**Міністерство освіти і науки України**

**Національний університет «Запорізька Політехніка»**

Кафедра програмних засобів

**ЗВІТ**

з лабораторної роботи №7

з дисципліни «Об’єктно-орієнтоване програмування»

На тему «ВВЕДЕННЯ В УЗАГАЛЬНЕНЕ ПРОГРАМУВАННЯ»

Варіант №20

**Виконав:**

Студент групи КНТ-122 О. А. Онищенко

**Прийняли:**

Ст. Викладач Т. В. Голуб

Ст. Викладач Л. Ю. Дейнега

2023

[Мета роботи 3](#_Toc133221102)

[Текст завдання №5 3](#_Toc133221103)

[1) Код програми – main.cpp 3](#_Toc133221104)

[1) Код програми – sup.h 5](#_Toc133221105)

[1) Приклад роботи 9](#_Toc133221106)

[2) Код програми – main.cpp 11](#_Toc133221107)

[2) Код програми – sup.h 12](#_Toc133221108)

[2) Приклад роботи 19](#_Toc133221109)

[3) Код програми – main.cpp 21](#_Toc133221110)

[3) Код програми – sup.h 22](#_Toc133221111)

[3) Приклад роботи 27](#_Toc133221112)

[Код бібліотеки – lib.h 29](#_Toc133221113)

[Висновки 36](#_Toc133221114)

Мета роботи

Навчитись використовувати шаблони при створені програм мовою програмування С++.

Текст завдання №5

1. Виконати завдання з лабораторної роботи №1, де тип елементу заданої структури даних довільний. Використати шаблонні функції.
2. Виконати завдання з лабораторної роботи №5 з довільним типом даних. Використати шаблонні класи відповідно до завдання викладача.
3. Виконати завдання з лабораторної роботи №1 використавши, для зберігання даних класи Standard Template Library (STL) або list або vector. Поясніть різницю в використанні цих класів.

1) Код програми – main.cpp

#include "D:\repos\university\lib.h"

#include "sup.h"

// Створити клас Delta таким чином, щоб кожний об’єкт вміщував свій персональний номер (дескриптор об’єкта) та функцію, яка повертає його значення. Дескриптор об’єкта – унікальне для об’єктів даного типу ціле число.

// Виконати завдання з лабораторної роботи №1, де тип елементу заданої структури даних довільний. Використати шаблонні функції.

int main()

{

    srand(time(NULL));

    char doContinue;

    char doReturnToMenu;

    cout << "\n";

    do

    {

        ///////////////////////////////////////

        vector<string> menuItems = {

            "Integer",

            "Double",

            "String",

            "Exit"};

        ll userDecision = showMenu(menuItems);

        if (userDecision == 1)

        {

            vector<unique\_ptr<Delta<int>>> deltaObjectsVector;

            outputMenu(deltaObjectsVector);

        }

        else if (userDecision == 2)

        {

            vector<unique\_ptr<Delta<double>>> deltaObjectsVector;

            outputMenu(deltaObjectsVector);

        }

        else if (userDecision == 3)

        {

            vector<unique\_ptr<Delta<string>>> deltaObjectsVector;

            outputMenu(deltaObjectsVector);

        }

        ///////////////////////////////////////

        cout << "\nWould you like to continue program execution? (Y | N): ";

        cin >> doContinue;

        if (doContinue == 'Y' || doContinue == 'y')

        {

            cout << "\n\n";

            continue;

        }

        else

            break;

    } while (doContinue = 'Y' || doContinue == 'y');

    cout << "\nThanks for using this program\n\n";

    return 0;

}

1) Код програми – sup.h

#include "sup.h"

#include "D:\repos\university\lib.h"

// Створити клас Delta таким чином, щоб кожний об’єкт вміщував свій персональний номер (дескриптор об’єкта) та функцію, яка повертає його значення. Дескриптор об’єкта – унікальне для об’єктів даного типу ціле число.

// Виконати завдання з лабораторної роботи №1, де тип елементу заданої структури даних довільний. Використати шаблонні функції.

template <typename T>

class Delta

{

private:

    ll descriptor;

    T value;

public:

    Delta() : value(), descriptor(nextDescriptor()) {}

    Delta(T inValue) : value(inValue), descriptor(nextDescriptor()) {}

    static ll nextDescriptor()

    {

        static ll nextDescriptor = 0;

        return nextDescriptor++;

    }

    T getValue() const { return value; }

    ll getDescriptor() const { return descriptor; }

    void setValue(T inValue) { value = inValue; }

    ~Delta() {}

};

template <typename T>

void createObjects(vector<unique\_ptr<Delta<T>>> &container)

{

    ll initSize = container.size();

    cout << "Enter number of objects to create: ";

    ll objectsAmount = getNum();

    if (objectsAmount < 1)

    {

        bad("Invalid amount of objects");

        return;

    }

    cout << "\nAdding objects (" << objectsAmount << "):\n";

    for (ll i = 0; i < objectsAmount; i++)

