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

**Міністерство освіти і науки України**

**Національний технічний університет**

**«Дніпровська політехніка»**

****

**ЗВІТ**

**про виконання лабораторних робіт**

**з дисципліни**

# «Алгоритми та структури даних»

**Лабораторна робота № 7**

Виконав:

студент гр. 124-19-2

Моторний Андрій Сергійович

Прийняв:

викладач. каф. ФІТ

Сергеева Екатерина Леонидовна

**Дніпро**

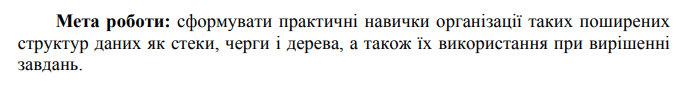
**2020**

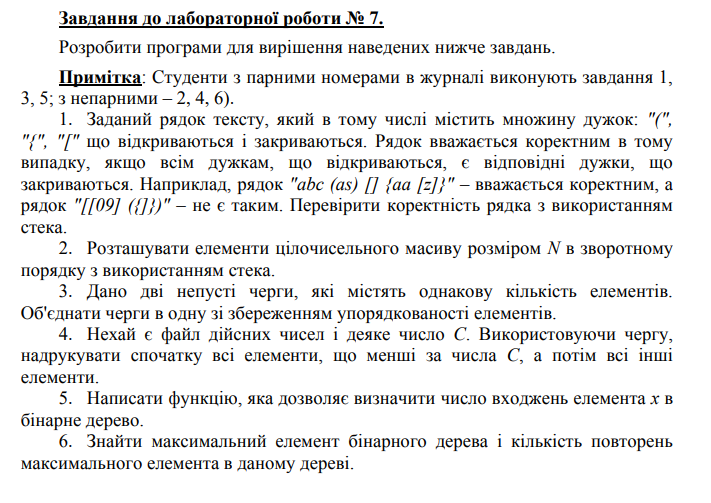
Звіт

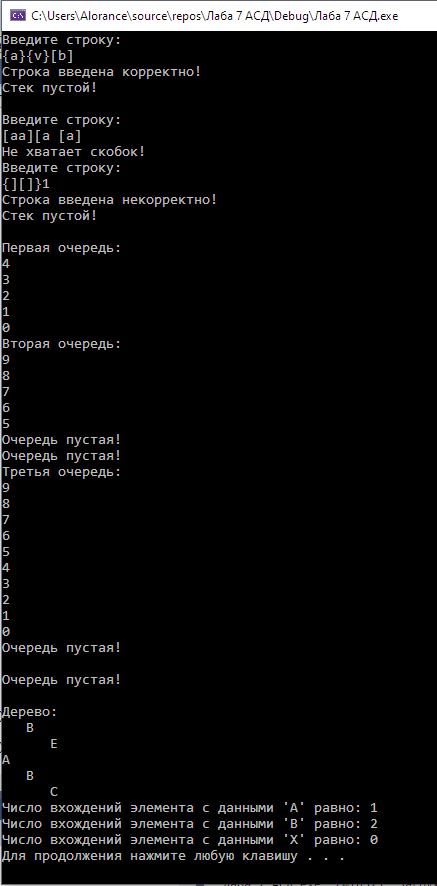
1)Лабораторна робота №7

## 2) Структура даних : стек, черга, дерева

3)Варіант №12







#include<iostream>

#include<Windows.h>

using namespace std;

enum Flag

{

Catch\_up,

Put\_down,

Not\_defined

};

struct Stack\_node

{

private:

char data;

Stack\_node\* next;

friend void push\_Stack(Stack\_node\*& head, char data);

friend int pop\_Stack(Stack\_node\*& head);

friend void clear\_Stack(Stack\_node\*& head);

friend void check\_String(Stack\_node\*& head, char\* str);

};

void push\_Stack(Stack\_node\*& head, char data)

{

Stack\_node\* New = new Stack\_node();

New->data = data;

