**ЛАБОРАТОРНА РОБОТА №13**

## Програмування задач з використанням лінійних списків : стека, черги.

**Мета:**

* Ознайомлення з динамічною структурою даних -. стек, черга.
* Отримання навиків роботи з змінними вказівного типу.
* Ознайомлення з можливістю виконання операції, включення елементів в стек, чергу та їх виключення зі вказаного списку.

Порядок виконання роботи

**Оформити звіт:**

1. Тема.
2. Мета.
3. Постановка задачі.
4. Програма.
5. Результати роботи.
6. Висновки.

**Задача 1 завдання 13**

**Умови завдання:**

1. Створити список з іменами хлопчиків та дівчаток. Створити файл з лічилками. Обрати у довільний спосіб одну з лічилок і полічити дітей. Вивести на екран імена дітей з відповідним словом лічилки. Вилучити зі списку ім’я того, хто вибув.

Дана програма буде складатися з основної функції int main() та 10 функцій

**3)Лістинг програми;**

#include <iostream>

#include <iomanip>

#include <fstream>

#include <string>

#include <sstream>

#include <ctime>

#include <cstdlib>

#include <Windows.h>

using namespace std;

struct Node

{

string data;

Node\* next;

};

Node\* Head = nullptr;

Node\* tail = nullptr;

Node\* boys = nullptr;

Node\* girls = nullptr;

void Show\_definetly\_value\_of\_list(Node\* node, int numder);

void addNode(Node\*& head, Node\*& tail, string name);

void removeNode(Node\*\* node, int index);

void Create\_list\_and\_Choise\_list();

int find\_kil\_words(string filed);

void show\_licilka(string filed);

void Create\_List\_Boys\_Girls();

int kil\_of\_people(Node\* node);

void Delete\_List(Node\* node);

//void Aplly\_file\_lichilks();

void Show\_list(Node\* node);

int main() {

setlocale(LC\_ALL, "ukr");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

bool Index = true;

//Node\* head = NULL;

//Aplly\_file\_lichilks();

int P = 0;

Create\_List\_Boys\_Girls();

Create\_list\_and\_Choise\_list();

Delete\_List(Head);

cout << endl << "Список був видалений" << endl;

}

void Create\_list\_and\_Choise\_list()

{

ifstream File("file.txt");

int choice;

const int max\_kil\_of\_Lichilk = 10;

string counters[max\_kil\_of\_Lichilk];

string counter;

int kil\_of\_Lichilk = 0;

while (getline(File, counter) && kil\_of\_Lichilk < max\_kil\_of\_Lichilk) {

counters[kil\_of\_Lichilk] = counter;

kil\_of\_Lichilk++;

}

File.close();

cout << "Виберіть номер лічилки ( пам'ятайте про кількість ліилок = " << kil\_of\_Lichilk << " ):" << endl;

cin >> choice;

if (choice >= 1 && choice <= kil\_of\_Lichilk) {

cout << "Обрана лічилка: " << endl;

cout << counters[choice - 1] << endl;

}

else

{

cout << "Введіть коректрне значення";

}

string filed = counters[choice - 1];

int kil\_sign\_before\_last\_word = 0;

int kil\_of\_words = find\_kil\_words(filed);

cout << "Кількість слів - " << kil\_of\_words << endl;

int kil\_people = kil\_of\_people(Head);

cout << "Кількість людей - " << kil\_people << endl;

int index\_number\_name, k = 0;

for (int i = 0; filed[i] != '\0'; i++)

{

if (filed[i] == ' ')

{

k++;

}

if (k != (kil\_of\_words - 1))

{

kil\_sign\_before\_last\_word++;

}

}

Node\* current = Head;

int i;

while (kil\_people != 1)

{

show\_licilka(filed);

if (Head != NULL) {

i = 0;

do {

if (i % kil\_people == 0 && i != 0)

{

current = Head;

}

cout << current->data << setw(8);

current = current->next;

i++;

} while (i != kil\_of\_words);

}

current = Head;

//виводимо список

cout << endl;

for (int i = 0; filed[i] != '\0'; i++)

{

if (i >= kil\_sign\_before\_last\_word)

{

cout << filed[i];

}

}

cout << " - ";

if (kil\_of\_words % kil\_people == 0)

{

index\_number\_name = (kil\_people - 1);