    {

        cin.ignore();

        T value;

        cout << i + 1 << ". Enter value for object: ";

        cin >> value;

        container.eb(make\_unique<Delta<T>>(value));

    }

    if (container.size() == initSize + objectsAmount)

        good("Objects successfully added");

    else

        bad("Failed to add objects");

    cout << endl;

    printObjects(container);

}

template <typename T>

void printObjects(vector<unique\_ptr<Delta<T>>> &container)

{

    ll vectorSize = container.size();

    if (vectorSize == 0)

    {

        bad("No objects to print");

        return;

    }

    cout << BOLD << "\nYour objects (" << vectorSize << "): \n"

         << UNBOLD;

    for (ll i = 0; i < vectorSize; i++)

    {

        cout << container[i]->getDescriptor() << ". " << container[i]->getValue() << endl;

    }

}

template <typename T>

void deleteObjects(vector<unique\_ptr<Delta<T>>> &container)

{

    ll initSize = container.size();

    if (initSize == 0)

    {

        bad("No objects to delete");

        return;

    }

    cout << endl;

    printObjects(container);

    cout << endl;

    cout << "Enter a number of object to delete: ";

    ll numToDelete = getNum();

    if (numToDelete >= initSize || numToDelete < 0)

    {

        bad("Invalid object number");

        return;

    }

    container.erase(container.begin() + numToDelete);

    cout << endl;

    if (container.size() == initSize - 1)

        good("Object successfully deleted");

    else

        bad("Failed to delete object");

    cout << endl;

    printObjects(container);

}

template <typename T>

void outputMenu(vector<unique\_ptr<Delta<T>>> &container)

{

    char doContinue;

    do

    {

        vector<string> menuItems = {

            "Add objects",

            "Delete objects",

            "Print objects",

            "Exit"};

        ll userDecision = showMenu(menuItems);

        if (userDecision == 1)

        {

            createObjects(container);

        }

        else if (userDecision == 2)

        {

            deleteObjects(container);

        }

        else if (userDecision == 3)

            printObjects(container);

        cout << "\nWould you like to return to menu? (Y | N): ";

        cin >> doContinue;

        if (doContinue == 'y' || doContinue == 'Y')

        {

            cout << endl;

            continue;

        }

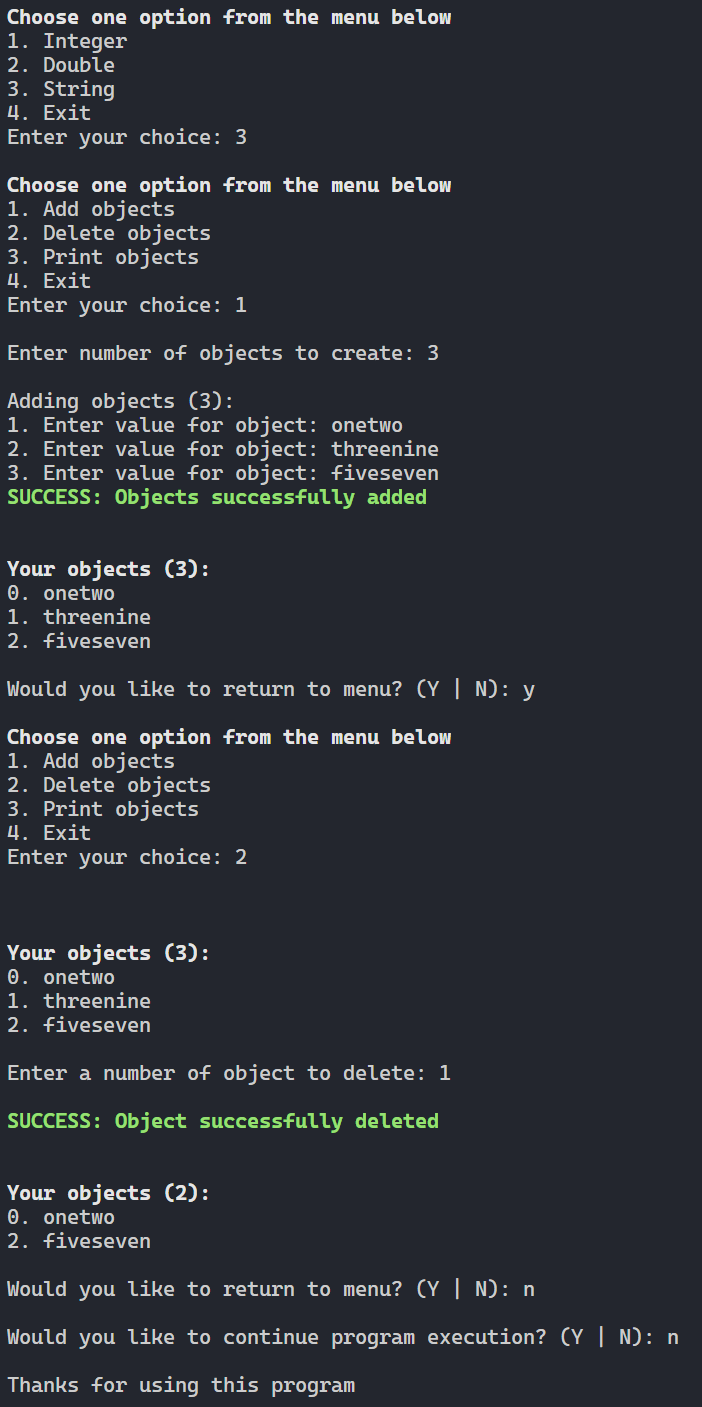
        else

            break;

