Министерство образования РФ

Пермский национальный исследовательский политехнический университет

Кафедра ИТАС

ЛАБОРАТОРНАЯ РАБОТА № 8

ПО ИНФОРМАТИКЕ ЗА I СЕМЕСТР

Вариант 5

|  |
| --- |
| Выполнил студент:  Отинов Иван Юрьевич  Группа РИС-20-1бз  Шифр 20-ЭТФ-635  Кафедра ИТАС:  Полякова Ольга Андреевна |

ПЕРМЬ 2020

# Лабораторная работа №8 "Динамические структуры данных”

**Вариант №5**

**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.

**Вариант задания:**

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | Тип информационного поля char\*.  Добавить в список элемент после элемента с заданным информационным полем. | Тип информационного поля int.  Удалить из списка все элементы с четными информационными полями. | Тип информационного поля char.  Найти высоту дерева. |

**Текст программы № 1**

#include <iostream>

using namespace std;

struct point {

char data;

point\* next;

};

struct point\* init(char data);

struct point\* addToHead(point\* head, char data);

struct point\* addToTail(point\* tail, char data);

void addAfterElement(point\* head, char data, char afterdata);

void printList(point\* head);

void deleteList(point\* head);

int main() {

point \*Head;

point \*Tail;

Head = init('b');

Tail = Head;

Head = addToHead(Head, 'a');

Tail = addToTail(Tail, 'c');

Tail = addToTail(Tail, 'd');

Head = addToHead(Head, 'h');

printList(Head);

cout << "Add '4' after 'c'\n";

addAfterElement(Head, '4', 'c');

printList(Head);

deleteList(Head);

Head = NULL;

Tail = NULL;

return 0;

}

struct point\* init(char data) {

struct point\* firstPoint;

firstPoint = (struct point\*)malloc(sizeof(struct point));

firstPoint -> data = data;

firstPoint -> next = NULL;

return(firstPoint);

}

struct point\* addToHead(point \*head, char data) {

struct point \*newElement = (struct point\*)malloc(sizeof(point));

newElement -> data = data;

newElement -> next = head;

return newElement;

}

struct point\* addToTail(point\* tail, char data) {

struct point\* newElement = (struct point\*)malloc(sizeof(point));

newElement->data = data;

newElement->next = NULL;

tail -> next = newElement;

return newElement;

}

void addAfterElement(point \*head, char data, char afterdata) {

struct point\* newElement = (struct point\*)malloc(sizeof(point));

(\*newElement).data = data;

point\* current = head;

while (current != NULL) {

if ((\*current).data == afterdata) {

(\*newElement).next = (\*current).next;

(\*current).next = newElement;

break;

}

current = current->next;

}

}

void printList(point \*head) {

cout << "List = {";

point\* current = head;

if (current == NULL) {

cout << "Empty";

return;

}

while (current != NULL) {

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

current = current->next;

}

cout << "}\n";

}

void deleteList(point \*head) {

point\* current = head;

while (current != NULL) {

point\* toDeleteElement = current;

current = current->next;

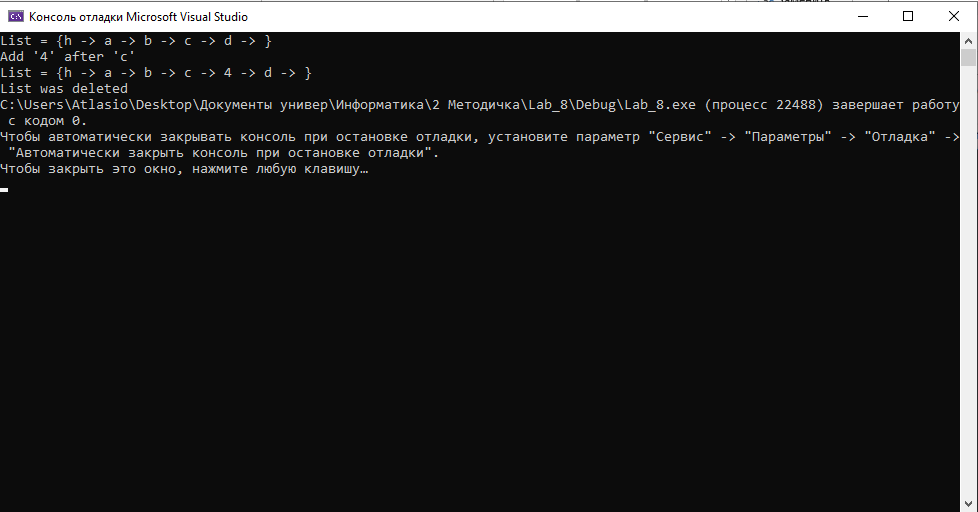
free(toDeleteElement);

