==========================================================

**Міністерство освіти і науки України**

**Національний технічний університет**

**«Дніпровська політехніка»**

****

**ЗВІТ**

**про виконання лабораторних робіт**

**з дисципліни**

# «Алгоритми та структури даних»

**Лабораторна робота № 6**

Виконав:

студент гр. 124-19-2

Моторний Андрій Сергійович

Прийняв:

викладач. каф. ФІТ

Сергеева Екатерина Леонидовна

**Дніпро**

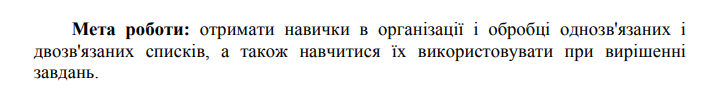
**2020**

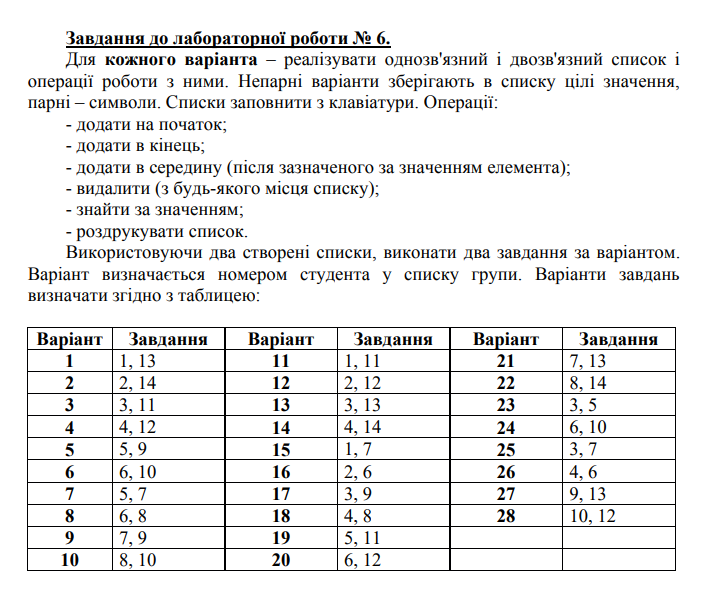
Звіт

1)Лабораторна робота №6

## 2) Лінійні однозв'язані і двозв'язані списки

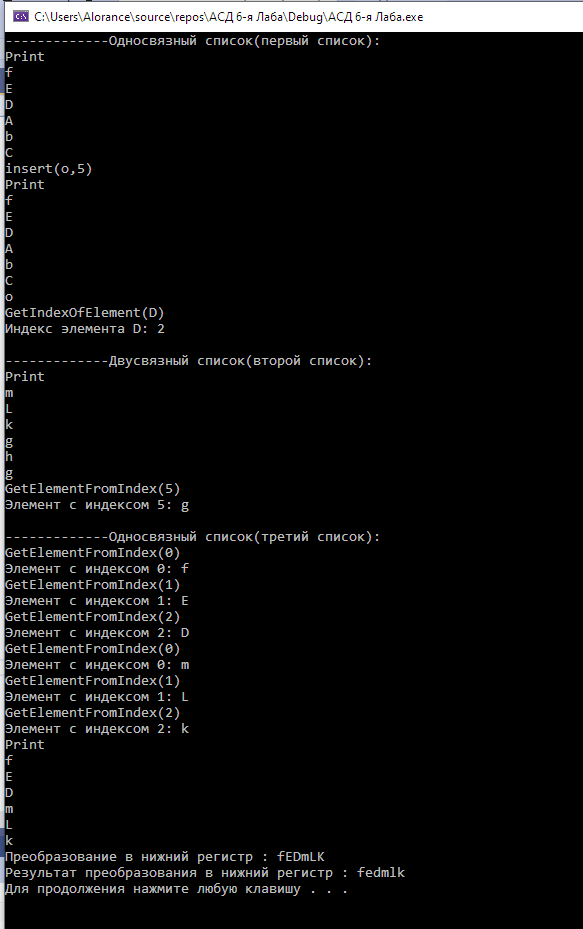
3)Варіант №12











#include <algorithm>

#include <string>

#include <cctype>

#include<iostream>

#include<Windows.h>

using namespace std;

struct Node

{

char data = char();

Node\* next = nullptr;

};

void push\_front(Node\*& head, char data)

{

Node\* temp = new Node;

temp->data = data;

temp->next = head;

head = temp;

}

void push\_back(Node\*& head, char data)

{

if (head == nullptr)

{

Node\* temp = new Node;

temp->data = data;

head = temp;

}

else

{

Node\* current = head;

while (current->next != nullptr)

{

current = current->next;

}

Node\* temp = new Node;

temp->data = data;

current->next = temp;

}

}

void insert(Node\*& head, char data, int index)