    } while (doContinue == 'y' || doContinue == 'Y');

}

1) Приклад роботи



2) Код програми – main.cpp

#include "D:\repos\university\lib.h"

#include "sup.h"

using namespace std;

// Виконати завдання з лабораторної роботи #5 з довільним типом даних

int main()

{

    srand(time(NULL));

    char doContinue;

    vector<unique\_ptr<DynamicString>> container;

    char doReturnToMenu;

    cout << "\n";

    do

    {

        ///////////////////////////////////////

        do

        {

            outputMenu(container);

            cout << "\nWould you like to return to menu? (Y | N): ";

            cin >> doReturnToMenu;

            if (doReturnToMenu == 'Y' || doReturnToMenu == 'y')

            {

                cout << endl

                     << endl;

                continue;

            }

            else

                break;

        } while (doReturnToMenu == 'y' || doReturnToMenu == 'Y');

        ///////////////////////////////////////

        cout << "\nWould you like to continue program execution? (Y | N): ";

        cin >> doContinue;

        if (doContinue == 'Y' || doContinue == 'y')

        {

            cout << "\n\n";

            continue;

        }

        else

            break;

    } while (doContinue = 'Y' || doContinue == 'y');

    cout << "\nThanks for using this program\n\n";

    return 0;

}

2) Код програми – sup.h

#include "sup.h"

#include "D:\repos\university\lib.h"

// Виконати завдання з лабораторної роботи #5 з довільним типом даних

const string ROOT\_PATH = "D:/repos/university/year1-term2/OOP/lb7/tsk2/";

class DynamicString

{

private:

    string m\_value;

    ll m\_size;

public:

    DynamicString() : m\_value(""), m\_size(0) {}

    DynamicString(string value) : m\_value(value), m\_size(value.length()) {}

    void setValue(string value)

    {

        m\_value = value;

        m\_size = value.length();

    }

    string getValue() const { return m\_value; }

    ll getSize() const { return m\_size; }

    void m\_reverse() { reverse(m\_value.begin(), m\_value.end()); }

    void m\_replace(string replaceThis, string withThis) { m\_value.replace(m\_value.find(replaceThis), replaceThis.length(), withThis); }

    void m\_remove\_spaces()

    {

        stringstream ss(m\_value);

        string word;

        m\_value.clear();

        while (ss >> word)

            m\_value += word + " ";

        if (!m\_value.empty())

            m\_value.pop\_back();

    }

    void m\_to\_upper() { transform(m\_value.begin(), m\_value.end(), m\_value.begin(), ::toupper); }

    void m\_to\_lower() { transform(m\_value.begin(), m\_value.end(), m\_value.begin(), ::tolower); }

    ~DynamicString() {}

};

void showStrings(vector<unique\_ptr<DynamicString>> &container)

{

    ll stringsNum = container.size();

    if (stringsNum == 0)

    {

        bad("No strings found");

        return;

    }

    cout << "Available strings (" << stringsNum << "):\n";

    for (ll i = 0; i < stringsNum; i++)

    {

        cout << i + 1 << ". " << container[i]->getValue() << " - " << container[i]->getSize() << " symbols\n";

    }

}

void showStrings(vector<unique\_ptr<DynamicString>> &container, const string &FILE)

{

    ll stringsNum = container.size();

    if (stringsNum == 0)

    {

        bad("No strings found");

        return;

    }

    ofstream file(FILE);

    if (!file.is\_open())

    {

        bad("File could not be opened");

        return;

    }

    file << "==============================\n\n";

    file << "Available strings (" << stringsNum << "):\n";

    for (ll i = 0; i < stringsNum; i++)

    {

        file << i + 1 << ". " << container[i]->getValue() << " - " << container[i]->getSize() << " symbols\n";

    }

    file << "\n==============================\n\n";

    file.close();

    if (file.good())

        good("Strings succesfully saved");

    else

        bad("Strings were not saved");

}

void addStrings(vector<unique\_ptr<DynamicString>> &container)

{

    ll initSize = container.size();

    cout << "Enter number of strings to add: ";

    ll numToAdd = getNum();

    cout << endl;

    cin.ignore();

    if (numToAdd == 0)

    {

        bad("Enter a valid number of strings");

        return;

    }

    for (ll i = 0; i < numToAdd; i++)

