МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ



**Дніпровський національний університет  
залізничного транспорту імені академіка В. Лазаряна**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №11**

**з дисципліни «Основи програмування»**

**на тему: «Лінійні динамічні структури»**

Виконав: студент гр. ПЗ1911

Сіньков Г.О.

Прийняла: ас. каф. КІТ

Нежуміра О. І.

Дніпро, 2020

**Лабораторна робота №11**

**Тема.** Лінійні динамічні структури.

**Мета.** Навчитися реалізовувати лінійні динамічні структури. Оволодіти навичками використання лінійних динамічних структур в розробці програм

**Постановка задачі**

1.Загальна постановка: Розробити програму для заданого індивідуального завдання і обраного рівня складності.

2.Індивідуальна постановка:

Рівень I (60-81 балів):

- реалізувати лінійний двозв’язний список за прикладом лінійного однозв’язного списку (див. п. 2.1.)(інформаційною частиною елементів є цілі числа);

- створити функції формування вказаної структури даних рандомно, з консолі, з текстового і бінарного файлу;

- створити функціїзапису структури у текстовий і бінарний файл;

- реалізувати вказані в табл.Таблиця1 додаткові функції.

|  |  |  |  |
| --- | --- | --- | --- |
| Варіант | Завдання | | |
| Рівень E (достатньо) | Рівень D (задовільно) | Рівень C (добре) |
| 9. | Знайти правого сусіда заданого елемента. | Додати до списку новий елемент перед заданим елементом. | Зі списку виключити ті елементи, які менші за суму своїх сусідів. |

3.Вимоги до програми:

- валідація вхідних даних;

- програма повинна мати багатомодульну структуру, яка базується на реалізації функціонала програми.

**4.Вимоги до тексту програми:**

- специфікації програми і її функцій;

- самодокументованість коду: всі ідентифікатори повинні мати назви, що відповідають суті змінних.

**Зовнішні специфікації**

1. Вхідні дані

Вхідні дані:

Формат вхідних даних

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Найменування даних | Умовне позначення | Вимоги до даних | Приклад |
| 1. | Меню | menu | ціле число | 2 |
| 2. | Під меню | paragraph | ціле число | 6 |
| 3. | Елемент | value | ціле число | 10 |
| 4. | Кількість елементів які треба добавити | quantity | ціле число | 5 |
| 5. | Ліва границя | Lb | ціле число | 0 |
| 6. | Прави границя | Rb | ціле число | 10 |

2. Вихідні дані

Формат вихідних даних

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Найменування даних | Умовне позначення | Вимоги до даних | Приклад |
| 1. | Правий сусід | tmp->next->data (right\_neighbor) | ціле число | 2 |
| 2. | Лівий сусід | left\_neighbor | ціле число | 4 |
| 3. | Елемент який треба виключити | current\_item | ціле число | 3 |

**3. Функціональні вимоги до програми**

Програма повинна реалізувати такі дії:

-створення структури(рандомно, з консолі);

-пошук елементів(сусідів заданого елемента);

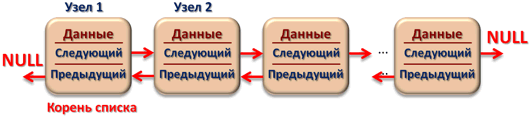
-друк структури;

-видалення елемента;

-перевірка елемента, який ми додаємо в структуру.

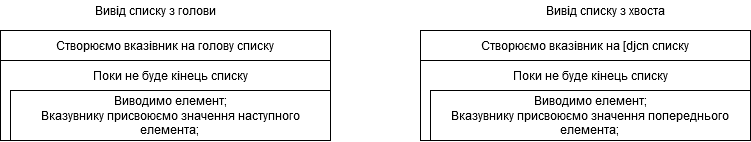
**Опис структури даних, використаних для реалізації завдання**

Відмінність від однозв’язного списку полягає в тому, що в двозв’язному (або двонаправленому списку) вузол складається не з двох, а з трьох частин. У третьому компоненті зберігається покажчик на попередній елемент.

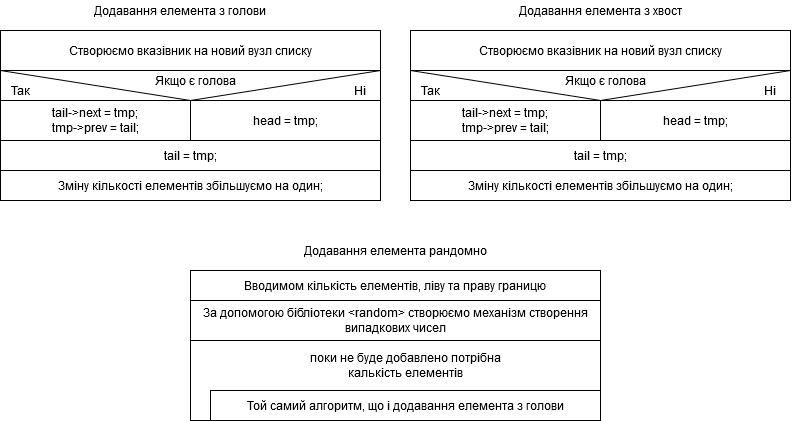


**Алгоритм розв’язання завдання**

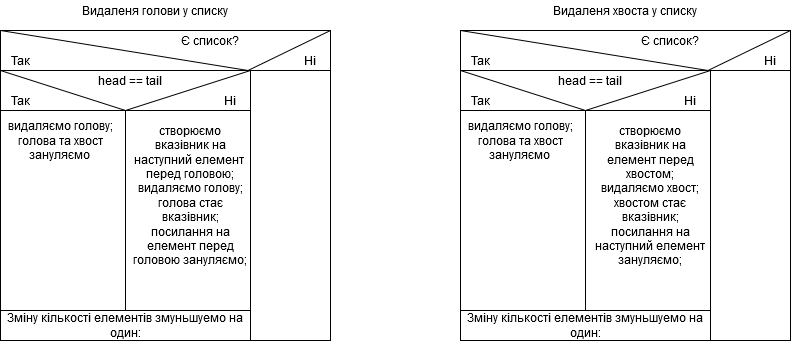
Виведення списку:



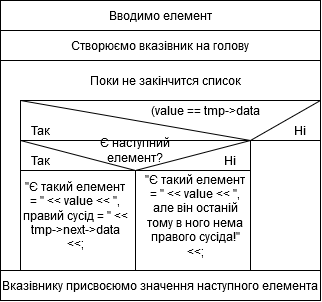
Додавання елемента в список:



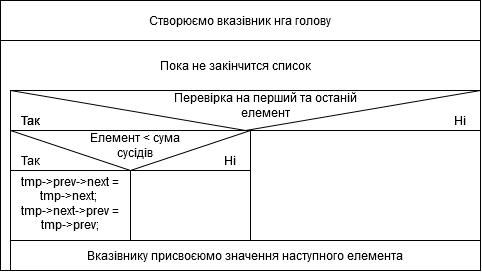
Видалення елемента зі списка:



Пошук правого сусіда:



Виключення елемента, який менше за суму своїх сусідів:



**Текст програми**

#include <iostream>

#include <random>

#include <Windows.h>

typedef int DataType;

struct Node {

DataType data;

Node \*prev;

Node \*next;

};

Node \*newNode(DataType &value) {//додавання нового вузола в список

Node \*tmp = new Node;

tmp->data = value;

tmp->next = NULL;

tmp->prev = NULL;

return tmp;

}

struct Unknown\_ {

int Count = NULL;

Node\* head = NULL;

Node\* tail = NULL;

bool isEmpty();

void addToHead(DataType value);

void addToTail(DataType value);

void Filling();

void delFromHead();

void delFromTail();

void printFromHead();

void printFromTail();

void rightNeighbor(DataType value);

void excludeItems();

};

bool Unknown\_::isEmpty() {

return (head == NULL) ? true : false;

}

void Unknown\_::addToHead(DataType value) {//додавання елемента з голови

Node \*tmp = newNode(value);

if (head) {

head->prev = tmp;

tmp->next = head;

}

else

tail = tmp;

head = tmp;

Count++;

}

void Unknown\_::addToTail(DataType value) {//додавання елемента з хвоста

Node \*tmp = newNode(value);

if (tail) {

tail->next = tmp;

tmp->prev = tail;

}

else

head = tmp;

tail = tmp;

Count++;

}

void Unknown\_::Filling() {//створення структури за допомогою генератора чисел

int Lb, Rb, quantity;

std::cout << "Введіть кількість елементів які треба добавити: ";

std::cin >> quantity;

std::cout << "Введіть ліву границю: ";

std::cin >> Lb;

std::cout << "Введіть ліву границю: ";

std::cin >> Rb;

DataType value;

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<DataType> dis(Lb, Rb);

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

value = dis(gen);

Node \*tmp = newNode(value);

if (head) {

head->prev = tmp;

tmp->next = head;

}

else

tail = tmp;

head = tmp;

Count++;

}

}

void Unknown\_::delFromHead() {//видалення елемента з голови

if (head) {

if (head == tail) {

std::cout << "Видаляємо елемент: " << head->data << std::endl;

delete head;

head = tail = NULL;

}

else {

Node \*tmp = head->next;

std::cout << "Видаляємо елемент: " << head->data << std::endl;

delete head;

head = tmp;

head->prev = NULL;

}

Count--;

}

}

void Unknown\_::delFromTail() {//видалення елемента з хвоста

if (head) {

if (head == tail) {

std::cout << "Видаляємо елемент: " << head->data << std::endl;

delete head;

head = tail = NULL;

}

else {

Node \*tmp = tail->prev;

std::cout << "Видаляємо елемент: " << tail->data << std::endl;

delete tail;

tail = tmp;

tail->next = NULL;

}

Count--;

}

}

void Unknown\_::printFromTail() {//друк інформаційної частини вузла структури з голови

Node \*tmp = tail;

while (tmp) {

std::cout << tmp->data << ' ';

tmp = tmp->prev;

}

std::cout << std::endl;

}

void Unknown\_::printFromHead() {//друк інформаційної частини вузла структури з хвоста

Node \*tmp = head;

while (tmp) {

std::cout << tmp->data << ' ';

tmp = tmp->next;

}

std::cout << std::endl;

}

void Unknown\_::rightNeighbor(DataType value) {//знаходження правого сусіда заданого елемента

DataType a;

Node \*tmp = head;

while (tmp) {

if (value == tmp->data) {

if (tmp->next != NULL)

std::cout << "Є такий елемент = " << value << ", правий сусід = " << tmp->next->data << std::endl;

else

std::cout << "Є такий елемент = " << value << ", але він останій тому в ного нема правого сусіда!" << std::endl;

}

tmp = tmp->next;

}

}

void Unknown\_::excludeItems() {//Зі списку виключити ті елементи, які менші за суму своїх сусідів

DataType current\_item, right\_neighbor, left\_neighbor;

int pos = 1;

Node \*tmp = head;

while (tmp) {

current\_item = tmp->data;

if (pos != Count && pos != 1) {

left\_neighbor = tmp->prev->data;//лівий сусід

right\_neighbor = tmp->next->data;//правий сусід

if (current\_item < (right\_neighbor + left\_neighbor)) {

std::cout << "Виключаємо елемент = " << current\_item << std::endl;

tmp->prev->next = tmp->next;

tmp->next->prev = tmp->prev;

