**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](#_Toc132561518)

[Текст завдання №5 3](#_Toc132561519)

[Код програми – main.cpp 3](#_Toc132561520)

[Код програми – sup.h 3](#_Toc132561521)

[Код бібліотеки – lib.h 3](#_Toc132561522)

[Приклад роботи 3](#_Toc132561523)

[Висновки 3](#_Toc132561524)

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 necessary libraries

#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.

// func main start

int main()

{

    // declare local variables //

    srand(time(NULL));

    char doContinue;

    ll userDecision;

    /////////////////////////////

    // project intro

    cout << "\n";

    do

    {

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

        char doReturnToMenu;

        // output menu to user and prompt them to choose an option

        cout << BOLD << "Choose a data type\n"

             << UNBOLD;

        cout << "1. Integer\n";

        cout << "2. Double\n";

        cout << "3. String\n";

        cout << "4. Exit\n";

        cout << "Enter: ";

        userDecision = getNum();

        cout << endl;

        // if user chose to add objects

        if (userDecision == 1)

        {

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

            outputMenu(deltaObjectsVector);

        }

        // if user chose to delete objects

        else if (userDecision == 2)

        {

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

            outputMenu(deltaObjectsVector);

        }

        // if user chose to print objects

        else if (userDecision == 3)

        {

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

            outputMenu(deltaObjectsVector);

        }

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

        // ask user if they would like to continue execution of program

        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');

    // stop main function execution

    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.

// delta class declaration

template <typename T>

class Delta

{

private:

    // declare private members

    ll descriptor;

    T value;

public:

    // default constructor for when we don't specify the value

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

    // parameterised constructor for when we do specify the value

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

    static ll nextDescriptor()

    {

        static ll nextDescriptor = 0;

        return nextDescriptor++;

    }

    // next identifier creator

    T getValue() const

    {

        return value;

    }

    ll getDescriptor() const

    {

        return descriptor;

    }

    void setValue(T inValue)

    {

        value = inValue;

    }

    // create default destructor function

    ~Delta()

    {

    }

};

// object creation function

template <typename T>

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

{

    // ask user to enter number of delta objects to create

    cout << "\nEnter an amount of objects to create: ";

    ll objectsAmount = getNum();

    // if entered amount is less than one

    if (objectsAmount < 1)

    {

        // output error and stop function

        cout << RED << "\nERROR: Invalid amount of objects...\n\n"

             << UNRED;

        // stop function execution

        return;

    }

    // create specified amount of objects using a for loop

    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;

        deltaObjectsVector.push\_back(make\_unique<Delta<T>>(value));

    }

    // end function execution

    return;

}

// printing objects function

template <typename T>

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

{

    // get vector size

    ll vectorSize = deltaObjectsVector.size();

    if (vectorSize == 0)

    {

        bad("No objects to print");

        return;

    }

    // output all objects with their identifiers using a for loop

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

         << UNBOLD;

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

    {

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

    }

}

// object deletion function

template <typename T>

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

{

    // check if vector is empty

    if (deltaObjectsVector.size() == 0)

        // if so, output error message

        cout << GRAY << "\nNo objects to delete\n"

             << UNGRAY;

    // if not

    else

    {

        // print all objects to user

        printObjects(deltaObjectsVector);

        cout << "\nEnter a number of object to delete: ";

        ll numToDelete = getNum();

        // if the object number is out of range

        if (numToDelete > deltaObjectsVector.size() - 1 || numToDelete < 0)

            // output error message

            cout << RED << "\nERROR: Invalid object number\n"

                 << UNRED;

        // if not

        else

        {

            // output success message

            cout << GREEN << endl

                 << deltaObjectsVector[numToDelete]->getValue() << " successfully deleted\n"

                 << UNGREEN;

            // erase object from vector

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

        }

    }

    // end function execution

    return;

}

// for showing the main menu of the application

template <typename T>

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

{

    char doContinue;

    do

    {

        // output menu to user and prompt them to choose an option

        cout << BOLD << "Choose an option from menu\n"

             << UNBOLD;

        cout << "1. Add objects\n";

        cout << "2. Delete objects\n";

        cout << "3. Print objects\n";

        cout << "4. Exit\n";

        cout << "Enter: ";

        ll userDecision = getNum();

        cout << endl;