    {

        string value;

        cout << i + 1 << ". Enter value: ";

        getline(cin, value);

        container.push\_back(make\_unique<DynamicString>(value));

    }

    cout << endl;

    if (container.size() == initSize + numToAdd)

        good("Strings succesfully added");

    else

        bad("Strings were not added");

    cout << endl;

    showStrings(container);

}

void addStrings(vector<unique\_ptr<DynamicString>> &container, const string &FILE)

{

    ll initSize = container.size();

    ifstream file(FILE);

    string line;

    vector<string> lines;

    if (!file.is\_open())

    {

        bad("File not found");

        return;

    }

    while (getline(file, line))

    {

        lines.push\_back(line);

    }

    file.close();

    for (auto &line : lines)

    {

        container.push\_back(make\_unique<DynamicString>(line));

    }

    if (container.size() == initSize + lines.size())

        good("Strings succesfully added");

    else

        bad("Strings were not added");

}

void removeString(vector<unique\_ptr<DynamicString>> &container)

{

    ll initSize = container.size();

    if (initSize == 0)

    {

        bad("No strings found");

        return;

    }

    cout << endl;

    showStrings(container);

    cout << endl;

    cout << "Enter number of string to remove: ";

    ll numToRemove = getNum();

    numToRemove--;

    if (numToRemove < 0 || numToRemove >= initSize)

    {

        bad("Enter a valid number string number");

        return;

    }

    container.erase(container.begin() + numToRemove);

    cout << endl;

    if (container.size() == initSize - 1)

        good("String succesfully removed");

    else

        bad("String was not removed");

    cout << endl;

    showStrings(container);

}

void modifyString(vector<unique\_ptr<DynamicString>> &container)

{

    if (container.size() == 0)

    {

        bad("No strings found");

        return;

    }

    cout << endl;

    showStrings(container);

    cout << endl;

    cout << "Enter number of string to modify: ";

    ll numToModify = getNum();

    numToModify--;

    cout << endl;

    if (numToModify < 0 || numToModify >= container.size())

    {

        bad("Enter a valid number string number");

        return;

    }

    vector<string> menuItems = {

        "Change value",

        "Reverse",

        "Replace (given substring with another string)",

        "Remove excessive spaces",

        "Convert to uppercase",

        "Convert to lowercase",

        "Exit"};

    ll userDecision = showMenu(menuItems);

    cin.ignore();

    if (userDecision == 1)

    {

        string value;

        cout << "Enter new value: ";

        getline(cin, value);

        container[numToModify]->setValue(value);

    }

    else if (userDecision == 2)

    {

        container[numToModify]->m\_reverse();

    }

    else if (userDecision == 3)

    {

        string replaceThis;

        string withThis;

        cout << "Enter substring to replace: ";

        getline(cin, replaceThis);

        cout << "Enter substring to replace with: ";

        getline(cin, withThis);

        container[numToModify]->m\_replace(replaceThis, withThis);

    }

    else if (userDecision == 4)

    {

        container[numToModify]->m\_remove\_spaces();

    }

    else if (userDecision == 5)

    {

        container[numToModify]->m\_to\_upper();

    }

    else if (userDecision == 6)

    {

        container[numToModify]->m\_to\_lower();

    }

    cout << endl;

    showStrings(container);

}

void outputMenu(vector<unique\_ptr<DynamicString>> &container)

{

    vector<string> menuItems = {

        "Show strings",

        "Add strings",

        "Remove strings",

        "Modify strings",

        "Exit"};

    ll userDecision = showMenu(menuItems);

    if (userDecision == 1)

    {

        menuItems = {

            "Show in console",

            "Show in file",

            "Exit"};

        userDecision = showMenu(menuItems);

        if (userDecision == 1)

            showStrings(container);

        else if (userDecision == 2)

        {

            string fileName = ROOT\_PATH;

            fileName += getFileName();

            showStrings(container, fileName);

        }

    }

    else if (userDecision == 2)

    {

        menuItems = {

            "Add strings from console",

            "Add strings from file",

            "Exit"};

        userDecision = showMenu(menuItems);

        if (userDecision == 1)

            addStrings(container);

        else if (userDecision == 2)

        {

            string fileName = ROOT\_PATH;

            fileName += getFileName();

            addStrings(container, fileName);

        }

    }

    else if (userDecision == 3)

    {

        removeString(container);

    }

    else if (userDecision == 4)