}

}

else

std::cout << "Є такий елемент = " << current\_item << ", але він останій тому в нього нема правого або лівого сусіда!" << std::endl;

tmp = tmp->next;

pos++;

}

}

int input\_value() {//перевірка елемента

int value;

while (true) {

system("cls");

std::cout << "Ведіть елемент: ";

std::cin >> value;

if ((std::cin.peek() != '\n')) {

std::cin.clear();

std::cin.ignore(32767, '\n');

}

else

return value;

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Unknown\_ unknown;

int menu, paragraph;

int value;

do {

system("cls");

std::cout << "1. Вивід списку" << std::endl;

std::cout << "2. Додавання елемента" << std::endl;

std::cout << "3. Видалення елемента" << std::endl;

std::cout << "4. Пошук правого елемента" << std::endl;

std::cout << "5. Зі списку виключити ті елементи, які менші за суму своїх сусідів" << std::endl;

std::cout << "6. Вихід" << std::endl;

std::cin >> menu;

switch (menu) {

case 1: {

do {

system("cls");

std::cout << "1.Вивести список з голови" << std::endl;

std::cout << "2.Вивести список з хвоста" << std::endl;

std::cout << "3.Вихід в меню" << std::endl;

std::cout << "4.Вихід з програми" << std::endl;

std::cin >> paragraph;

switch (paragraph) {

case 1: {

system("cls");

std::cout << "Вивід списку з голови: " << std::endl;

unknown.printFromHead();

system("pause");

break;

}

case 2: {

system("cls");

std::cout << "Вивід списку з хвоста: " << std::endl;

unknown.printFromTail();

system("pause");

break;

}

case 4:

return 0;

default:

break;

}

} while (paragraph != 3);

break;

}

case 2: {

do {

system("cls");

std::cout << "1.Додавання елемента з голови" << std::endl;

std::cout << "2.Додавання елемента з хвоста" << std::endl;

std::cout << "3.Додавання елемента за допомогою генератора чисел" << std::endl;

std::cout << "4.Вихід в меню" << std::endl;

std::cout << "5.Вихід з програми" << std::endl;

std::cin >> paragraph;

switch (paragraph) {

case 1: {

system("cls");

value = input\_value();

unknown.addToHead(value);

system("pause");

break;

}

case 2: {

system("cls");

value = input\_value();

unknown.addToTail(value);

system("pause");

break;

}

case 3: {

system("cls");

unknown.Filling();

system("pause");

break;

}

case 5:

return 0;

default:

break;

}

} while (paragraph != 4);

break;

}

case 3: {

do {

system("cls");

std::cout << "1.Виділення елемента з голови" << std::endl;

std::cout << "2.Виділення елемента з хвоста" << std::endl;

std::cout << "3.Вихід в меню" << std::endl;

std::cout << "4.Вихід з програми" << std::endl;

std::cin >> paragraph;

switch (paragraph) {

case 1: {

system("cls");

unknown.delFromHead();

system("pause");

break;

}

case 2: {

system("cls");

unknown.delFromTail();

system("pause");

break;

}

case 4:

return 0;

default:

break;

}

} while (paragraph != 3);

break;

}

case 4: {

system("cls");

std::cout << "Введіть число: ";

std::cin >> value;

unknown.rightNeighbor(value);

system("pause");

break;

}

case 5: {

system("cls");

unknown.excludeItems();

system("pause");

break;

}

case 6:

return 0;

default:

break;

}

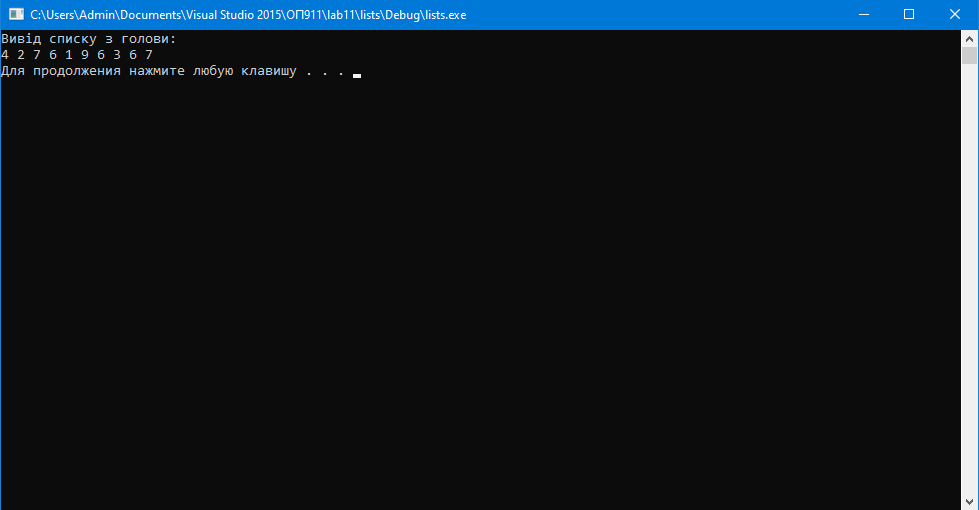
} while (menu != 6);