New->next = head;

head = New;

}

int pop\_Stack(Stack\_node\*& head)

{

if (head == nullptr)

return printf("Стек пустой!\n");

else

{

char value;

Stack\_node\* toDelete = head;

head = toDelete->next;

value = toDelete->data;

delete toDelete;

return value;

}

}

void check\_String(Stack\_node\*& head, char\* str)

{

Flag check = Not\_defined;

int length = strlen(str);

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

{

if (str[i] == 40 || str[i] == 91 || str[i] == 123)

{

push\_Stack(head, str[i]);

}

else if (str[i] == 41)

{

char symbol = pop\_Stack(head);

if (symbol == 40)

check = Catch\_up;

else

{

check = Put\_down;

break;

}

}

else if (str[i] == 93)

{

char symbol = pop\_Stack(head);

if (symbol == 91)

check = Catch\_up;

else

{

check = Put\_down;

break;

}

}

else if (str[i] == 125)

{

char symbol = pop\_Stack(head);

if (symbol == 123)

check = Catch\_up;

else

{

check = Put\_down;

break;

}

}

}

if (head != nullptr)

{

cout << "Не хватает скобок!" << endl;

return;

}

if (check == Catch\_up)

cout << "Строка введена корректно!" << endl;

else if (check == Put\_down)

cout << "Строка введена некорректно!" << endl;

else

cout << "В строке отсутствуют скобки!" << endl;

}

void clear\_Stack(Stack\_node\*& head)

{

if (head == nullptr)

cout << "Стек пустой!\n" << endl;

else

{

Stack\_node\* toDelete = head;

while (head != nullptr)

{

head = toDelete->next;

delete toDelete;

}

}

}

struct Queue\_node

{

private:

char data;

Queue\_node\* next;

friend void push\_Queue(Queue\_node\*& first, Queue\_node\*& last, char data);

friend int pop\_Queue(Queue\_node\*& first, Queue\_node\*& last);

friend void Merge\_Queues(Queue\_node\*& New\_first, Queue\_node\*& New\_last, Queue\_node\*& first, Queue\_node\*& last, Queue\_node\*& first2, Queue\_node\*& last2);

friend void clear\_Queue(Queue\_node\*& first, Queue\_node\*& last);

friend void print\_Queue(Queue\_node\*& last);

};

void push\_Queue(Queue\_node\*& first, Queue\_node\*& last, char data)

{

if ((first == nullptr) || (last == nullptr))

{

Queue\_node\* New = new Queue\_node();

New->data = data;

New->next = last;

last = New;

first = last;

}

else

{

Queue\_node\* New = new Queue\_node();

New->data = data;

New->next = last;

last = New;

}

}

int pop\_Queue(Queue\_node\*& first, Queue\_node\*& last)

{

if (first == nullptr)

return printf("Очередь пустая!\n");

else

{

char value;

Queue\_node\* toDelete = first;

Queue\_node\* temp = last;

while ((temp->next != first) && (temp->next != nullptr))

temp = temp->next;

if (temp == first)

{

value = toDelete->data;

delete toDelete;

first->next = nullptr;

first = last = nullptr;

return value;

}

else

{

first = temp;

value = toDelete->data;

delete toDelete;

first->next = nullptr;

return value;

}

}

}

void Merge\_Queues(Queue\_node\*& New\_first, Queue\_node\*& New\_last, Queue\_node\*& first, Queue\_node\*& last, Queue\_node\*& first2, Queue\_node\*& last2)

{

char Data, Data2;

bool flag = false, flag2 = false;

while (!flag)

{

Data = pop\_Queue(first, last);

if (Data == 16)

flag = true;

else

push\_Queue(New\_first, New\_last, Data);

}

while (!flag2)

{

Data2 = pop\_Queue(first2, last2);

if (Data2 == 16)

flag2 = true;

else

push\_Queue(New\_first, New\_last, Data2);

}

}

void print\_Queue(Queue\_node\*& last)