}

cout << "List was deleted";

}

**Результаты работы программы № 1:**



**Текст программы № 2**

#include <iostream>

using namespace std;

struct point {

int data;

point\* next;

point\* prev;

};

struct point\* init(int data);

struct point\* addToHead(point\* head, int data);

struct point\* addToTail(point\* tail, int data);

void deleteEvenNumbers(point\*\* head, point\*\* tail);

void printList(point\* head);

void deleteList(point\* head);

int main() {

point\* Head;

point\* Tail;

Head = init(2);

Tail = Head;

Head = addToHead(Head, 1);

Tail = addToTail(Tail, 3);

Tail = addToTail(Tail, 4);

Head = addToHead(Head, 0);

printList(Head);

cout << "Delete even numbers\n";

deleteEvenNumbers(&Head, &Tail);

printList(Head);

deleteList(Head);

return 0;

}

struct point\* init(int data) {

struct point\* firstPoint;

firstPoint = (struct point\*)malloc(sizeof(struct point));

(\*firstPoint).data = data;

(\*firstPoint).next = NULL;

(\*firstPoint).next = NULL;

return(firstPoint);

}

struct point\* addToHead(point\* head, int data) {

struct point\* newElement = (struct point\*)malloc(sizeof(point));

(\*newElement).data = data;

(\*newElement).next = head;

(\*newElement).prev = NULL;

(\*head).prev = newElement;

return newElement;

}

struct point\* addToTail(point\* tail, int data) {

struct point\* newElement = (struct point\*)malloc(sizeof(point));

(\*newElement).data = data;

(\*newElement).next = NULL;

(\*newElement).prev = tail;

(\*tail).next = newElement;

return newElement;

}

void deleteEvenNumbers(point\*\* head, point\*\* tail) {

point\* current = \*head;

if (current == NULL) {

return;

}

while (current != NULL) {

if ((\*current).data % 2 == 0) {

point\* prev = (\*current).prev;

point\* next = (\*current).next;

if (prev == NULL && next != NULL) {

(\*next).prev == NULL;

\*head = next;

} else if (next == NULL && prev != NULL) {

(\*prev).next = NULL;

\*tail = prev;

} else if (next == NULL && prev == NULL) {

\*head = NULL;

\*tail = NULL;

} else {

(\*prev).next = next;

(\*next).prev = prev;

}

point\* toDelete = current;

current = current->next;

free(toDelete);

} else {

current = current->next;

}

}

}

void printList(point\* head) {

cout << "List = {";

point\* current = head;

if (current == NULL) {

cout << "Empty";

return;

}

while (current != NULL) {

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

current = current->next;

}

cout << "}\n";

}

void deleteList(point\* head) {

point\* current = head;

while (current != NULL) {

point\* toDeleteElement = current;

current = current->next;