}

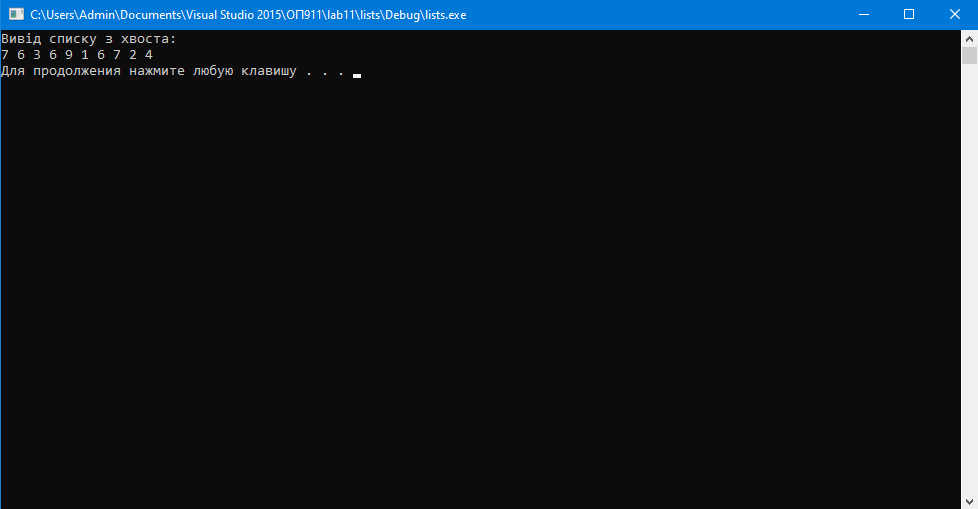
**Результати тестування програми**

Виведення списку:

-з голови

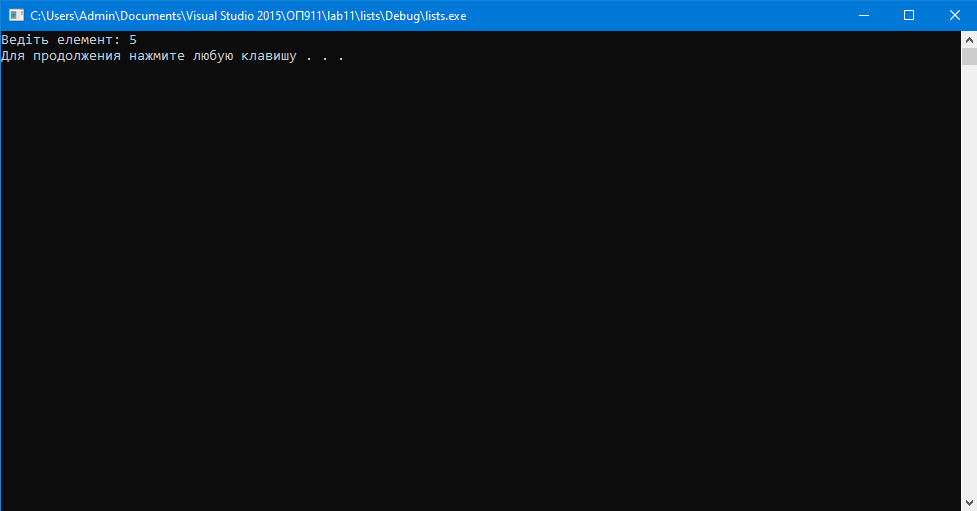


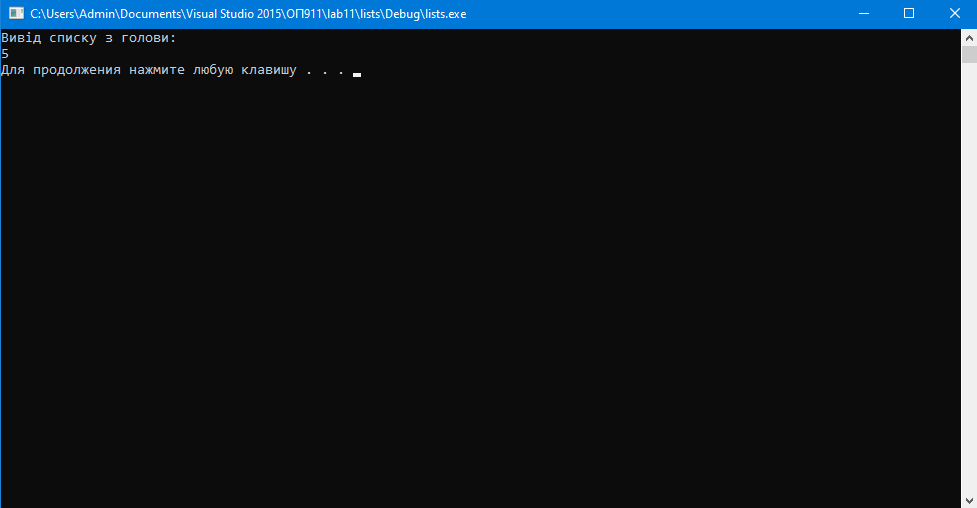
-з хвоста

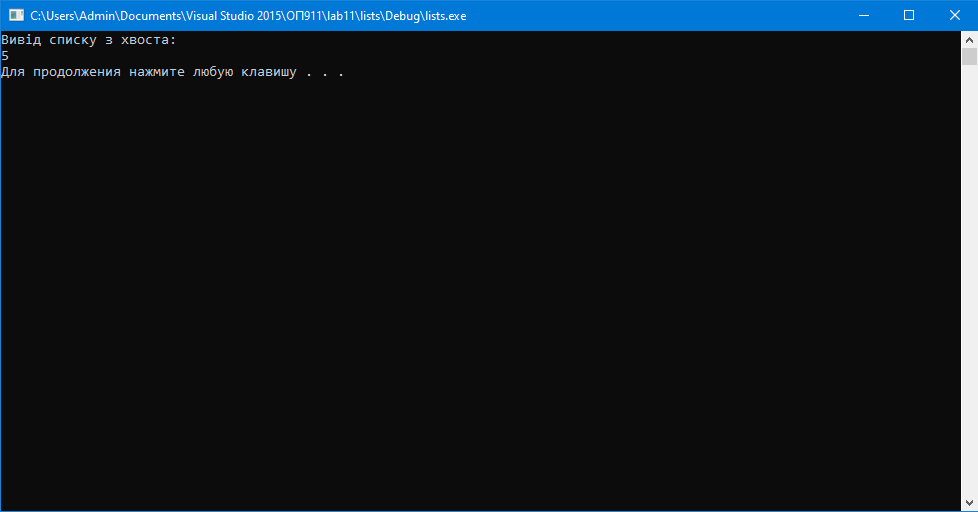


Додавання елемента в список:

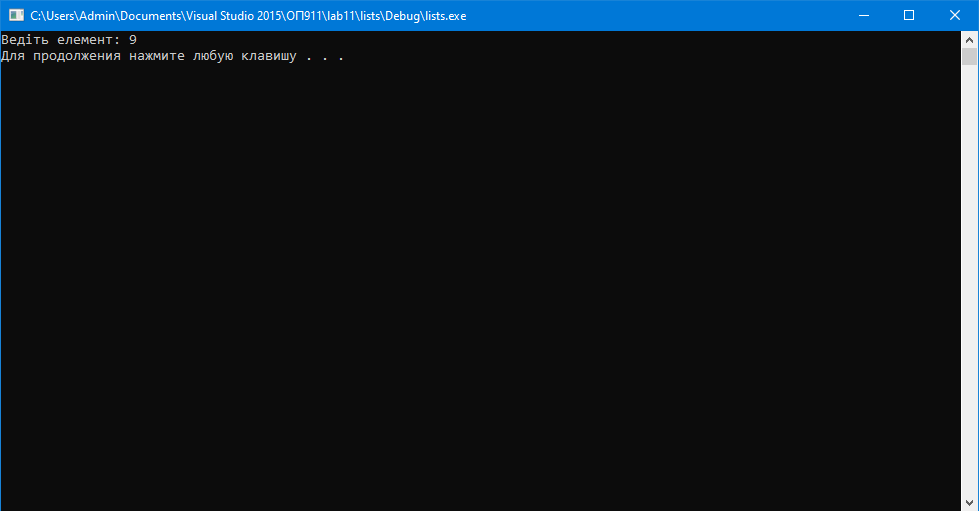
-в голови

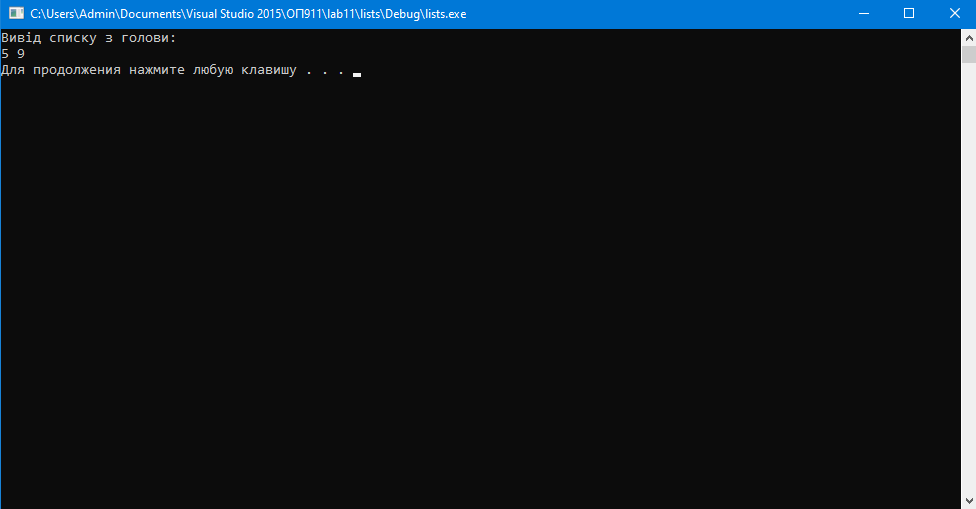


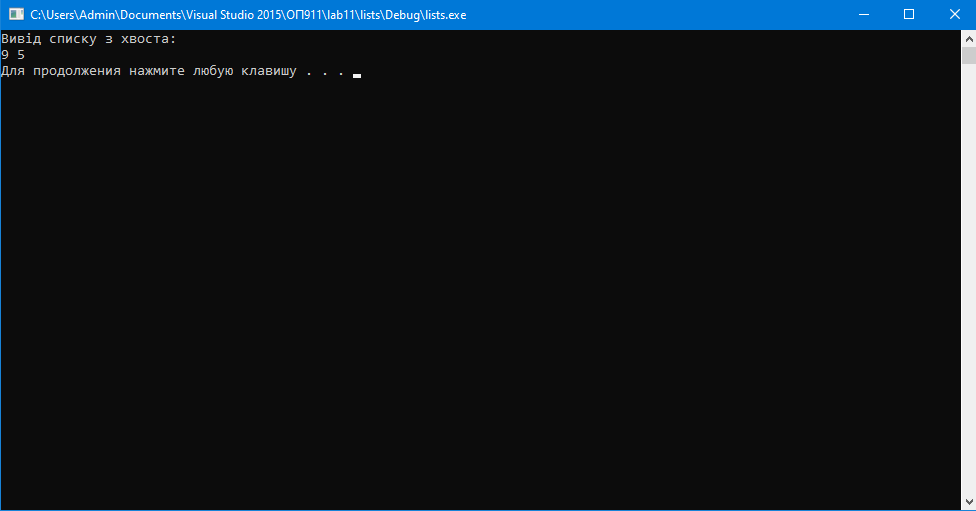




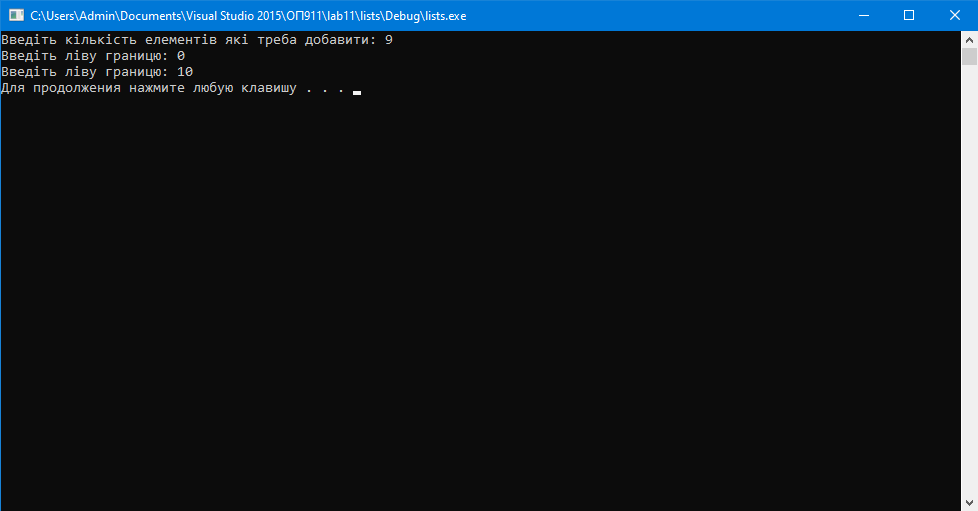
-в хвіст