{

if (last == nullptr)

cout << " " << endl;

else

{

Queue\_node\* temp = last;

while (temp != nullptr)

{

cout << temp->data << endl;

temp = temp->next;

}

}

}

void clear\_Queue(Queue\_node\*& first, Queue\_node\*& last)

{

if (last == nullptr)

cout << "Очередь пустая!\n" << endl;

else

{

Queue\_node\* toDelete = new Queue\_node();

while (last != nullptr)

{

toDelete = last;

last = toDelete->next;

delete toDelete;

}

first = nullptr;

}

}

struct Tree\_node

{

private:

char data;

Tree\_node\* left, \* right;

friend Tree\_node\* create\_tree(int size, char\* arr, int& index);

friend void distruct(Tree\_node\*& root);

friend void print\_Tree(Tree\_node\*& root, int number);

friend int search\_entries(Tree\_node\*& root, int& quantity, char data);

};

Tree\_node\* create\_tree(int size, char\* arr, int& index)

{

if (size <= 0)

return nullptr;

else

{

int size\_left, size\_right;

Tree\_node\* new\_root = new Tree\_node();

size\_left = size / 2;

size\_right = size - size\_left - 1;

new\_root->data = arr[index];

index++;

new\_root->left = create\_tree(size\_left, arr, index);

new\_root->right = create\_tree(size\_right, arr, index);

return new\_root;

}

}

void distruct(Tree\_node\*& root)

{

if (root != nullptr)

{

distruct(root->left);

distruct(root->right);

delete root;

root = nullptr;

}

}

void print\_Tree(Tree\_node\*& root, int number)

{

if (root != nullptr)

{

print\_Tree(root->right, number + 3);

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

printf(" ");

printf("%c \n", root->data);

print\_Tree(root->left, number + 3);

}

}

int search\_entries(Tree\_node\*& root, int& quantity, char data)

{

if (root != nullptr)

{

search\_entries(root->left, quantity, data);

if (root->data == data)

quantity++;

search\_entries(root->right, quantity, data);

return quantity;

}

else

return 0;

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Stack\_node\* head = nullptr;

char str[100];

for (int i = 0; i < 3; i++)

{

cout << "Введите строку:" << endl;

gets\_s(str);

check\_String(head, str);

clear\_Stack(head);

}

Queue\_node\* first = nullptr, \* last = nullptr;

Queue\_node\* first2 = nullptr, \* last2 = nullptr;

for (char i = 48; i < 53; i++)

push\_Queue(first, last, i);

cout << "Первая очередь:" << endl;

print\_Queue(last);

for (char i = 53; i < 58; i++)

push\_Queue(first2, last2, i);

cout << "Вторая очередь:" << endl;

print\_Queue(last2);

Queue\_node\* New\_first = nullptr;

Queue\_node\* New\_last = nullptr;

Merge\_Queues(New\_first, New\_last, first, last, first2, last2);

cout << "Третья очередь:" << endl;

print\_Queue(New\_last);

clear\_Queue(first, last);

clear\_Queue(first2, last2);

clear\_Queue(New\_first, New\_last);

Tree\_node\* root = nullptr;

const int n = 6;

char ARR[] = { 'A','B','C','B','E', '\0' };

int i = 0;

root = create\_tree(n - 1, ARR, i);

cout << "Дерево:" << endl;

print\_Tree(root, 0);

int Quantity = 0;

cout << "Число вхождений элемента с данными 'A' равно: " << search\_entries(root, Quantity, 'A') << endl;

Quantity = 0;

cout << "Число вхождений элемента с данными 'B' равно: " << search\_entries(root, Quantity, 'B') << endl;

Quantity = 0;

cout << "Число вхождений элемента с данными 'X' равно: " << search\_entries(root, Quantity, 'X') << endl;

distruct(root);

SetConsoleCP(866);

SetConsoleOutputCP(866);

system("pause");

return 0;

}