}

else

{

index\_number\_name = (kil\_of\_words % kil\_people) - 1;

}

Show\_definetly\_value\_of\_list(Head, index\_number\_name);

cout << "- вилучено";

removeNode(&Head, index\_number\_name);

kil\_people--;

cout << endl;

}

cout << "Залишився: " << Head->data;

}

void removeNode(Node\*\* node, int index)

{

if (\*node == nullptr) {

return;

}

int size = 0;

Node\* current = \*node;

Node\* prev = nullptr;

while (size < index) {

prev = current;

current = current->next;

size++;

if (current == \*node) {

return;

}

}

if (current == \*node) {

\*node = current->next;

}

if (prev != nullptr) {

prev->next = current->next;

}

delete current;

}

void Create\_List\_Boys\_Girls()

{

int N;

cout << "Введи кількість людей: "; cin >> N;

if (N <= 0)

{

cout << "Невірне число" << endl;

return;

}

int p;

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

{

cout << "1 - хлопці, 0 - дівчата" << endl;

cin >> p;

string field;

if (p == 1)

{

cout << i + 1 << "-й" << endl;

cout << "Ім'я: ";

cin.ignore();

cin >> field;

addNode(Head, tail, field);

}

else

{

cout << i + 1 << "-й" << endl;

cout << "Ім'я: ";

cin.ignore();

cin >> field;

addNode(Head, tail, field);

}

}

}

void addNode(Node \*&node, Node \*&tail, string name)

{

Node \*temp = new Node;

temp->data = name;

temp->next = NULL;

if (node == NULL) {

node = temp;

tail = temp;

temp->next = node;

}

else {

tail->next = temp;

tail = temp;

tail->next = node;

}

}

void Delete\_List(Node\* node)

{

if(node == nullptr) {

return;

}

delete node;

}

int kil\_of\_people(Node\* node)

{

int i = 0;

Node\* current = node;

do {

i++;

current = current->next;

} while (current != node);

return i;

}

void show\_licilka(string filed)

{

for (int i = 0; filed[i] != '\0'; i++)

{

if (filed[i] == ' ')

{

cout << setw(5);

}

else cout << filed[i];

}

cout << endl;

}

int find\_kil\_words(string filed)

{

int kil\_words = 0;

for (int i = 0; filed[i] != '\0'; i++)

{

if (filed[i] == ' ')

{

kil\_words++;

}

}

kil\_words++;

return kil\_words;

}

void Show\_list(Node\* node)

{

Node\* current = Head;

if (Head == NULL) {

cout << "Cписок порожній" << endl;

return;

}

do {

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

current = current->next;

} while (current != Head);

}

void Show\_definetly\_value\_of\_list(Node\* node, int numder)

{

Node\* cur = Head;

if (Head == NULL) {

cout << "Cписок порожній" << endl;

return;

}

int i = 0;

do {

if (i == numder)

{

cout << cur->data;

}

cur = cur->next;

i++;

} while (cur != Head);

}

//void Aplly\_file\_lichilks()

//{

// ofstream File("file.txt", ios::out);

// File << "Раз два три чотири Кицю грамоті навчили Не читати не писати А за мишками ганяти" << endl;

// File << "Бігли коні під мостами З золотими копитами Дзень-брязь Вийшов князь." << endl;

// File << "Сів метелик на травичку Пригадав собі дурничку Раз два три Ти метелику лети!" << endl;