{

cout << "insert(" << data << "," << index << ")" << endl;

if (head == nullptr)

push\_front(head, data);

else

{

if (index < 0)

cout << "Введён отрицательный индекс!" << endl;

else

{

Node\* previous = head;

for (int i = 0; i < index; i++)

{

previous = previous->next;

if (previous == nullptr)

{

cout << "Индекс элемента слишком велик!" << endl;

return;

}

}

Node\* newNode = new Node;

newNode->data = data;

newNode->next = previous->next;

previous->next = newNode;

}

}

}

void remove\_at(Node\*& head, int index)

{

cout << "remove\_at(" << index << ")" << endl;

if (head == nullptr)

cout << "В списке нет элементов для удаления!" << endl;

else

{

if (index == 0)

{

Node\* temp = head;

head = head->next;

delete temp;

temp = nullptr;

}

else

{

if (index < 0)

cout << "Введён отрицательный индекс!" << endl;

else

{

Node\* previous = head;

for (int i = 0; i < index - 1; i++)

{

previous = previous->next;

if (previous == nullptr)

{

cout << "Индекс элемента слишком велик!" << endl;

return;

}

}

Node\* toDelete = previous->next;

previous->next = toDelete->next;

delete toDelete;

toDelete = nullptr;

}

}

}

}

int GetIndexOfElement(Node\* head, char data)

{

cout << "GetIndexOfElement(" << data << ")" << endl;

cout << "Индекс элемента " << data << ": ";

bool check = true;

Node\* current = head;

int i = 0;

while ((current != nullptr) && (check == true))

{

if (current->data == data)

{

check = false;

break;

}

i++;

current = current->next;

}

if (check == true)

return printf("\rТакого элемента нет в списке!\n");

else

{

cout << i << endl;

return i;

}

}

char GetElementFromIndex(Node\* head, int index)

{

cout << "GetElementFromIndex(" << index << ")" << endl;

cout << "Элемент с индексом " << index << ": ";

if (head == nullptr)

return printf("\rСписок не содержит элементов!\n");

else

{

if (index < 0)

return printf("\rВведён отрицательный индекс!\n");

else

{

Node\* current = head;

for (int i = 0; i < index; i++)

{

current = current->next;

if (current == nullptr)

return printf("\rИндекс элемента слишком велик!\n");

}

cout << current->data << endl;

return current->data;

}

}

}

int GetSizeOfList(Node\* head)

{

cout << "Размер односвязного списка: ";

int Size = 0;

Node\* current = head;

while (current != nullptr)

{

Size++;

current = current->next;

}

cout << Size << endl;

return Size;

}

void Print(Node\* head)

{

cout << "Print" << endl;

if (head == nullptr)

cout << "Список не содержит элементов!" << endl;

Node\* current = head;

while (current != nullptr)

{

cout << current->data << endl;

current = current->next;

}

}

void clear(Node\*& head)

{

cout << "clear" << endl;

while (head != nullptr)

{

Node\* temp = head;

head = head->next;

delete temp;

temp = nullptr;

}

}

struct NodeT

{

char dataT = char();

NodeT\* nextT = nullptr;

NodeT\* prevT = nullptr;

};

void push\_front(NodeT\*& headT, NodeT\*& tailT, char data)

{

if (headT == nullptr)

{

NodeT\* tempT = new NodeT;

tempT->dataT = data;

tempT->nextT = headT;

tempT->prevT = headT;

headT = tempT;

tailT = tempT;

}

else

{

NodeT\* tempT = new NodeT;

tempT->dataT = data;

tempT->nextT = headT;

headT->prevT = tempT;

headT = tempT;

}

}

void push\_back(NodeT\*& headT, NodeT\*& tailT, char data)

{

if (headT == nullptr)

{

NodeT\* tempT = new NodeT;

tempT->dataT = data;

tempT->nextT = headT;

tempT->prevT = headT;

headT = tempT;

tailT = tempT;

}

else

{

NodeT\* tempT = new NodeT;

tempT->dataT = data;

tempT->prevT = tailT;

tailT->nextT = tempT;

tailT = tempT;

}

}

void insert(NodeT\*& headT, NodeT\*& tailT, char data, int index)

{

cout << "insert(" << data << "," << index << ")" << endl;

if (headT == nullptr)

push\_front(headT, tailT, data);

else

{

if (index < 0)

cout << "Введён отрицательный индекс!" << endl;

else

{

NodeT\* previousT = headT;

for (int i = 0; i < index; i++)

{

previousT = previousT->nextT;

if (previousT == nullptr)

{

cout << "Индекс элемента слишком велик!" << endl;

return;

}

}

if (previousT->nextT == nullptr)

push\_back(headT, tailT, data);

else

{

NodeT\* nextElementT = tailT;

while (nextElementT->prevT != previousT)

nextElementT = nextElementT->prevT;

NodeT\* newNodeT = new NodeT;

newNodeT->dataT = data;

newNodeT->nextT = nextElementT;

newNodeT->prevT = previousT;

previousT->nextT = newNodeT;

nextElementT->prevT = newNodeT;

