МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

**КАФЕДРА ИИТ**

ОТЧЁТ

по лабораторной работе №14

**«Динамические структуры: списки и деревья»**

Выполнил:

студент 1 курса

группы ПО-9

Харитонович Захар Сергеевич

Проверила:

Хацкевич М. В.

Брест 2022

**Цель работы:** приобретение навыков работы с динамической памятью и указателями. Изучение принципов работы с динамическими структурами данных.

**Порядок выполнения работы**

**Текст задания**

Написать программу, которая вводит с клавиатуры список целых чисел, строит инверсную копию списка, затем вводит с клавиатуры сбалансированное дерево и считает количество листьев дерева.

**Реализация**

*main.cpp*

#include <iostream>

#include "list.h"

#include "baltree.h"

using namespace std;

int main() {

cout << "Size of list: ";

int n;

cin >> n;

cout << "List:\n";

int val;

cin >> val;

listnode \*list = list\_create(val);

for (int i = 1; i < n; i++) {

int val;

cin >> val;

list\_add(list, val);

}

listnode \*revlist = list\_copy(list);

revlist = list\_reverse(revlist);

list\_display(revlist);

cout << "Size of tree: ";

cin >> n;

cout << "Tree:\n";

cin >> val;

treenode \*tree = tree\_create(val);

for (int i = 1; i < n; i++) {

int val;

cin >> val;

tree\_add(tree, val);

}

cout << "Number of leaves: " << tree\_count\_leaves(tree);

return 0;

}

*list.h*

#ifndef TASK5\_LIST\_H

#define TASK5\_LIST\_H

struct listnode {

int value;

listnode \*next;

};

listnode \*list\_create(int value);

void list\_add(listnode \*list, int value);

void list\_display(listnode \*list);

listnode \*list\_copy(listnode \*listsrc);

listnode \*list\_reverse(listnode \*list);

#endif

*list.cpp*

#include "list.h"

#include <iostream>

listnode \*list\_create(int value) {

listnode \*current = new listnode;

current->value = value;

current->next = nullptr;

return current;

}

void list\_add(listnode \*list, int value) {

while (list->next != nullptr) {

list = list->next;

}

listnode \*cur = new listnode;

list->next = cur;

cur->value = value;

cur->next = nullptr;

}

void list\_display(listnode \*list) {

while (list != nullptr) {

std::cout << list->value << std::endl;

list = list->next;

}

}

listnode \*list\_copy(listnode \*listsrc) {

listnode \*listdst = list\_create(listsrc->value);

listsrc = listsrc->next;

while (listsrc != nullptr) {

list\_add(listdst, listsrc->value);

listsrc = listsrc->next;

}

return listdst;

}

listnode \*list\_reverse(listnode \*list) {

listnode \*temp = nullptr;

while (list->next != nullptr) {

listnode \*next = list->next;

list->next = temp;

temp = list;

list = next;

}

list->next = temp;

return list;

}

*baltree.h*

#ifndef TASK5\_BALTREE\_H

#define TASK5\_BALTREE\_H

struct treenode {

int value;

treenode \*left;

treenode \*right;

int leftChildren;

int rightChildren;

};

treenode \*tree\_create(int value);

void tree\_add(treenode \*tree, int value);

int tree\_count\_leaves(treenode \*tree);

int tree\_dfs(treenode \*tree);

#endif

*baltree.cpp*

#include "baltree.h"

treenode \*tree\_create(int value) {

auto \*node = new treenode;

node->left = nullptr;

node->right = nullptr;

node->value = value;

node->leftChildren = 0;

node->rightChildren = 0;

return node;

}

void tree\_add(treenode \*tree, int value) {

while (tree->left != nullptr && tree->right != nullptr) {

if (tree->leftChildren <= tree->rightChildren) {

tree->leftChildren++;

tree = tree->left;

} else {

tree->rightChildren++;

tree = tree->right;

}

}

if (tree->left == nullptr) {

tree->left = tree\_create(value);

tree->leftChildren++;

} else {

tree->right = tree\_create(value);

tree->rightChildren++;

}

}

int tree\_count\_leaves(treenode \*tree) {

return tree\_dfs(tree);

}

int tree\_dfs(treenode \*tree) {

int leaves = 0;

if (tree->left != nullptr) {

leaves += tree\_dfs(tree->left);

}

if (tree->right != nullptr) {

leaves += tree\_dfs(tree->right);

}

if (tree->left == nullptr && tree->right == nullptr) return 1;

return leaves;

}

**Текст задания**

Написать программу, обеспечивающую работу с однонаправленным циклическим списком: добавление/удаление элементов в хвост, просмотр списка, удаление элементов с позиции N по K (не включая N и K).