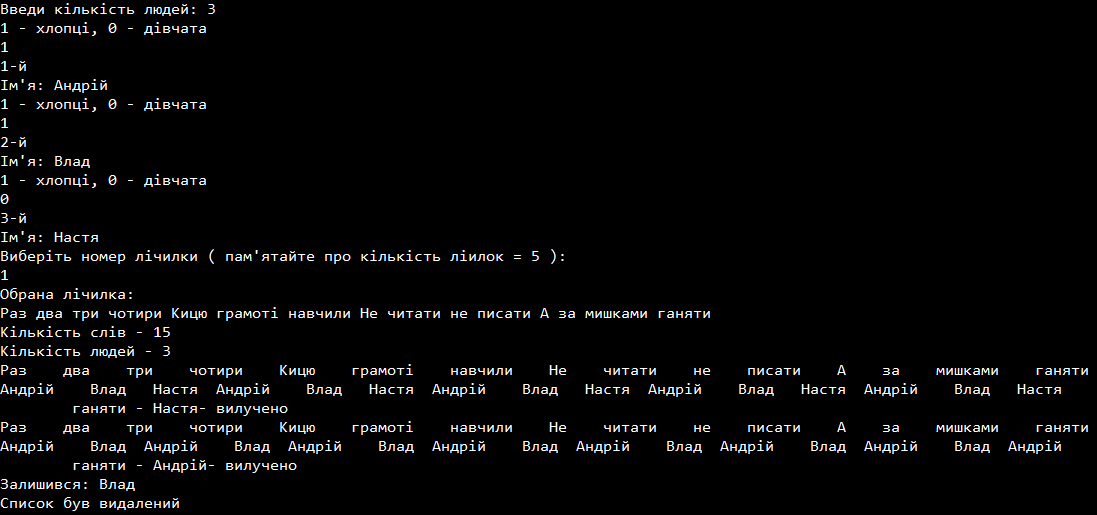
// File << "Сорока-ворона На дубі сиділа Боярів лічила Калина-малина Суниця-чорниця В’яз Хто буде жмуриться Той буде князь" << endl;

// File << "Ходила квочка коло кілочка Водила діточок біля квіточок! Квок!" << endl;

// File.close();

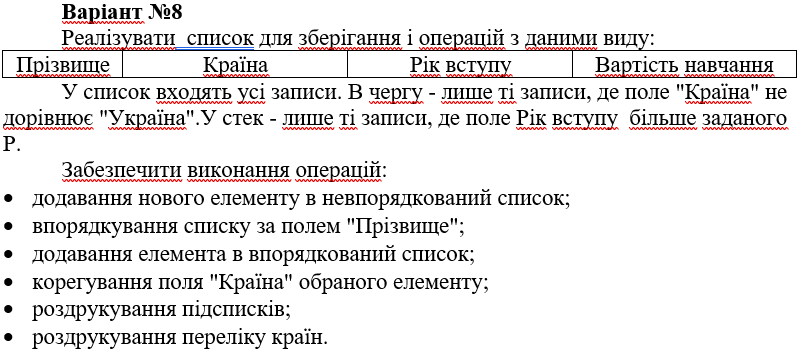
//}

**5)Результати**



**Задача 2 завдання 8**

**Умови завдань:**

****

**3)Лістинг програми;**

#include <iostream>

#include <iomanip>

#include <set>

#include <string>

#include <iomanip>

#include <algorithm>

#include <Windows.h>

const int WIDTH = 15;

const int MAX\_YEAR = 2010;

using namespace std;

struct Student {

string name;

string country;

int year;

double cost;

};

struct Node {

Student data;

Node\* next;

Node(Student data) : data(data), next(nullptr) {}

};

struct Queue {

Node\* head;

Node\* tail;

Queue() : head(nullptr), tail(nullptr) {}

void push(Student data) {

Node\* newNode = new Node(Student(data));

if (tail == nullptr) {

head = newNode;

tail = newNode;

}

else {

tail->next = newNode;

tail = newNode;

}

}

void pop() {

if (empty()) {

cout << "Queue is empty\n";

}

Node\* temp = head;

head = head->next;

delete temp;

if (empty()) {

tail = nullptr;

}

}

Student& front() {

if (empty()) {

cout << "Queue is empty\n";

}

return head->data;

}

bool empty() {

return head == nullptr;

}

};

struct Stack {

Stack() : top(nullptr) {}

Node\* top;

void push(Student data) {

Node\* newNode = new Node(data);

newNode->next = top;

top = newNode;

}

void pop() {

if (empty()) {

cout << "Stack is empty\n";

}

Node\* temp = top;

top = top->next;

delete temp;

}

Student& getTop() {

if (empty()) {

cout << "Stack is empty\n";

}

return top->data;

}

bool empty() {

return top == nullptr;

}

};

struct Vector {

Node\* head;

int size;

Vector() : head(nullptr), size(0) {}

void push\_back(Student data) {

Node\* newNode = new Node(Student(data));

if (head == nullptr) {

head = newNode;

}

else {

Node\* current = head;

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

}

size++;

}

Student& operator[](int index) {

if (index < 0 || index >= size) {

cout << "Index non found\n";