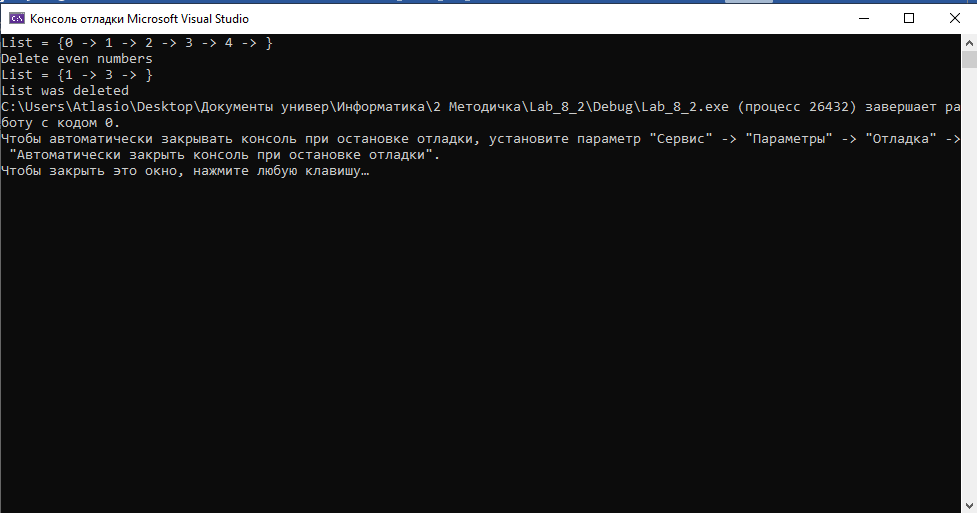
free(toDeleteElement);

}

cout << "List was deleted";

}

**Результаты работы программы № 2:**



**Текст программы № 3**

#include <iostream>

#include <iomanip>

using namespace std;

struct point {

char data;

point\* left;

point\* right;

};

point\* initTree(char data);

void addToTree(point\* root, char data);

void printTree(point\* p, int indent);

int findH(point\* root);

int main() {

point\* Root = initTree('k');

addToTree(Root, 'f');

addToTree(Root, 'c');

addToTree(Root, 'h');

addToTree(Root, 'y');

addToTree(Root, 'a');

addToTree(Root, 'o');

addToTree(Root, 'b');

addToTree(Root, 'e');

cout << "\n";

printTree(Root, 0);

int h = findH(Root);

cout << "H = " << h;

}

point\* initTree(char data) {

struct point\* newElement = (struct point\*)malloc(sizeof(struct point));

(\*newElement).data = data;

(\*newElement).right = NULL;

(\*newElement).left = NULL;

return newElement;

}

void addToTree(point\* root, char data) {

cout << "\nCurrent root = " << (\*root).data;

if ((\*root).data < data) {

cout << "Left";

if ((\*root).right != NULL) {

addToTree((\*root).right, data);

} else {

struct point\* newElement = (struct point\*)malloc(sizeof(struct point));

(\*newElement).data = data;

(\*newElement).right = NULL;

(\*newElement).left = NULL;

(\*root).right = newElement;

}

} else if ((\*root).data > data) {

cout << "Right";

if ((\*root).left != NULL) {

addToTree((\*root).left, data);

} else {

struct point\* newElement = (struct point\*)malloc(sizeof(struct point));

(\*newElement).data = data;

(\*newElement).right = NULL;

(\*newElement).left = NULL;

(\*root).left = newElement;

}

} else {

cout << "MIddle";

return;

}

}

int findH(point\* root) {

if (root == NULL) {

return 0;

}

int hR = findH((\*root).right);

int hL = findH((\*root).left);

if (hR > hL) {

return ++hR;

} else {

return ++hL;

}

}

void printTree(point\* p, int indent)

{

if (p != NULL) {

if (p->right) {

printTree(p->right, indent + 4);

}

if (indent) {

std::cout << std::setw(indent) << ' ';

}

if (p->right) std::cout << " /\n" << std::setw(indent) << ' ';

std::cout << p->data << "\n ";

if (p->left) {

std::cout << std::setw(indent) << ' ' << " \\\n";

printTree(p->left, indent + 4);

}

}

}

**Результаты работы программы № 3:**