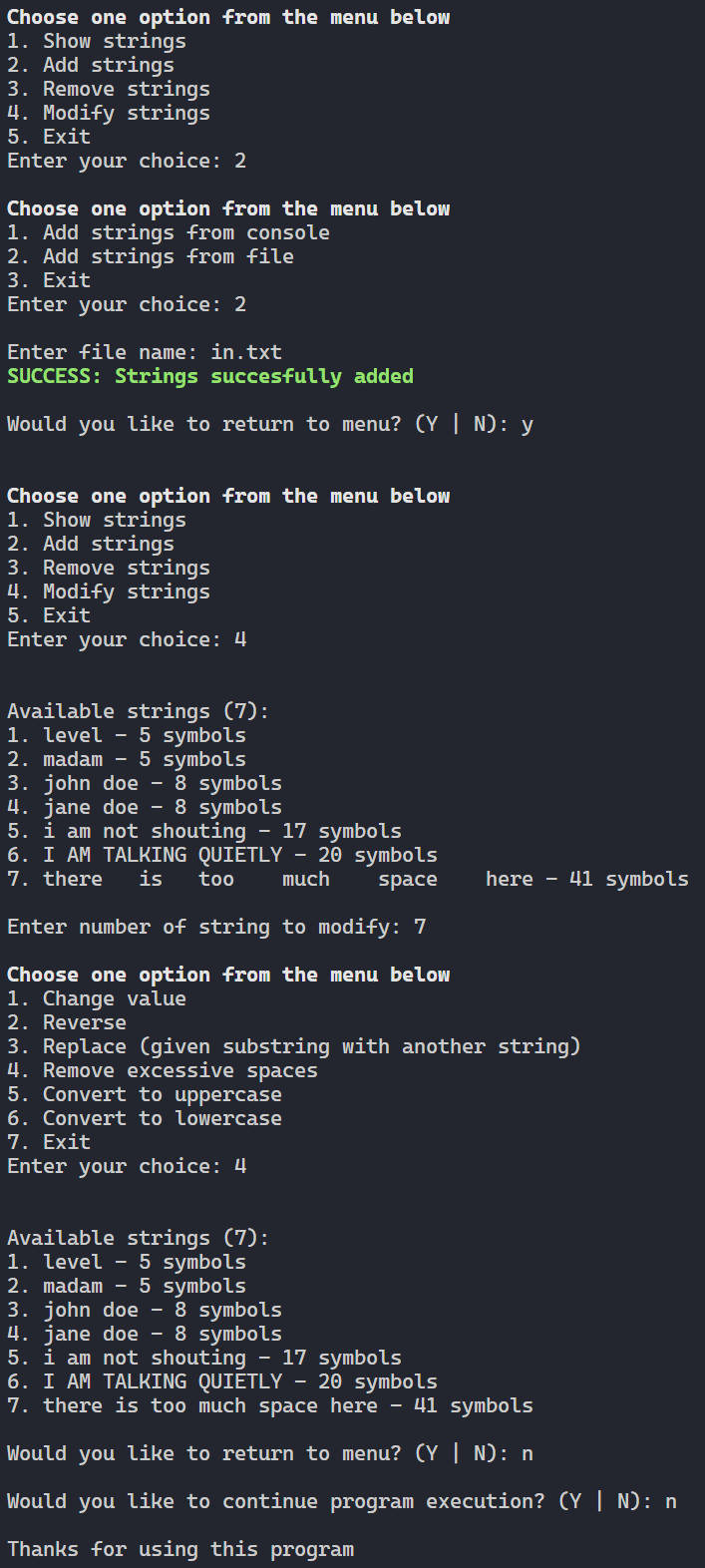
    {

        modifyString(container);

    }

}

2) Приклад роботи



3) Код програми – main.cpp

#include "D:\repos\university\lib.h"

#include "sup.h"

// Виконати завдання з лабораторної роботи #1 використавши, для зберігання даних класи Standard Template Library (STL) або list або vector. Поясніть різницю в використанні цих класів.

int main()

{

    srand(time(NULL));

    char doContinue;

    char doReturnToMenu;

    vector<unique\_ptr<Delta>> container;

    list<unique\_ptr<Delta>> container\_list;

    cout << "\n";

    vector<string> menuItems = {

        "Vector",

        "List",

    };

    ll userDecision = showMenu(menuItems);

    do

    {

        do

        {

            ///////////////////////////////////////

            if (userDecision == 1)

            {

                outputMenu(container);

            }

            else if (userDecision == 2)

            {

                outputMenu(container\_list);

            }

            ///////////////////////////////////////

            cout << "\nWould you like to return to menu? (Y | N): ";

            cin >> doReturnToMenu;

            if (doReturnToMenu == 'Y' || doReturnToMenu == 'y')

            {

                cout << endl

                     << endl;

                continue;

            }

            else

                break;

        } while (doReturnToMenu == 'y' || doReturnToMenu == 'Y');

        cout << "\nWould you like to continue program execution? (Y | N): ";

        cin >> doContinue;

        if (doContinue == 'Y' || doContinue == 'y')

        {

            cout << "\n\n";

            continue;

        }

        else

            break;

    } while (doContinue = 'Y' || doContinue == 'y');

    cout << "\nThanks for using this program\n\n";

    return 0;

}

3) Код програми – sup.h

#include "D:\repos\university\lib.h"

#include "sup.h"

// Виконати завдання з лабораторної роботи #1 використавши, для зберігання даних класи Standard Template Library (STL) або list або vector. Поясніть різницю в використанні цих класів.

class Delta

{

private:

    ll m\_id;

public:

    Delta() : m\_id(nextID()) {}

    ll nextID()

    {

        static ll id = 0;

        return id++;

    }

    ll getID() const { return m\_id; }

    ~Delta() {}

};

void showObjs(vector<unique\_ptr<Delta>> &container)

{

    ll objsNum = container.size();

    if (objsNum == 0)

    {

        bad("No objects to show");

        return;

    }

    cout << "Available objects (" << objsNum << "):\n";

    for (ll i = 0; i < objsNum; i++)

    {

        cout << i + 1 << ". Descriptor: " << container[i]->getID() << endl;

    }

}

void showObjs(list<unique\_ptr<Delta>> &container)

{

    ll objsNum = container.size();

    if (objsNum == 0)

    {

        bad("No objects to show");

        return;

    }

    cout << "Available objects (" << objsNum << "):\n";

    ll i = 0;

    for (auto it = container.begin(); it != container.end(); it++)

    {

        cout << i + 1 << ". Descriptor: " << it->get()->getID() << endl;

        i++;

    }

}

void addObjs(vector<unique\_ptr<Delta>> &container)

{

    ll initSize = container.size();

    cout << "Enter number of objects to add: ";

    ll numToAdd = getNum();

    cout << endl;

    if (numToAdd == 0)

    {

        bad("Enter a valid number of objects");

        return;

    }

    for (ll i = 0; i < numToAdd; i++)

    {

        container.push\_back(make\_unique<Delta>());

    }

    cout << endl;

    if (container.size() == initSize + numToAdd)

        good("Objects added successfully");

    else

        bad("Failed to add objects");

    cout << endl;

    showObjs(container);

}

void addObjs(list<unique\_ptr<Delta>> &container)

{

    ll initSize = container.size();

    cout << "Enter number of objects to add: ";

    ll numToAdd = getNum();

    cout << endl;

    if (numToAdd == 0)

    {

        bad("Enter a valid number of objects");

        return;

    }

    for (ll i = 0; i < numToAdd; i++)

    {

        container.push\_back(make\_unique<Delta>());

    }

    cout << endl;

    if (container.size() == initSize + numToAdd)

        good("Objects added successfully");

    else

        bad("Failed to add objects");

    cout << endl;

    showObjs(container);

}

void delObjs(vector<unique\_ptr<Delta>> &container)

{

    ll initSize = container.size();

    if (initSize == 0)

    {

        bad("No objects found");

        return;

    }

    cout << endl;

    showObjs(container);

    cout << endl;

    cout << "Enter object number to remove: ";

    ll numToRemove = getNum();

    numToRemove--;

    if (numToRemove < 0 || numToRemove >= initSize)

    {

        bad("Enter a valid object number");

        return;

    }

    container.erase(container.begin() + numToRemove);

    cout << endl;

    if (container.size() == initSize - 1)

        good("Object succesfully removed");

    else

        bad("Object was not removed");

    cout << endl;

    showObjs(container);

}

void delObjs(list<unique\_ptr<Delta>> &container)

{

    ll initSize = container.size();

    if (initSize == 0)

    {

        bad("No objects found");

        return;

    }

    cout << endl;

    showObjs(container);

    cout << endl;

    cout << "Enter object number to remove: ";

    ll numToRemove = getNum();

    numToRemove--;

    if (numToRemove < 0 || numToRemove >= initSize)

    {

        bad("Enter a valid object number");

        return;

    }

    auto it = next(container.begin(), numToRemove);

    container.erase(it);

    if (container.size() == initSize - 1)

        good("Object succesfully removed");

    else

        bad("Object was not removed");

    cout << endl;

    showObjs(container);

}

void outputMenu(vector<unique\_ptr<Delta>> &container)

{

    vector<string> menuItems = {

        "Show objects",

        "Add objects",

        "Remove objects",

        "Exit"};

    ll userDecision = showMenu(menuItems);

    if (userDecision == 1)

    {

        showObjs(container);

    }

    else if (userDecision == 2)

    {

        addObjs(container);

    }

    else if (userDecision == 3)

    {

        delObjs(container);

    }

}

void outputMenu(list<unique\_ptr<Delta>> &container)

{

    vector<string> menuItems = {

        "Show objects",

        "Add objects",

        "Remove objects",

        "Exit"};

    ll userDecision = showMenu(menuItems);

    if (userDecision == 1)

    {

        showObjs(container);

    }

    else if (userDecision == 2)

    {

        addObjs(container);

    }

    else if (userDecision == 3)