}

Node\* current = head;

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

current = current->next;

}

return current->data;

}

int getSize() {

return size;

}

};

int getIndexOfCountryToChange();

string getNewCountry();

Student inputStudent();

void printAllCounties(const set<string>& allCountries);

void printList(Vector& list);

void printStack(Stack stack);

void printQueue(Queue queue);

void printHeadLine();

void changeCountry(const int& index, const std::string& newCountry, Vector& list);

void addStudentSorted(const Student& student, Vector& list);

void sortByName(Vector& list);

void print(Vector list);

void printStudent(const Student& student);

void addStudent(const Student& student, Vector& list, Queue& queue, Stack& stack);

int getOption();

void showPosibilities();

int main()

{

setlocale(LC\_ALL, "ukr");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Vector list;

Queue queue;

Stack stack;

int option;

do {

option = getOption();

switch (option) {

case 1: {

Student student = inputStudent();

addStudent(student, list, queue, stack);

break;

}

case 2: {

sortByName(list);

break;

}

case 3: {

printList(list);

break;

}

case 4: {

Student student = inputStudent();

addStudentSorted(student, list);

break;

}

case 5: {

int index = getIndexOfCountryToChange();

std::string newCountry = getNewCountry();

changeCountry(index, newCountry, list);

break;

}

case 6: {

if (list.getSize() == 0) {

cout << "No students in list\n" << std::endl;

}

else {

printList(list);

if (queue.empty()) {

cout << "No students in queue\n";

}

else {

printQueue(queue);

}

if (stack.empty()) {

cout << "No students in stack\n";

}

else {

printStack(stack);

}

}

break;

}

case 7: {

set<string> allCountries = getCountries(list);

printAllCounties(allCountries);

break;

}

case 8: {

cout << "Quitting program.\n";

break;

}

default: {

cout << "Invalid choice. Please try again.\n";

break;

}

}

} while (option != 8);

}

void showPosibilities() {

cout << "1. Add student\n"

<< "2. Sort list by name\n"

<< "3. Print list\n"

<< "4. Add student to sorted list\n"

<< "5. Change country of selected student\n"

<< "6. Print sub-lists\n"

<< "7. Print countries\n"

<< "8. Quit\n"

<< "Choose an option:\n";

}

int getOption() {

int option;

showPosibilities();

cin >> option;

return option;

}

void addStudent(const Student& student, Vector& list, Queue& queue, Stack& stack) {

list.push\_back(student);

if (student.country != "Ukraine") {

queue.push(student);

}

if (student.year > MAX\_YEAR) {

stack.push(student);

}

}

void printStudent(const Student& student) {

cout << setw(WIDTH) << student.name << setw(WIDTH)

<< student.country << setw(WIDTH) << student.year

<< setw(WIDTH) << student.cost << '\n';

}

void print(Vector list) {

for (int student = 0; student < list.getSize(); student++) {

printStudent(list[student]);

}

}

void sortByName(Vector& list)

{

for (int i = 0; i < list.getSize() - 1; i++) {

for (int j = 0; j < list.getSize() - i - 1; j++) {

if (list[j].name > list[j + 1].name) {

std::swap(list[j], list[j + 1]);

}

}

}

}

void addStudentSorted(const Student& student, Vector& list)

{

Node\* current = list.head;

Node\* prev = nullptr;

while (current != nullptr && current->data.name < student.name) {

prev = current;

current = current->next;

}

if (prev == nullptr) {

list.push\_back(student);

}

else {

prev->next = new Node(student);

prev->next->next = current;

list.size++;

}

}

void changeCountry(const int& index, const std::string& newCountry, Vector& list) {

list[index].country = newCountry;

}

void printHeadLine() {

cout << setw(WIDTH) << "Name" << setw(WIDTH) << "Country" << setw(WIDTH) << "Year" << setw(WIDTH) << "Cost" << endl;

}

void printQueue(Queue queue) {

std::cout << "Printing sublist of students not from Ukraine:\n";

printHeadLine();

while (!queue.empty()) {

Student student = queue.front();

printStudent(student);

queue.pop();

}

}

void printStack(Stack stack) {

cout << "Printing sublist of students who started after " << MAX\_YEAR << ":\n";

printHeadLine();

while (!stack.empty()) {

Student student = stack.getTop();

printStudent(student);

stack.pop();