        // if user chose to add objects

        if (userDecision == 1)

        {

            // add objects

            createObjects(deltaObjectsVector);

            // print them to console

            printObjects(deltaObjectsVector);

        }

        // if user chose to delete objects

        else if (userDecision == 2)

        {

            // delete them

            deleteObjects(deltaObjectsVector);

            // print them to console

            printObjects(deltaObjectsVector);

        }

        // if user chose to print objects

        else if (userDecision == 3)

            // print objects to console

            printObjects(deltaObjectsVector);

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

        cin >> doContinue;

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

        {

            cout << endl;

            continue;

        }

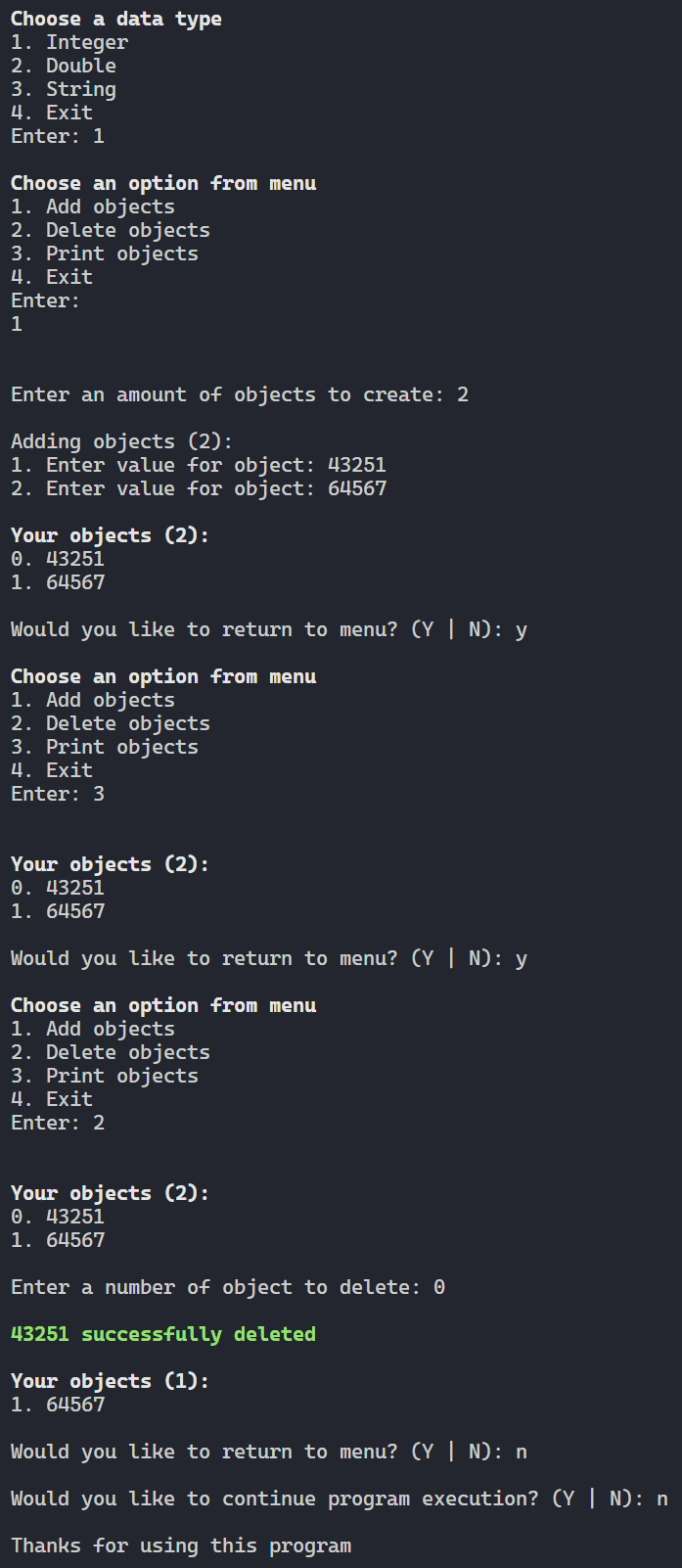
        else

            break;

