Министерство науки и высшего образования РФ

Федеральное бюджетное государственное образовательное учреждение высшего профессионального образования

Пермский национальный исследовательский политехнический университет (ПНИПУ)

Факультет: Электротехнический (ЭТФ)

Направление: 09.03.01 «Информатика и вычислительная техника (Автоматизированные системы обработки информации и управления)»

Кафедра: «Информационные технологии и автоматизированных систем» (ИТАС)

Теория алгоритмов

Лабораторная работа №8

Динамические структуры данных

Студент: Балтаев Э. А.

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

(дата, подпись)

Группа: АСУ-20-1бзу

Работу проверил:

доцент кафедры ИТАС

Полякова О. А.

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Подпись:\_\_\_\_\_\_\_\_\_\_\_

Пермь 2021г.

Постановка задачи:

1. Сформировать однонаправленный список, тип информационного поля указан в варианте.
2. Распечатать полученный список.
3. Выполнить обработку списка в соответствии с заданием.
4. Распечатать полученный список.
5. Удалить список из памяти.
6. Сформировать двунаправленный список, тип информационного поля указан в варианте.
7. Распечатать полученный список.
8. Выполнить обработку списка в соответствии с заданием.
9. Распечатать полученный список.
10. Удалить список из памяти.
11. Сформировать идеально сбалансированное бинарное дерево, тип информационного поля указан в варианте.
12. Распечатать полученное дерево.
13. Выполнить обработку дерева в соответствии с заданием, вывести полученный результат.
14. Преобразовать идеально сбалансированное дерево в дерево поиска.
15. Распечатать полученное дерево.

Задание варианта:

|  |  |  |  |
| --- | --- | --- | --- |
| 7 | Тип информационного поля double.  Добавить в список после каждого элемента с отрицательным информационным полем элемент с информационным полем равным 0. | - | Тип информационного поля char\*.  Найти количество элементов дерева, начинающихся с заданного символа. |

Определение классов

1) Очередь

#pragma once

#include <iostream>

#include "point.h"

using namespace std;

template <class T>

class MyQueue

{

private:

point<T>\* curr;

point<T>\* beg;

int len;

public:

MyQueue();

MyQueue(const MyQueue<T>& que);

~MyQueue();

T& front();

T& back();

int size();

bool empty();

void push(const T& val);

void pop();

};

template<class T>

inline MyQueue<T>::MyQueue()

{

beg = curr = 0;

len = 0;

}

template<class T>

inline MyQueue<T>::MyQueue(const MyQueue<T>& que)

{

point<T>\* p = que.curr;

while (p != 0) {

push(p->data);

p = p->prev;

}

}

template<class T>

inline MyQueue<T>::~MyQueue()

{

point<T>\* p = beg;

while (p != 0) {

p = beg->next;

delete beg;

beg = p;

}

beg = curr = 0;

}

template<class T>

inline T& MyQueue<T>::front()

{

if (curr == 0)

throw exception("пустая очередь");

return curr->data;

}

template<class T>

inline T& MyQueue<T>::back()

{

if (beg == 0)

throw exception("пустая очередь");

return beg->data;

}

template<class T>

inline int MyQueue<T>::size()

{

return len;

}

template<class T>

inline bool MyQueue<T>::empty()

{

return len == 0;

}

template<class T>

inline void MyQueue<T>::push(const T& val)

{

point<T>\* temp = make\_point(val);

// если пустой

if (len == 0) {

curr = beg = temp;

}

else {

temp->next = beg;

beg->prev = temp;

beg = temp;

// чтобы удалять с начала

if (len == 1)

curr->prev = temp;

}

len++;

}

template<class T>

inline void MyQueue<T>::pop()

{

point<T>\* temp = curr;

curr = curr->prev;

delete temp;

len--;

if (len != 0)

curr->next = 0;

else

curr = beg = 0;

}

2) Стек

#pragma once

#include <iostream>

#include "point.h"

using namespace std;

template <class T>

class MyStack

{

private:

point<T>\* beg;

point<T>\* curr;

int len;

public:

MyStack();

MyStack(const MyStack<T>& st);

~MyStack();

int size();

bool empty();

T& top();

void push(const T& val);

void pop();

};

template<class T>

inline MyStack<T>::MyStack()

{

beg = curr = 0;

len = 0;

}

template<class T>

inline MyStack<T>::MyStack(const MyStack<T>& st)