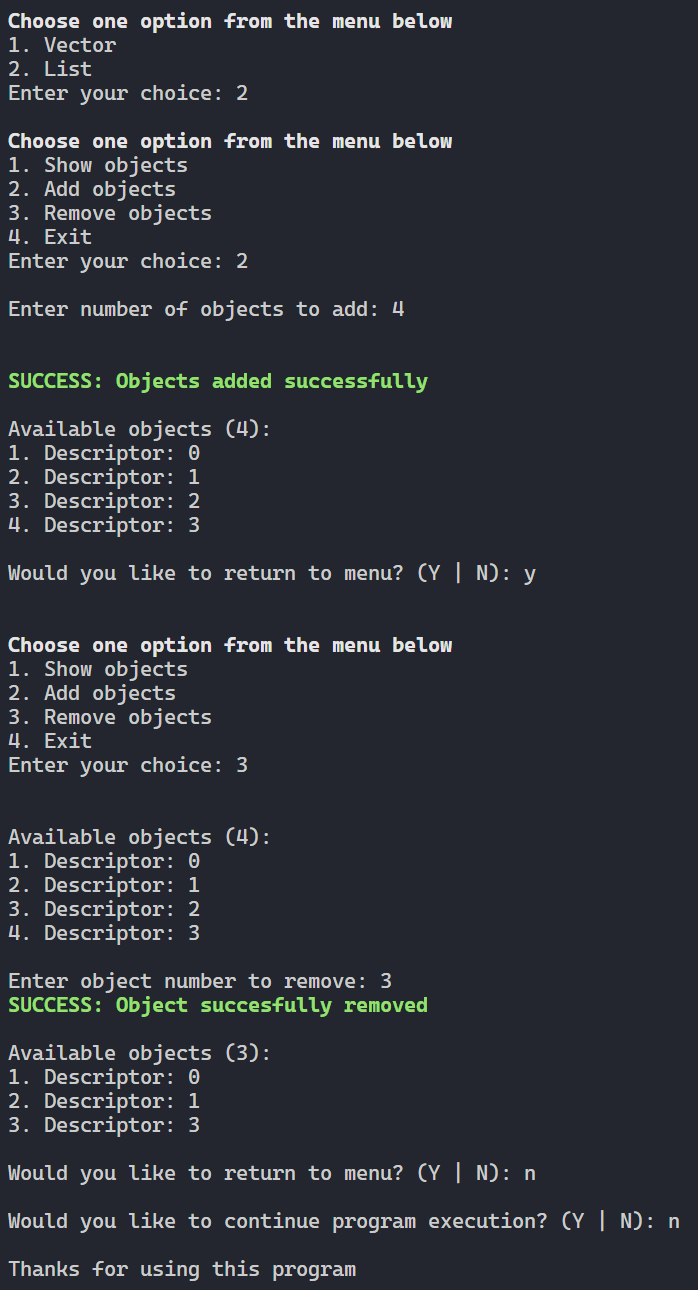
    {

        delObjs(container);

    }

}

3) Приклад роботи



Код бібліотеки – lib.h

#include <bits/stdc++.h>

#include "lib.h"

using namespace std;

#define ll long long

#define all(x) (x).begin(), (x).end()

#define pb push\_back

#define eb emplace\_back

#define mp make\_pair

#define endl "\n"

void dbg\_out()

{

    cerr << endl;

}

template <typename Head, typename... Tail>

void dbg\_out(Head H, Tail... T)

{

    cerr << ' ' << H;

    dbg\_out(T...);

}

#define dbg(...) cerr << "(" << #\_\_VA\_ARGS\_\_ << "):", dbg\_out(\_\_VA\_ARGS\_\_)

void bad(const string &INPUT)

{

    stringstream ss;

    ss << "\033[1;31mERROR: " << INPUT << "\033[0m";

    cerr << ss.str() << endl;

}

void good(const string &INPUT)

{

    stringstream ss;

    ss << "\033[1;32mSUCCESS: " << INPUT << "\033[0m";

    cerr << ss.str() << endl;

}

ll getNum()

{

    ll number;

    while (!(cin >> number))

    {

        cin.clear();

        cin.ignore(256, '\n');

        cout << endl;

        bad("Enter an integer");

        cout << endl;

    }

    return number;

}

ostream &BOLD(ostream &os)

{

    return os << "\e[1m";

}

ostream &UNBOLD(ostream &os)

{

    return os << "\e[0m";

}

ostream &RED(ostream &os)

{

    return os << "\033[1;31m";

}

ostream &UNRED(ostream &os)

{

    return os << "\033[0m";

}

ostream &GREEN(ostream &os)

{

    return os << "\033[1;32m";

}

ostream &UNGREEN(ostream &os)

{

    return os << "\033[0m";

}

ostream &GRAY(ostream &os)

{

    return os << "\033[1;30m";

}

ostream &UNGRAY(ostream &os)

{

    return os << "\033[0m";

}

ostream &YELLOW(ostream &os)

{

    return os << "\033[1;33m";

}

ostream &UNYELLOW(ostream &os)

{

    return os << "\033[0m";

}

ll showMenu(const vector<string> &MENU\_OPTIONS)

{

    cout << BOLD << "Choose one option from the menu below\n"

         << UNBOLD;

    for (ll i = 0; i < MENU\_OPTIONS.size(); i++)

    {

        cout << i + 1 << ". " << MENU\_OPTIONS[i] << endl;