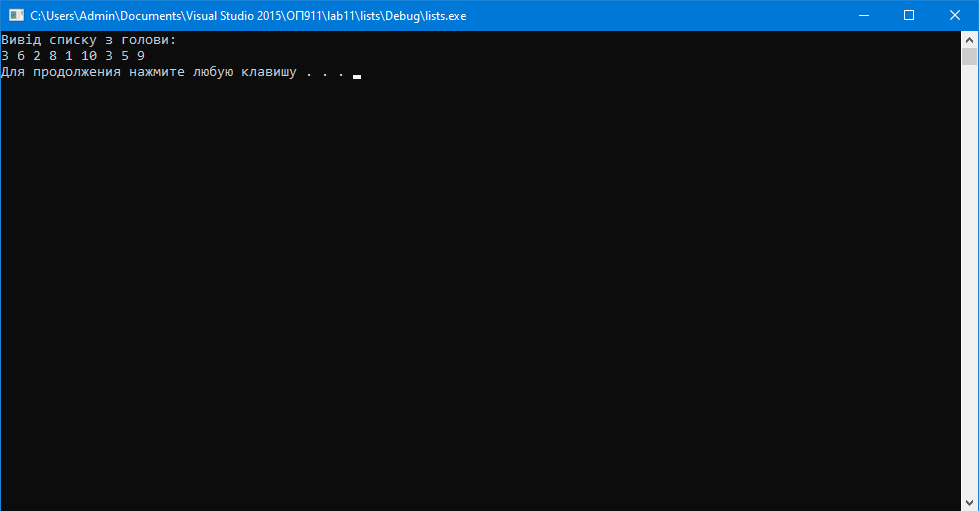






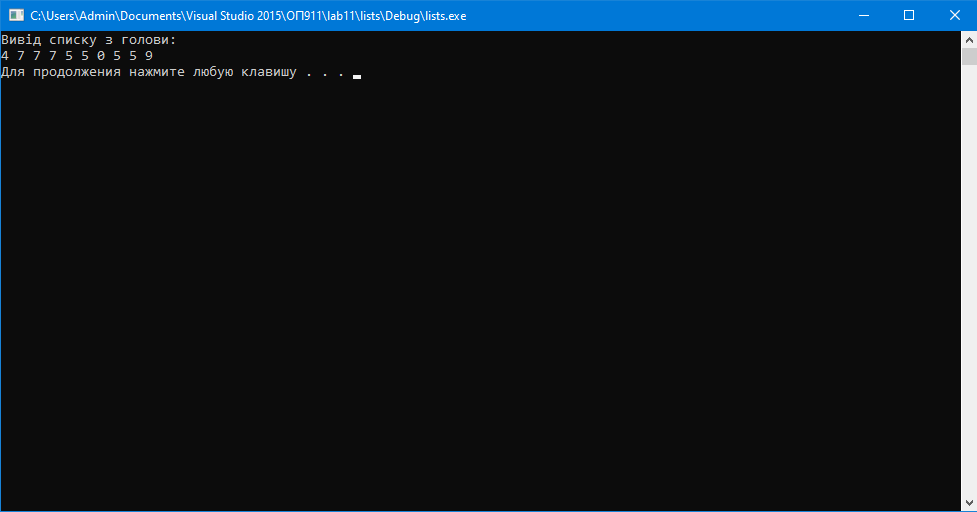
-рандомно

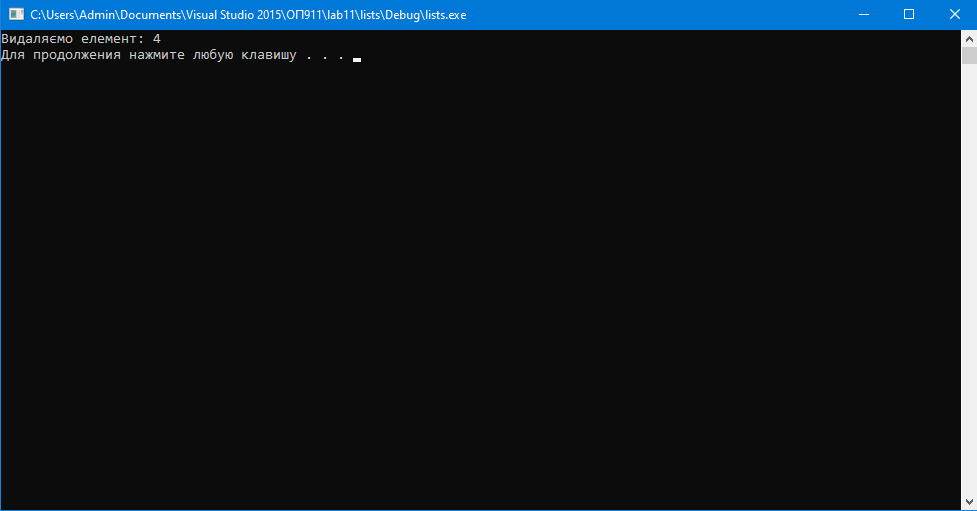


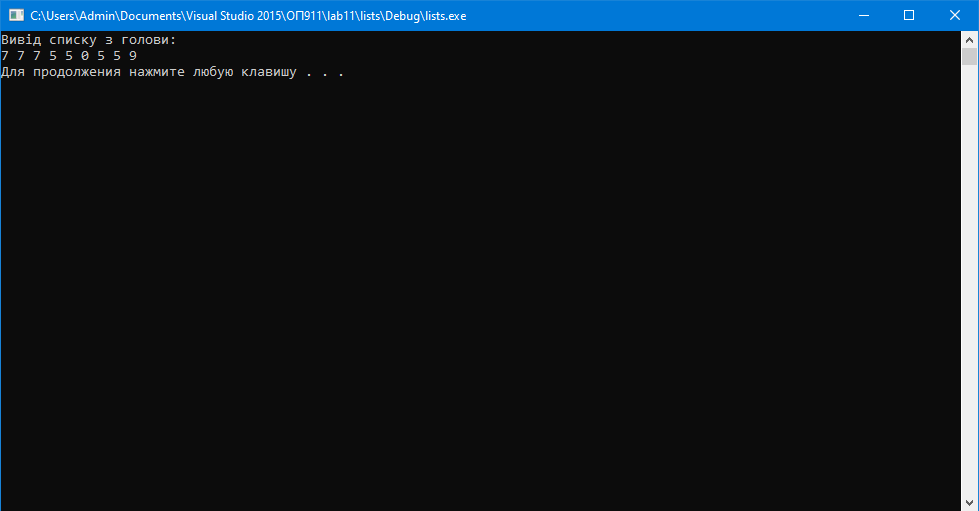


Видалення елемента зі списка:

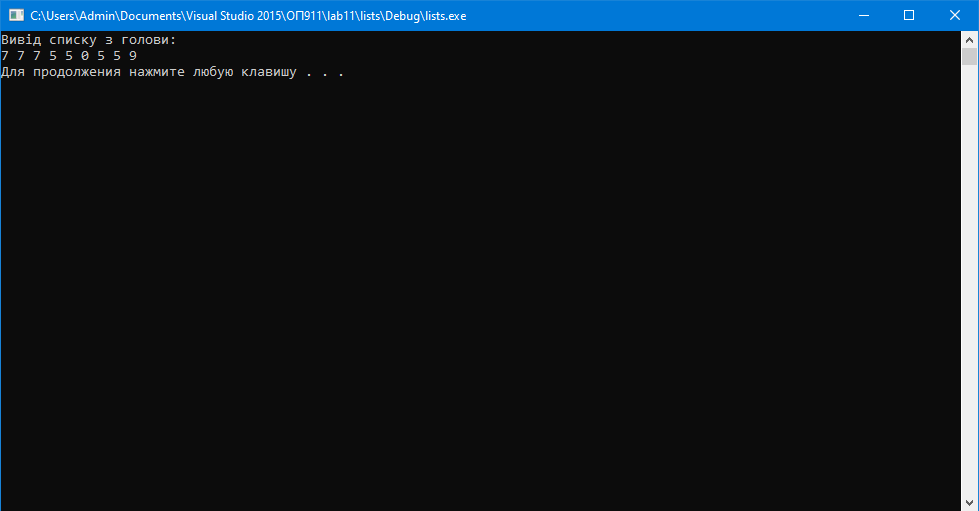
-з голови

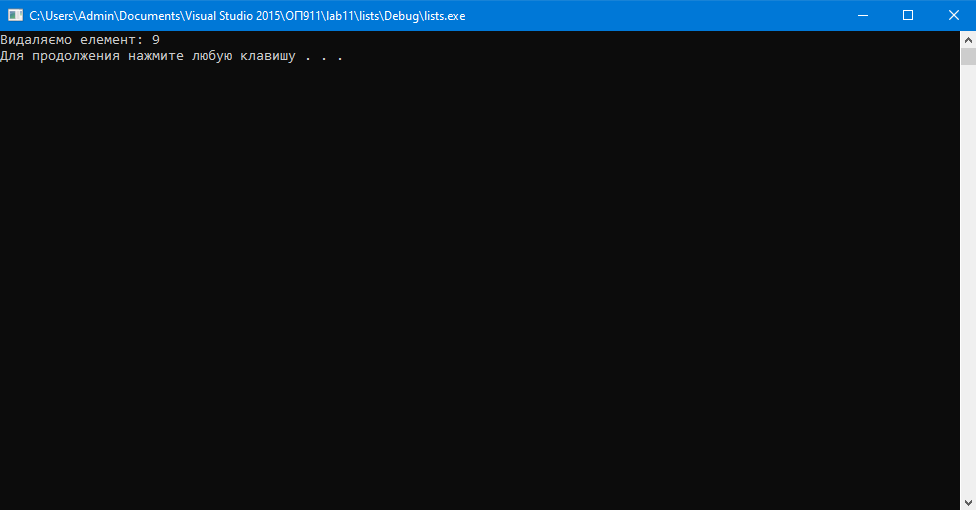


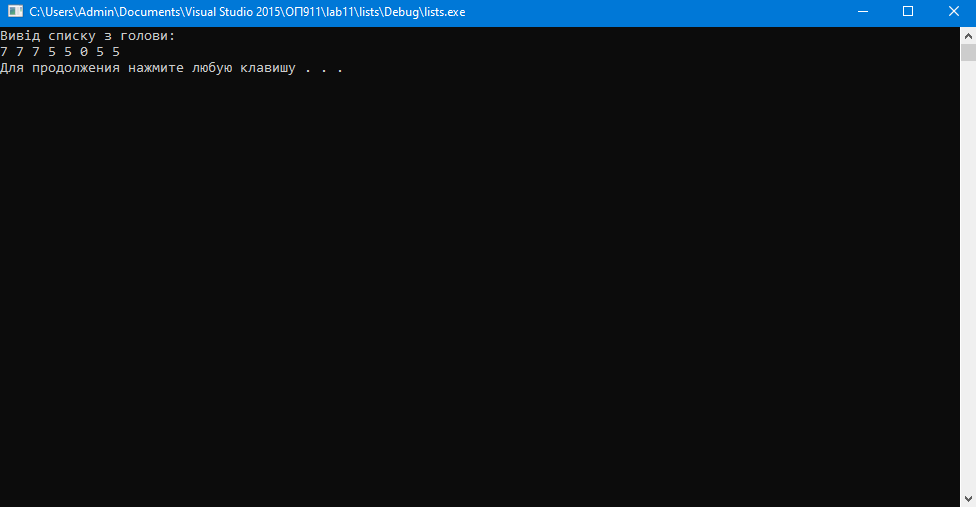




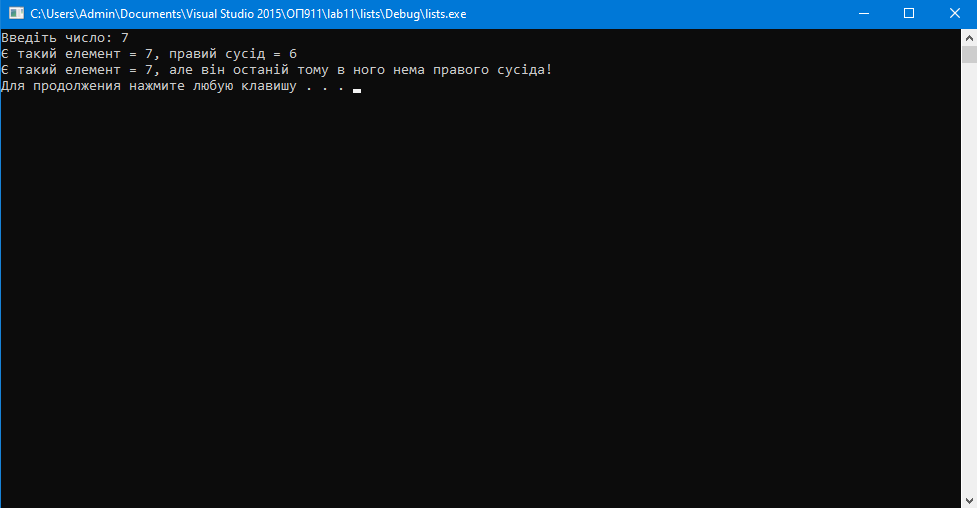
-з хвоста



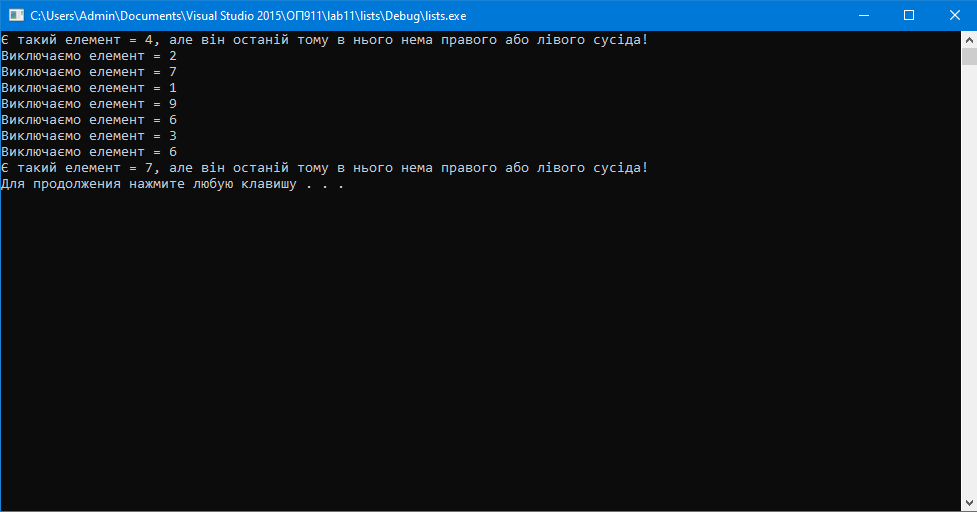




Пошук правого сусіда:



Виключення елемента, який менше за суму своїх сусідів:



Висновок: динамічна структура даних розміщується в динамічної пам'яті, її розмір змінюється під час виконання програми: пам'ять виділяється окремими блоками по мірі необхідності. Блоки зв'язуються один з одним за допомогою покажчиків.

Види динамічних структур (лінійні списки, стеки, черги, дерева) розрізняються способами зв'язку окремих елементів і допустимими операціями. Динамічна структура на відміну від масиву або записи може займати несуміжні ділянки оперативної пам'яті. Динамічні структури широко застосовують і для більш ефективної роботи з даними, розмір яких відомий, особливо для вирішення завдань сортування.

Елемент будь динамічної структури складається з двох частин: *інформаційної,* заради збереження якої і створюється структура, *і покажчиків,* які забезпечують зв'язок елементів один з одним.