}

}

}

}

void remove\_at(NodeT\*& headT, NodeT\* tailT, int index)

{

cout << "remove\_at(" << index << ")" << endl;

if (headT == nullptr)

cout << "В списке нет элементов для удаления!" << endl;

else

{

if (index == 0)

{

NodeT\* tempT = headT;

headT = headT->nextT;

delete tempT;

tempT = nullptr;

}

else

{

if (index < 0)

cout << "Введён неправильный индекс!" << endl;

else

{

NodeT\* previousT = headT;

for (int i = 0; i < index - 1; i++)

{

previousT = previousT->nextT;

if (previousT == nullptr)

{

cout << "Индекс элемента слишком велик!" << endl;

return;

}

}

if (previousT->nextT == nullptr)

{

NodeT\* toDeleteT = previousT;

tailT = previousT->prevT;

delete toDeleteT;

toDeleteT = nullptr;

}

else

{

NodeT\* nextElementT = tailT;

while (nextElementT->prevT != previousT->nextT)

nextElementT = nextElementT->prevT;

NodeT\* toDeleteT = previousT->nextT;

previousT->nextT = nextElementT;

nextElementT->prevT = previousT;

delete toDeleteT;

toDeleteT = nullptr;

}

}

}

}

}

int GetIndexOfElement(NodeT\* headT, char data)

{

cout << "GetIndexOfElement(" << data << ")" << endl;

cout << "Индекс элемента " << data << ": ";

bool check = true;

NodeT\* currentT = headT;

int i = 0;

while ((currentT != nullptr) && (check == true))

{

if (currentT->dataT == data)

{

check = false;

break;

}

i++;

currentT = currentT->nextT;

}

if (check == true)

return printf("\rТакого элемента нет в списке!\n");

else

{

cout << i << endl;

return i;

}

}

char GetElementFromIndex(NodeT\* headT, int index)

{

cout << "GetElementFromIndex(" << index << ")" << endl;

cout << "Элемент с индексом " << index << ": ";

if (headT == nullptr)

return printf("\rСписок не содержит элементов!\n");

else

{

if (index < 0)

return printf("\rВведён отрицательный индекс!\n");

else

{

NodeT\* currentT = headT;

for (int i = 0; i < index; i++)

{

currentT = currentT->nextT;

if (currentT == nullptr)

return printf("\rИндекс элемента слишком велик!\n");

}

cout << currentT->dataT << endl;

return currentT->dataT;

}

}

}

int GetSizeOfList(NodeT\* headT)

{

cout << "Размер двусвязного списка: ";

int SizeT = 0;

NodeT\* currentT = headT;

while (currentT != nullptr)

{

SizeT++;

currentT = currentT->nextT;

}

cout << SizeT << endl;

return SizeT;

}

void Print(NodeT\* headT)

{

cout << "Print" << endl;

if (headT == nullptr)

cout << "Список не содержит элементов!" << endl;

NodeT\* currentT = headT;

while (currentT != nullptr)

{

cout << currentT->dataT << endl;

currentT = currentT->nextT;

}

}

void clear(NodeT\*& headT)

{

cout << "clear" << endl;

while (headT != nullptr)

{

NodeT\* tempT = headT;

headT = headT->nextT;

delete tempT;

tempT = nullptr;

}

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Node\* head = nullptr;

cout << "-------------Односвязный список(первый список): " << endl;

push\_back(head, 'A');

push\_back(head, 'b');

push\_back(head, 'C');

push\_front(head, 'D');

push\_front(head, 'E');

push\_front(head, 'f');

Print(head);

insert(head, 'o', 5);

Print(head);

GetIndexOfElement(head, 'D');

NodeT\* headT = nullptr, \* tailT = nullptr;

cout << "\n-------------Двусвязный список(второй список): " << endl;

push\_back(headT, tailT, 'g');

push\_back(headT, tailT, 'h');

push\_back(headT, tailT, 'g');

push\_front(headT, tailT, 'k');

push\_front(headT, tailT, 'L');

push\_front(headT, tailT, 'm');

Print(headT);

GetElementFromIndex(headT, 5);

Node\* new\_head = nullptr;

cout << "\n-------------Односвязный список(третий список): " << endl;

int Size\_first = 3;

int Size\_second = 3;

for (int i = 0; i < Size\_first; i++)

{

char element = GetElementFromIndex(head, i);

push\_back(new\_head, element);

}

for (int i = 0; i < Size\_second; i++)

{

char element = GetElementFromIndex(headT, i);

push\_back(new\_head, element);

}

Print(new\_head);

cout << "Преобразование в нижний регистр : ";

string s = "fEDmLK";

cout << s << endl;

cout << "Результат преобразования в нижний регистр : ";

transform(s.begin(), s.end(), s.begin(), tolower);

cout << s << endl;

SetConsoleCP(866);

SetConsoleOutputCP(866);

system("pause");

return 0;

}