**Реализация**

*main.cpp*

#include <iostream>

#include "list.h"

using namespace std;

int main() {

listnode \*list = nullptr;

cout << "List size: ";

int s;

cin >> s;

cout << "List:\n";

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

int val;

cin >> val;

list\_push\_back(list, val);

}

list\_display(list);

cout << endl;

list\_pop\_back(list);

list\_display(list);

cout << endl;

cout << "Range to delete (n & k): ";

int n, k;

cin >> n >> k;

list\_delete\_range(list, n, k);

list\_display(list);

return 0;

}

*list.h*

#ifndef TASK9\_LIST\_H

#define TASK9\_LIST\_H

struct listnode {

int value;

listnode \*next;

};

listnode \*list\_create(int value);

void list\_push\_back(listnode \*&list, int value);

void list\_display(listnode \*list);

void list\_pop\_back(listnode \*&list);

void list\_delete\_range(listnode \*&list, int n, int k);

#endif

*list.cpp*

#include "list.h"

#include <iostream>

listnode \*list\_create(int value) {

listnode \*current = new listnode;

current->value = value;

current->next = current;

return current;

}

void list\_push\_back(listnode \*&list, int value) {

if (list == nullptr) {

list = list\_create(value);

return;

}

listnode \*head = list;

while (list->next != head) {

list = list->next;

}

listnode \*cur = new listnode;

list->next = cur;

cur->value = value;

cur->next = head;

list = head;

}

void list\_display(listnode \*list) {

if (list == nullptr) return;

listnode \*head = list;

do {

std::cout << list->value << std::endl;

list = list->next;

} while (list != head);

}

void list\_pop\_back(listnode \*&list) {

listnode \*head = list;

if (list->next == head) {

delete list;

list = nullptr;

return;

}

if (list == nullptr) return;

while (list->next->next != head) {

list = list->next;

}

delete list->next;

list->next = head;

list = head;

}

void list\_delete\_range(listnode \*&list, int n, int k) {

if (list == nullptr) return;

if (k <= n) return;

listnode \*head = list, \*temp;

int count = 0;

do {

if (count == n) {

temp = list;

list = list->next;

temp->next = head;

count++;

continue;

}

if (count == k) {

temp->next = list;

break;

}

if (count > n) {

listnode \*todel = list;

list = list->next;

count++;

delete todel;

continue;

}

count++;

list = list->next;

} while (list != head);

list = head;

}

**Текст задания**

Написать программу, обеспечивающую работу с однонаправленным циклическим списком: добавление/удаление элементов в хвост, просмотр списка, удаление элементов с позиции N по K (не включая N и K).

**Реализация**

*main.cpp*

#include <iostream>

#include "list.h"

using namespace std;

int main() {

listnode \*list = nullptr;

cout << "List size: ";

int n; cin >> n;

cout << "List\n";

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

int temp;

cin >> temp;

list\_push\_front(list, temp);

}

list\_display(list);

cout << endl << "Value to double: ";

int toDouble;

cin >> toDouble;

list\_double(list, toDouble);

list\_display(list);

return 0;

}

*list.h*

#ifndef TASK11\_LIST\_H

#define TASK11\_LIST\_H

struct listnode {

int value;

listnode \*next, \*prev;

};

listnode \*list\_create(int value);

void list\_push\_front(listnode \*&list, int value);

void list\_display(listnode \*list);

void list\_insert(listnode \*pos, int value);

void list\_double(listnode \*list, int toDouble);

#endif

*list.cpp*

#include "list.h"

#include <iostream>

listnode \*list\_create(int value) {

listnode \*current = new listnode;

current->value = value;

current->next = nullptr;

current->prev = nullptr;

return current;

}

void list\_push\_front(listnode \*&list, int value) {

if (list == nullptr) {

list = list\_create(value);

return;

}

listnode \*current = list\_create(value);

current->next = list;

list->prev = current;

list = current;

}

void list\_display(listnode \*list) {

while (list != nullptr) {

std::cout << list->value << std::endl;

list = list->next;

}

}

void list\_insert(listnode \*pos, int value) {

listnode \*cur = list\_create(value);

cur->next = pos->next;

cur->prev = pos;

pos->next->prev = cur;

pos->next = cur;

}

void list\_double(listnode \*list, int toDouble) {

while (list != nullptr) {

if (list->value == toDouble) {

list\_insert(list, toDouble);

list = list->next;

}

list = list->next;

}

}

**Вывод:** изучены и применены принципы использования динамических структур данных.