}

}

void printList(Vector& list) {

cout << "ALL LIST\n";

printHeadLine();

for (int i = 0; i < list.getSize(); i++) {

printStudent(list[i]);

}

}

set<string> getCountries(Vector list) {

set<string> allCountries;

for (int i = 0; i < list.getSize(); i++) {

allCountries.insert(list[i].country);

}

return allCountries;

}

void printAllCounties(const set<string>& allCountries) {

cout << "Printing list of countries:\n";

for (const auto& country : allCountries) {

cout << country << '\n';

}

}

Student inputStudent() {

Student student;

cout << "Enter name: ";

cin >> student.name;

cout << "Enter country: ";

cin >> student.country;

cout << "Enter year: ";

cin >> student.year;

cout << "Enter cost: ";

cin >> student.cost;

return student;

}

string getNewCountry() {

string newCountry;

cout << "Enter new country: ";

cin >> newCountry;

return newCountry;

}

int getIndexOfCountryToChange() {

int index;

cout << "Enter index of coutry to change: ";

cin >> index;

return index;

}**5)Результати**

1. Add student

2. Sort list by name

3. Print list

4. Add student to sorted list

5. Change country of selected student

6. Print sub-lists

7. Print countries

8. Quit

Choose an option:

1

Enter name: Andrii

Enter country: Ukraine

Enter year: 2001

Enter cost: 1000

1. Add student

2. Sort list by name

3. Print list

4. Add student to sorted list

5. Change country of selected student

6. Print sub-lists

7. Print countries

8. Quit

Choose an option:

1

Enter name: Paul

Enter country: Germany

Enter year: 2015

Enter cost: 3000

1. Add student

2. Sort list by name

3. Print list

4. Add student to sorted list

5. Change country of selected student

6. Print sub-lists

7. Print countries

8. Quit

Choose an option:

1

Enter name: Anna

Enter country: USA

Enter year: 2020

Enter cost: 500

1. Add student

2. Sort list by name

3. Print list

4. Add student to sorted list

5. Change country of selected student

6. Print sub-lists

7. Print countries

8. Quit

Choose an option:

3

ALL LIST

Name Country Year Cost

Andrii Ukraine 2001 1000

Paul Germany 2015 3000

Anna USA 2020 500

1. Add student

2. Sort list by name

3. Print list

4. Add student to sorted list

5. Change country of selected student

6. Print sub-lists

7. Print countries

8. Quit

Choose an option:

6

ALL LIST

Name Country Year Cost

Andrii Ukraine 2001 1000

Paul Germany 2015 3000

Anna USA 2020 500

Printing sublist of students not from Ukraine:

Name Country Year Cost

Paul Germany 2015 3000

Anna USA 2020 500

Printing sublist of students who started after 2010:

Name Country Year Cost

Anna USA 2020 500

Paul Germany 2015 3000

1. Add student

2. Sort list by name

3. Print list

4. Add student to sorted list

5. Change country of selected student

6. Print sub-lists

7. Print countries

8. Quit

Choose an option:

7

Printing list of countries:

Germany

USA

Ukraine

1. Add student

2. Sort list by name

3. Print list

4. Add student to sorted list

5. Change country of selected student

6. Print sub-lists

7. Print countries

8. Quit

Choose an option:

2

1. Add student

2. Sort list by name

3. Print list

4. Add student to sorted list

5. Change country of selected student

6. Print sub-lists

7. Print countries

8. Quit

Choose an option:

2

1. Add student

2. Sort list by name

3. Print list

4. Add student to sorted list

5. Change country of selected student

6. Print sub-lists

7. Print countries

8. Quit

Choose an option:

3

ALL LIST

Name Country Year Cost

Andrii Ukraine 2001 1000

Anna USA 2020 500

Paul Germany 2015 3000

1. Add student

2. Sort list by name

3. Print list

4. Add student to sorted list

5. Change country of selected student

6. Print sub-lists

7. Print countries

8. Quit

Choose an option:

8

Quitting program.

**Висновок:** Ознайомився з динамічною структурою даних: стек, черга. Отримав навички роботи з змінними вказівного типу. Ознайомився з можливістю виконання операції, включення елементів в стек, чергу та їх виключення зі вказаного списку.