{

point<T>\* p = st.beg;

while (p != 0) {

push(p->data);

p = p->next;

}

}

template<class T>

inline MyStack<T>::~MyStack()

{

point<T>\* p = beg;

while (p != 0) {

p = p->next;

delete beg;

beg = p;

}

beg = curr = 0;

}

template<class T>

inline int MyStack<T>::size()

{

return len;

}

template<class T>

inline bool MyStack<T>::empty()

{

return len == 0;

}

template<class T>

inline T& MyStack<T>::top()

{

if (curr == 0)

throw exception("пустой стек");

return curr->data;

}

template<class T>

inline void MyStack<T>::push(const T& val)

{

point<T>\* temp = make\_point(val);

// если пусто

if (curr == 0) {

beg = curr = temp;

}

else {

curr->next = temp;

temp->prev = curr;

curr = temp;

}

len++;

}

template<class T>

inline void MyStack<T>::pop()

{

point<T>\* p = curr;

curr = curr->prev;

delete p;

len--;

if (len == 0)

beg = curr = 0;

else

curr->next = 0;

}

3) Элемент

#pragma once

template <class T>

struct point

{

T data;

point<T>\* next;

point<T>\* prev;

};

template <class T>

point<T>\* make\_point(const T& data) {

point<T>\* p = new point<T>;

p->data = data;

p->next = 0;

p->prev = 0;

return p;

}

Определение компонентных функций

1) Очередь

#pragma once

#include "MyQueue.h"

MyQueue<double> first\_make() {

int n;

cout << "N? "; cin >> n;

MyQueue<double> temp;

double val;

cout << "Введите " << n << " значений через пробел: ";

for (int i = 0; i < n; i++) {

cin >> val;

temp.push(val);

}

return temp;

}

void first\_show(string text, MyQueue<double>& que) {

cout << text;

MyQueue<double> temp;

double p;

while (!que.empty()) {

p = que.front();

cout << p << " ";

temp.push(p);

que.pop();

}

while (!temp.empty()) {

p = temp.front();

que.push(p);

temp.pop();

}

cout << endl;

}

void first\_insert(MyQueue<double>& que) {

MyQueue<double> temp;

double p;

while (!que.empty()) {

p = que.front();

temp.push(p);

if (p < 0)

temp.push(0);

que.pop();

}

while (!temp.empty()) {

p = temp.front();

que.push(p);

temp.pop();

}

}

void first\_exec() {

try {

MyQueue<double> que = first\_make();

first\_show("Исходная очередь: ", que);

first\_insert(que);

first\_show("Очередь после добавления 0 после отрицательных: ", que);

}

catch (exception& err) {

cout << "Ошибка: " << err.what() << endl;

}

}

2) Стэк

#pragma once

#include "MyStack.h"

MyStack<double> second\_make() {

int n;

cout << "N? "; cin >> n;

MyStack<double> temp;

double val;

cout << "Введите " << n << " значений через пробел: ";

for (int i = 0; i < n; i++) {

cin >> val;

temp.push(val);

}

return temp;

}

void second\_show(string text, MyStack<double>& st) {

cout << text;

double val;

MyStack<double> temp;

while (!st.empty()) {

val = st.top();

cout << val << " ";

temp.push(val);

st.pop();

}

while (!temp.empty()) {

val = temp.top();

st.push(val);

temp.pop();

}

cout << endl;

}

void second\_insert(MyStack<double>& st) {

double val;

MyStack<double> temp;

while (!st.empty()) {

val = st.top();

if (val < 0)

temp.push(0);

temp.push(val);

st.pop();

}

while (!temp.empty()) {

val = temp.top();

st.push(val);

temp.pop();

}

}

void second\_exec() {

try {

MyStack<double> st = second\_make();

second\_show("Исходный стек: ", st);

second\_insert(st);

second\_show("Стек после добавления 0 после отрицательных: ", st);

}

catch (exception& err) {

cout << "Ошибка: " << err.what() << endl;

}

}

3) Бинарное дерево

#pragma once

#include <iostream>

#include <Windows.h>

using namespace std;

struct Point

{

char\* key;

Point\* left;

Point\* right;

};

Point\* make\_point(char\* d)

{

Point\* p = new Point;

p->key = new char[strlen(d)];

strcpy(p->key, d);

p->left = 0;

p->right = 0;

return p;

