Министерство образования РФ

Пермский национальный исследовательский политехнический университет

Кафедра ИТАС

ЛАБОРАТОРНАЯ РАБОТА № 8

ТЕОРИЯ АЛГОРИТМОВ И СТРУКТУР ДАННЫХ ЗА II СЕМЕСТР

Выполнил студент:

Тимолянов Григорий Констнатинович

Группа РИС-21-1бз

Проверила:

Доцент кафедры ИТАС

Полякова Ольга Андреевна

ПЕРМЬ 2024

# Лабораторная работа №8 "Динамические структуры данных"

**Вариант №15**

**Цель:** 1) Получить практические навыки работы с однонаправленными списками;

2) получить практические навыки работы с двунаправленными списками;

3) получить практические навыки работы с деревьями.

## **Задача:**

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.

|  |  |  |  |
| --- | --- | --- | --- |
| 15 | Тип информационного поля double.  Добавить в список после каждого элемента с отрицательным информационным полем элемент с информационным полем равным 0. | Тип информационного поля int.  Удалить из списка все элементы с четными номерами (2, 4, 6 и. т. д.). | Тип информационного поля char\*.  Найти количество элементов дерева, начинающихся с заданного символа |

**Решение:**

Файл Динамические структуры.cpp

#include <iostream>

#include "Односвязный список.cpp"

#include "Двусвязный список.cpp"

#include "Бинарное дерево.cpp"

using namespace std;

int main()

{

setlocale(LC\_ALL, "Rus");

SingleList singleList;

cout << "Односвязный список (0 - конец списка), для стека:\n\n";

singleList.GetData(true);

cout << "\nНаш список (стек):\n\n";

singleList.PrintData();

singleList.DeleteFirst();

cout << "\nОдносвязный список (0 - конец списка), для очереди:\n\n";

singleList.GetData();

cout << "\nНаш список (очередь):\n\n";

singleList.PrintData();

singleList.InsertDataAfterNegative();

cout << "\nДобавляем в список после каждого отрицательного элемента 0:\n\n";

singleList.PrintData();

DoubleList doubleList;

cout << "\n\nДвусвязный список (0 - конец списка):\n\n";

doubleList.GetData();

cout << "\nНаш список:\n\n";

doubleList.PrintData();

doubleList.DeleteEvenNumbers();

cout << "\nУдаляем из списка все элементы с четными номерами:\n\n";

doubleList.PrintData();

Tree tree;

cout << "\n\nБинарное дерево (0 - конец дерева):\n\n";

tree.GetData();

cout << "\nНаше дерево:\n\n";

tree.PrintData(tree.treetop);

int count = tree.compare(tree.treetop, (char\*)"1");

cout << endl << count << " строк, начинающихся с цифры 1" << endl;

return 0;

}

Файл Односвязный список.cpp

#include <iostream>

using namespace std;

struct NameDataSet {

double data;

NameDataSet\* next;

};

struct SingleList {

NameDataSet\* first = NULL;

NameDataSet\* last = NULL;

~SingleList() {

delete first;

delete last;

}

void AddDataQueue(double d) {

if (first == NULL) {

first = new NameDataSet;

last = first;

}

else {

last->next = new NameDataSet;

last = last->next;

}

last->data = d;

last->next = NULL;

}

void AddDataStack(double d) {

NameDataSet\* tmp = new NameDataSet;

tmp->data = d;

tmp->next = first;

first = tmp;

}

void GetData(bool stack = false) {

double data;

do {

cin >> data;

if (stack) {

AddDataStack(data);

}

else {

AddDataQueue(data);

}

} while (data != 0);

}

void InsertDataAfterNegative() {

NameDataSet\* newData = NULL;

last = first;

while (last) {

if (last->data < 0.0) {

newData = new NameDataSet;

newData->data = 0.0;

newData->next = last->next;

last->next = newData;

last = newData->next;

}

else {

last = last->next;

}

}

}

void PrintData() {

NameDataSet\* current = first;

while (current) {

cout << current->data << "\t";

current = current->next;

}

cout << endl;

}

void DeleteFirst() {

first = NULL;

}

};

Файл Двусвязный список.cpp

#include <iostream>

using namespace std;

struct NameDataDouble {

int data;

NameDataDouble\* prev, \*next;

};

struct DoubleList {

NameDataDouble\* first = NULL;

NameDataDouble\* last = NULL;

/\*~DoubleList() {

delete first;

delete last;

}\*/

void AddData(int d) {

if (first == NULL) {

first = new NameDataDouble;

last = first;

last->prev = NULL;

}

else {

last->next = new NameDataDouble;

last->next->prev = last;

last = last->next;

}

last->data = d;

last->next = NULL;

}

void GetData() {

int data;

do {

cin >> data;

AddData(data);

} while (data != 0);

}

void DeleteEvenNumbers() {

NameDataDouble\* current = first->next;

if (current->next == NULL) {

last = first;

first->next = NULL;;

}

while (current->next) {

current->next->prev = current->prev;

current->prev->next = current->next;

if (current->next->next != NULL) {

current = current->next->next;

}

else {

last = current->next;

return;

}

if (current->next == NULL) {

last = current->prev;

current->prev->next = NULL;

}

}

}

void PrintData() {

NameDataDouble\* current = first;

while (current) {

cout << current->data << "\t";

current = current->next;

}

cout << endl;

}

};

Файл Бинарное дерево.cpp

#include <iostream>

using namespace std;

// Заготовка бинарного дерева

struct NameDataTree {

char\* data;

NameDataTree\* left, \* right;

};

struct Tree {

NameDataTree\* treetop = NULL;

void AddData(struct NameDataTree\*\* root, char\* d) {

if ((!root) && (treetop)) {

root = &treetop;

}

struct NameDataTree\* \_root = NULL;

if (root) {

\_root = (\*root);

}

if (!(\_root)) {

\_root = new NameDataTree;

\_root->data = d;

\_root->left = NULL;

\_root->right = NULL;

if (!treetop) {

treetop = \_root;

}

else if (root) {

(\*root) = \_root;

}

}

else {

if (strlen(\_root->data) > strlen(d)) {

AddData(&(\_root->left), d);

}

else {

AddData(&(\_root->right), d);

}

}

}

void GetData() {

char\* data = NULL;

do {

data = new char[256]; // Allocate memory

cin >> data;

AddData(NULL, data);

} while (atoi(data) != 0);

}

void PrintData(struct NameDataTree\* treetop) {

if (!treetop)

return;

PrintData(treetop->left);

cout << treetop->data << endl;

PrintData(treetop->right);

}

bool \_compare(char\* string, char\* startwith) {

bool bResult = false;

if (strlen(startwith) > strlen(string))

{

return false;

}

int str\_size = strlen(startwith);

if (str\_size > 0) { // If startwith is not empty

bResult = true;

for (int i = 0; i < str\_size; i++) {

if (\*(string + i) != \*(startwith + i)) {

bResult = false;

break;

}

}

}

return bResult;

}

int compare(struct NameDataTree\* treetop, char\* startwith) {

int result = 0;

if (treetop) {

result += compare(treetop->left, startwith);

if (\_compare(treetop->data, startwith)) {

result++;

}

result += compare(treetop->right, startwith);

}

return result;

}

};

# 