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

}

1) Приклад роботи



2) Код програми – main.cpp

Код

2) Код програми – sup.h

Код

2) Приклад роботи

3) Код програми – main.cpp

Код

3) Код програми – sup.h

Код

3) Приклад роботи

Код бібліотеки – lib.h

// include necessary libraries

#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\_\_)

// For making text bold

ostream &BOLD(ostream &os)

{

    return os << "\e[1m";

}

// For changing text back to normal

ostream &UNBOLD(ostream &os)

{

    return os << "\e[0m";

}

// For setting text color to red

ostream &RED(ostream &os)

{

    return os << "\033[1;31m";

}

// For changing text back to normal

ostream &UNRED(ostream &os)

{

    return os << "\033[0m";

}

// For setting text color to green

ostream &GREEN(ostream &os)

{

    return os << "\033[1;32m";

}

// For changing text back to normal

ostream &UNGREEN(ostream &os)

{

    return os << "\033[0m";

}

// For changing text color to gray

ostream &GRAY(ostream &os)

{

    return os << "\033[1;30m";

}

// For changing text back to normal

ostream &UNGRAY(ostream &os)

{

    return os << "\033[0m";

}

// For changing text color to yellow

ostream &YELLOW(ostream &os)

{

    return os << "\033[1;33m";

}

// For changing text back to normal

ostream &UNYELLOW(ostream &os)

{

    return os << "\033[0m";

}

// for transforming string to lowercase

string toLower(string str)

{

    // transfort the whole stirng

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

    // return it

    return str;

}

// for outputting error messages

void bad(const string &INPUT)

{

    // create a stringstream object to store the message

    stringstream ss;

    // add ANSI escape codes for red text

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

    // output the message to cerr

    cerr << ss.str() << endl;

}

// for outputting success messages

void good(const string &INPUT)

{

    // create a stringstream object to store the message

    stringstream ss;

    // add ANSI escape codes for red text

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

    // output the message to cerr

    cerr << ss.str() << endl;

}

// for getting nubmer input from user

ll getNum()

{

    // declare variables for holding the input

    ll number;

    // while user enters invalid input

    while (!(cin >> number))

    {

        // clear buffer

        cin.clear();

        // ignore new line char

        cin.ignore(256, '\n');

        // output error message and continue

        cout << endl;

        bad("Enter an integer");

        cout << endl;

    }

    return number;

}

// validates an inputed name

string validateName(string inputString)

{

    stringstream stringProcessor(inputString); // create a stringstream object from the inputted name

    string wordHolder;

    string resultHolder;

    while (stringProcessor >> wordHolder)

    { // loop through each word in the stringstream

        if (!isupper(wordHolder[0]))

        {                                           // check if the first letter of the word is not uppercase

            wordHolder[0] = toupper(wordHolder[0]); // capitalize the first letter of the word

        }

        resultHolder += wordHolder + " "; // add the modified word to the new name

    }

    return resultHolder; // return the result of the validation

}

// implements the Quick Sort algorithm to sort the elements of the given vector in ascending order

template <typename T>

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

{

    int i = left, j = right;             // initialize two variables, i and j, to the values of left and right respectively.

    int pivot = arr[(left + right) / 2]; // find the pivot element by taking the average of the left and right

    // check if size of array is <= 1, if so stop function

    if (arr.size() <= 1)

        return;

    // iterate through the values of i and j

    while (i <= j)

    {

        // iterate through the array until finds an element that is greater than the pivot value

        while (arr[i] > pivot)

            i++; // increment i

        // iterate through the array until finds an element that is less than the pivot value

        while (arr[j] < pivot)

            j--; // decrement j

        // check if the value of i is <= j.

        if (i <= j)

        {

            // swap the values of arr[i] and arr[j], then increment i and decrement j

            swap(arr[i], arr[j]);

            i++;

            j--;

        }

    };

    // recursively sort the left part of the array

    if (left < j)

        quickSort(arr, left, j);

    // recursively sort the right part of the array

    if (i < right)

        quickSort(arr, i, right);

}

// implements the Exchange Sort algorithm to sort the elements of the given vector in ascending order

template <typename T>

void exchangeSort(vector<T> &arr)

{

    // check if size of array is <= 1, if so stop function

    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]);

