }
        catch (...)
        {
            cout << "It isn't an integer!" << endl;
            continue;
        }
    }
    return 0;
}
```

2. Код **btree.h:**

```
#include "myexception.h"
```

```
#ifndef BTREE_H
#define BTREE_H

typedef char base;

class BinTree
{
private:
    base info;
    BinTree* lt;
    BinTree* rt;
public:
    //-------------------------------------
    BinTree()
    {
        info = '\0';
        lt = nullptr;
        rt = nullptr;
    }
    //-------------------------------------
    base
    RootBT()
    {
        if(!this)
            throw new my_ex("Error: RootBT(null)");
        else
            return this->info;
    }
    //-------------------------------------
    BinTree*
    Left()
    {
        if (!this)
            throw new my_ex("Error: Left(null)");
        else
            return this->lt;
    }
    //-------------------------------------
    BinTree*
    Right()
    {
        if (!this)
            throw new my_ex("Error: Right(null)");
        else
            return this->rt;
    }
    //-------------------------------------
    void
    ConsBT(const base x, BinTree* lst=new BinTree, BinTree* rst=new BinTree)
    {
        if (this)
        {
            this->info = x;
            this->lt = lst;
            this->rt = rst;
        }
        else
            throw new my_ex("Error: ConsBT(null)");
    }
    //-------------------------------------
    ~BinTree()
```

```
    {
        if(this)
        {
            if(this->lt)
                delete this->lt;
            if(this->rt)
                delete this->rt;
        }
    }
    //-----------------------------------
};

#endif // BTREE_H
```

3. Код **api.h:**

```
#include <iostream>
#include <istream>
#include "btree.h"

using namespace std;

#ifndef API_H
#define API_H

bool
isSign(char ch)
{ return ch=='*' || ch=='+' || ch=='-'; }

bool
isTerm(char ch)
{ return isdigit(ch) || (ch>='a' && ch<='z'); }

void
buildBT_RLR(istream& is_str, BinTree* root=new BinTree, bool flag=false)
{
    char cur_ch;

    if(is_str>>cur_ch)
    {
        if(isSign(cur_ch))
        {
            root->ConsBT(cur_ch);
            buildBT_RLR(is_str, root->Left(), true);
            buildBT_RLR(is_str, root->Right(), true);
        }
        else if(isTerm(cur_ch))
        {
            root->ConsBT(cur_ch, nullptr, nullptr);
            flag = false;
        }
        else
            throw new my_ex("Error: Invalid character.");
    }
    else if(flag)
        throw new my_ex("Error: Argument expected.");
}

void
printBT(BinTree* root, unsigned int i=0)
{
    if(root)
```

```
    {
        if(root->Right())
            printBT(root->Right(), ++i);
        else
        {
            for(unsigned j=0; j<i; j++)
                cout << '\t';
            cout << root->RootBT() << endl;
            return;
        }

        for(unsigned j=0; j<i-1; j++)
            cout << '\t';
        cout << root->RootBT() << endl;

        if(root->Left())
            printBT(root->Left(), i);
    }
}

void
prefix_to_infix(BinTree* root, BinTree* prev=nullptr)
{
    if(root->Left())
    {
        if(root->Right() && isTerm(root->Right()->RootBT()))
            cout << '(';
        prefix_to_infix(root->Left(), root);
    }
    else
    {
        cout << root->RootBT();
        if(prev && root==prev->Right())
            cout << ')';
        return;
    }

    cout << root->RootBT();

    if(root->Right())
        prefix_to_infix(root->Right(), root);
}

void
fixBT(BinTree* root, BinTree* pv_rt=nullptr)
{
    if(root->Left())
        fixBT(root->Left(), root);
    else
    {
        switch(pv_rt->RootBT())
        {
        case '*':
            if(pv_rt->Left()->RootBT()=='0' || pv_rt->Right()->RootBT()=='0')
                pv_rt->ConsBT('0', nullptr, nullptr);
                     else  if(pv_rt->Left()->RootBT()=='1'  ||  pv_rt->Right()-
>RootBT()=='1')
            {
                if(pv_rt->Left()==root)
                             pv_rt->ConsBT(pv_rt->Right()->RootBT(),  nullptr,
nullptr);
                else if(pv_rt->Right()==root)
```

```
                    pv_rt->ConsBT(pv_rt->Left()->RootBT(), nullptr, nullptr);
            }
            break;
        case '+':
            if(pv_rt->Left()->RootBT()=='0')
                pv_rt->ConsBT(pv_rt->Right()->RootBT(), nullptr, nullptr);
            else if(pv_rt->Right()->RootBT()=='0')
                pv_rt->ConsBT(pv_rt->Left()->RootBT(), nullptr, nullptr);
            break;
        case '-':
            if(root->RootBT()=='0' && pv_rt->Right()==root)
                pv_rt->ConsBT(pv_rt->Left()->RootBT(), nullptr, nullptr);
            break;
        }
    }

    if(root->Right())
        fixBT(root->Right(), root);
}

#endif // API_H
```

4. Код **myexception.h:**

```
#include <exception>

using namespace std;

#ifndef MYEXCEPTION_H
#define MYEXCEPTION_H

class my_ex : public exception
{
public:
    my_ex(const char* exp) noexcept
    {
        msg = exp;
    }
    virtual const char* what() const noexcept
    {
        return msg;
    }
    virtual ~my_ex(){}
private:
    const char* msg;
};

#endif // MYEXCEPTION_H
```

5. Код **Makefile:**

```
all: main.cpp btree.h api.h myexception.h
	g++ main.cpp -o start

clean:
	rm start
```