}

//построение идеально сбалансированного дерева

Point\* Tree(Point\* root, int n)

{

char buf[255];

Point\* p = root;

Point\* r;

int nl, nr;

if (n == 0) {

p = 0;

return p;

}

nl = n / 2;

nr = n - nl - 1;

r = new Point;

cout << "val? "; cin >> buf;

r->key = new char[strlen(buf)];

strcpy(r->key, buf);

r->left = Tree(r->left, nl);

r->right = Tree(r->right, nr);

return r;

}

void getXY(CONSOLE\_SCREEN\_BUFFER\_INFO& csbiInfo) {

HANDLE hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

GetConsoleScreenBufferInfo(hStdout, &csbiInfo);

}

void goXY(int x, int y)

{

COORD coord;

coord.X = x;

coord.Y = y;

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), coord);

}

void printElement(Point\* p, int x, int y, int row)

//обход слева направо

{

if (p)

{

goXY(x, y);

for (int i = 0; i < strlen(p->key) && i < 3; i++)

cout << p->key[i];

printElement(p->left, x - 10, y + 1, row + 1); //переход к левому поддереву

printElement(p->right, x + 10, y + 1, row + 1); //переход к правому поддереву

}

}

void printVertical(Point\* root) {

CONSOLE\_SCREEN\_BUFFER\_INFO csbiInfo;

getXY(csbiInfo);

int x0 = csbiInfo.dwSize.X / 2,

y0 = csbiInfo.dwCursorPosition.Y;

printElement(root, x0, y0, 0);

cout << endl;

}

void find(Point\* p, const char& key, int& count) {

if (p)

{

if (strlen(p->key) > 0 && p->key[0] == key)

count++;

find(p->left, key, count); //переход к левому поддереву

find(p->right, key, count); //переход к правому поддереву

}

}

// ДЕРЕВО ПОИСКА

Point\* Add(Point\* root, char\* d)

{

Point\* p = root;//корень дерева

Point\* r = 0;

bool ok = false; //флаг для проверки существования элемента d в дереве

while (p && !ok) {

r = p;

if (strcmp(d, p->key) == 0)

ok = true; //элемент уже существует

else

if (strcmp(d, p->key) > 0)

p = p->left; //пойти в левое поддерево

else

p = p->right; //пойти в правое поддерево

}

if (ok)

return p; //найдено, не добавляем

Point\* New\_point = new Point;

New\_point->key = new char[strlen(d)];

strcpy(New\_point->key, d);

New\_point->left = 0;

New\_point->right = 0;

// если d<r->key, то добавляем его в левое поддерево

if (strcmp(d, r->key) > 0)

r->left = New\_point;

// если d>r->key, то добавляем его в правое поддерево

else

r->right = New\_point;

return New\_point;

}

void formTree(Point\* p, Point\* rootTree) {

if (p)

{

Add(rootTree, p->key);

formTree(p->left, rootTree); //переход к левому поддереву

formTree(p->right, rootTree); //переход к правому поддереву

}

}

void third\_exec() {

int n;

char buf[255];

cout << "\nИдеально-сбалансированное дерево\n";

cout << "N? "; cin >> n;

Point\* root = new Point;

root = Tree(root, n);

printVertical(root);

cout << "\nКоличество элементов нач-ся с заданного символа\n";

char key;

cout << "c? "; cin >> key;

int count = 0;

find(root, key, count);

cout << "Значение = " << count << endl;

cout << "\nДерево поиска\n";

Point\* rootTree = make\_point(root->key);

formTree(root, rootTree);

printVertical(rootTree);

}

Определение главного файла

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include "first.h"

#include "second.h"

#include "third.h"

int main()

{

system("chcp 1251");

cout << "\n1 часть\n"; first\_exec();

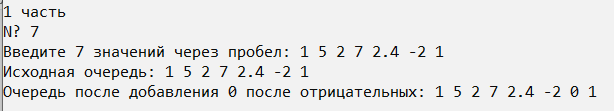
cout << "\n2 часть\n"; second\_exec();

cout << "\n3 часть\n"; third\_exec();

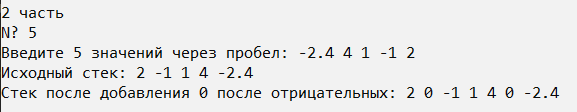
}

Результаты программы:

// Очередь



// Стэк



// Бинарное дерево