    }

    cout << "Enter your choice: ";

    ll userDecision = getNum();

    cout << endl;

    return userDecision;

}

string validateName(string inputString)

{

    stringstream stringProcessor(inputString);

    string wordHolder;

    string resultHolder;

    while (stringProcessor >> wordHolder)

    {

        if (!isupper(wordHolder[0]))

        {

            wordHolder[0] = toupper(wordHolder[0]);

        }

        resultHolder += wordHolder + " ";

    }

    return resultHolder;

}

string getEmailAddress()

{

    string emailAddress;

    cout << "Please enter an email address: ";

    cin >> emailAddress;

    if (emailAddress.find("@") == string::npos)

    {

        cout << "\nERROR: Invalid email address\n\n";

        getEmailAddress();

    }

    else

        return emailAddress;

    return "";

}

string getFileName()

{

    string fileName = "";

    bool isExtensionFound = true;

    do

    {

        cout << "Enter file name: ";

        cin >> fileName;

        if (fileName.find(".") == string::npos)

        {

            isExtensionFound = false;

            cout << "\nERROR: File extension not found. Try again...\n\n";

            continue;

        }

        else

            break;

    } while (isExtensionFound == false);

    return fileName;

}

string generateRandomString(int length)

{

    string chars = "abcdefghijklmnopqrstuvwxy";

    string randomString = "";

    for (int i = 0; i < length; i++)

    {

        int index = rand() % chars.size();

        randomString += chars[index];

    }

    return randomString;

}

string generateRandomPassword(int length)

{

    string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYabcdefghijklmnopqrstuvwxy1234567890!@#$%^&\*()\_+=-[]{}`~';/.,";

    string randomPass = "";

    for (int i = 0; i < length; i++)

    {

        int index = rand() % chars.size();

        randomPass += chars[index];

    }

    return randomPass;

}

template <typename T>

vector<T> getUniqueVector(vector<T> &inputVector)

{

    vector<T> uniqueElements;

    unordered\_set<T> seenElements;

    for (T element : inputVector)

    {

        if (seenElements.find(element) == seenElements.end())

        {

            uniqueElements.push\_back(element);

            seenElements.insert(element);

        }

    }

    return uniqueElements;

}

template <typename T>

void quickSort(vector<T> &arr, int left, int right)

{

    int i = left, j = right;

    int pivot = arr[(left + right) / 2];

    if (arr.size() <= 1)

        return;

    while (i <= j)

    {

        while (arr[i] > pivot)

            i++;

        while (arr[j] < pivot)

            j--;

        if (i <= j)

        {

            swap(arr[i], arr[j]);

            i++;

            j--;

        }

    };

    if (left < j)

        quickSort(arr, left, j);

    if (i < right)

        quickSort(arr, i, right);

}

template <typename T>

void exchangeSort(vector<T> &arr)

{

    if (arr.size() <= 1)

        return;

    for (int i = 0; i < arr.size() - 1; i++)

        for (int j = i + 1; j < arr.size(); j++)

            if (arr[i] < arr[j])

                swap(arr[i], arr[j]);

}

template <typename T>

void bubbleSort(vector<T> &arr)

{

    if (arr.size() <= 1)

        return;

    for (int i = 0; i < arr.size(); i++)

        for (int j = 0; j < arr.size() - i - 1; j++)

            if (arr[j] < arr[j + 1])

                swap(arr[j], arr[j + 1]);

}

template <typename T>

void mergeSort(vector<T> &arr)

{

    if (arr.size() <= 1)

        return;

    vector<int> left, right;

    int middle = arr.size() / 2;

    for (int i = 0; i < middle; i++)

        left.push\_back(arr[i]);

    for (int i = middle; i < arr.size(); i++)

        right.push\_back(arr[i]);

    mergeSort(left);

    mergeSort(right);

    int i = 0, j = 0, k = 0;

    while (i < left.size() && j < right.size())

    {

        if (left[i] > right[j])

        {

            arr[k] = left[i];

            i++;

        }

        else

        {

            arr[k] = right[j];

            j++;

        }

        k++;

    }

    while (i < left.size())

    {

        arr[k] = left[i];

        i++;

        k++;

    }

    while (j < right.size())

    {

        arr[k] = right[j];

        j++;

        k++;

    }

}

template <typename T>

void outputArray(vector<T> arr)

{

    for (auto i : arr)

        cout << i << " ";

    cout << endl;

}

void outputArray(int \*arr)

{

    int n = sizeof(arr) / sizeof(int);

    for (int i = 0; i < n; i++)

        cout << arr[i] << " ";

    cout << endl;

}

template <typename T>

void outputArray(vector<vector<T>> &arr)

{

    int n = arr.size();

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

            cout << arr[i][j] << " ";

        cout << "\n";

    }

}

string toLower(string str)

{

    transform(str.begin(), str.end(), str.begin(), ::tolower);

    return str;

}

Висновки

Таким чином, ми навчилися використовувати шаблони при створенні програм мовою програмування С++.