}

// implements the Bubble Sort algorithm to sort the elements of the given vector in ascending order

template <typename T>

void bubbleSort(vector<T> &arr)

{

    // check if size of array is <= 1, if so stop function

    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]);

}

// implements the Merge Sort algorithm to sort the elements of the given vector in ascending order

template <typename T>

void mergeSort(vector<T> &arr)

{

    // check if size of array is <= 1, if so stop function

    if (arr.size() <= 1)

        return;

    vector<int> left, right;     // for partitioning the array

    int middle = arr.size() / 2; // calculate the middle index of the array

    // iterate through the array from start to middle and add each element to left vector

    for (int i = 0; i < middle; i++)

        left.push\_back(arr[i]);

    // iterate through the array from middle to end and add each element to right vector

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

        right.push\_back(arr[i]);

    // recursively sort left and right vectors

    mergeSort(left);

    mergeSort(right);

    int i = 0, j = 0, k = 0; // declare indeces

    // iterate through the left and right vectors until the end of either vector

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

    {

        if (left[i] > right[j])

        {

            arr[k] = left[i];

            i++;

        }

        else

        {

            arr[k] = right[j];

            j++;

        }

        k++;

    }

    // merge left and input vectors

    while (i < left.size())

    {

        arr[k] = left[i];

        i++;

        k++;

    }

    // merge right and input vectors

    while (j < right.size())

    {

        arr[k] = right[j];

        j++;

        k++;

    }

}

// to output one dimensional vector

template <typename T>

void outputArray(vector<T> arr)

{

    for (auto i : arr)

        cout << i << " ";

    cout << endl;

}

// to output one dimensional array

void outputArray(int \*arr)

{

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

    for (int i = 0; i < n; i++)

        cout << arr[i] << " ";

    cout << endl;

}

// to output two dimensional vector

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";

    }

}

// for getting email address from user

string getEmailAddress()

{

    // ask user for email address

    string emailAddress;

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

    cin >> emailAddress;

    // validate email address

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

    {

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

        getEmailAddress();

    }

    else

        return emailAddress;

    // should not reach this pointt

    return "";

}

// For getting text file inputted from user

string getFileName()

{

    // Declare local variables

    string fileName = "";         // for storing file name

    bool isExtensionFound = true; // for tracking file extension

    // Create do while loop for properly getting file name with extension

    do

    {

        // Ask user to enter file name

        cout << "Enter file name: ";

        cin >> fileName;

        // Create condition to check if file extension is found

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

        {

            // Execute if not found

            isExtensionFound = false; // set tracker state to false

            // Output error message

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

            continue; // jump into next iteration

        }

        // If extension is found

        else

            break; // jump out of loop

    } while (isExtensionFound == false);

    // Return file name

    return fileName;

}

// For generating a random string of given length

string generateRandomString(int length)

{

    // Declare local variables

    string chars = "abcdefghijklmnopqrstuvwxy"; // characters pool to generate random string from

    string randomString = "";                   // random string result holder

    // Generate a random character and add it to the string until it reaches the desired length

    for (int i = 0; i < length; i++)

    {

        // Generate a random index between 0 and the size of our character pool

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

        // Add the character at that index to our string

        randomString += chars[index];

    }

    return randomString; // Append ".txt" and return the generated string

}

// For generating a random password of given length

string generateRandomPassword(int length)

{

    // Declare local variables

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

    string randomPass = "";                                                                                 // random string result holder

    // Generate a random character and add it to the string until it reaches the desired length

    for (int i = 0; i < length; i++)

    {

        // Generate a random index between 0 and the size of our character pool

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

        // Add the character at that index to our string

        randomPass += chars[index];

    }

    return randomPass;

}

// returns a vector containing only the unique elements from the input vector

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

{

    vector<string> uniqueElements;      // to store unique elements as strings

    unordered\_set<string> seenElements; // to store elements that have already been seen

    // iterate through each element of the inputVector

    for (string element : inputVector)

    {

        // check if the element is present in the seenElements container

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

        {

            uniqueElements.push\_back(element); // add the element to uniqueElements vector

            seenElements.insert(element);      // inserts the element into the seenElements set, if it is not already present

        }

    }

    // return a vector containing only the unique elements from the original vector

    return uniqueElements;

}

Висновки

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