**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по лабораторной работе №7**

**по дисциплине «****Объектно-ориентированное программирование»**

Тема: «Использование переменных ссылочного типа»

|  |  |  |
| --- | --- | --- |
| Студенты гр. 1335 |  | Максимов Ю. Е. |
| Преподаватель: |  | Новакова Н.Е. |

Санкт-Петербург

2024

# **1. Цель работы**

Изучение переменных ссылочного типа в языке C++ с помощью программного продукта компании CLion.

# **2. Анализ задачи**

Необходимо:

1. Написать программу, которая обеспечивает перевод денег с одного счета на другой;
2. Написать программу, которая читает строку символов и переносит ее в другую переменную в обратном порядке;
3. Написать программу, которая считывает текст из одного файла и переписывает в другой файл, переводя все буквы в верхний регистр.
4. Написать программу, которая добавляет статический метод IsItFormattable, который определяет поддерживается ли передающийся в этот метод объект IFormattabl, в класс Utils;
5. Написать программу, которая добавляет метод Display, который использует оператор as для определения передачи объекта как параметра в интерфейс IPrintable.

# **3. Ход выполнения работы**

## **3.1 Упражнение 1**

В ходе выполнения данного упражнения написана программа, которая переводит деньги с одного аккаунта на другой.

### 3.1.1 Пошаговое описание алгоритма

На счет каждого аккаунта кладется 100.

На экран пользователя выводится состояние аккаунтов до операции перевода.

Вызывается метод TransferFrom для перевода денег с одного аккаунта на другой.

На экран пользователя выводится состояние аккаунтов поле операции перевода.

### 3.1.2 Используемые классы и методы

В программе, написанной в данном упражнении, используются следующие методы:

- std::cout – служит для отображения на экране строк и значений переменных, переданных в метод в качестве параметров, с переходом на новую строку;

- std::cin – ожидает следующего нажатия клавиши пользователем;

- main() – служит для запуска программы.

- BankAccount – класс, который описывает банковский аккаунт с помощью методов: TransferFrom (перевод денег), Number (возвращает номер аккаунта), Balance (возвращает текущий баланс), Type (возвращает тип аккаунта), Withdraw (позволяет произвести снятие денег) и Deposit (позволяет пополнить счет), Populate (создание), NextNumber (следующий номер счета).

### 3.1.3 Контрольный пример

На рис.3.1 представлены результаты выполнения программы.

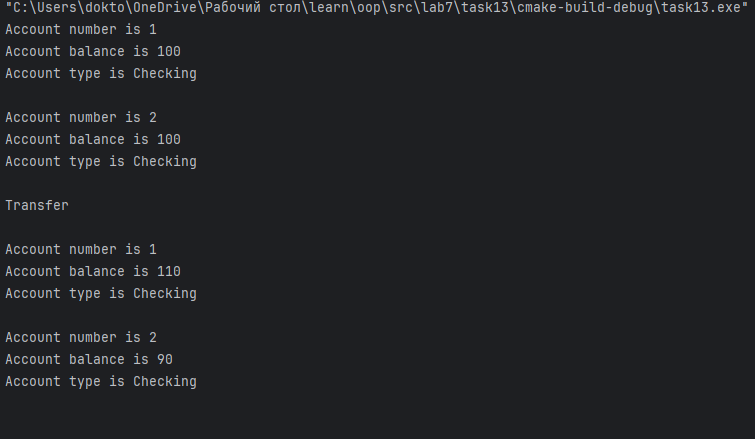
****

Рис.3.1 Контрольный пример для программы 1

Как видно из рисунка, на экран выводятся данные аккаунтов до и после перевода денег.

**3.1.4. Диаграмма**



# **3.1.5 Листинг программы**

**AccountType.h**

//

// Created by dokto on 11.06.2024.

//

#pragma once

namespace bank {

enum class AccountType {

Checking,

Deposit

};

}

**bankAccount.cpp**

//

// Created by dokto on 11.06.2024.

//

#include "bankAccount.h"

/\*\*

\* Populates the bank account with the given account number and balance.

\*

\* @param number The account number.

\* @param balance The account balance.

\*

\* @throws None.

\*/

void bank::BankAccount::populate(const int number, const double balance) {

accNo = number;

accBal = balance;

accType = AccountType::Checking;

}

/\*\*

\* Returns the account number.

\*

\* @return The account number.

\*

\* @throws None.

\*/

auto bank::BankAccount::getNumber() const -> long {

return accNo;

}

/\*\*

\* Returns the account balance.

\*

\* @return The account balance.

\*

\* @throws None.

\*/

auto bank::BankAccount::getBalance() const -> double {

return accBal;

}

/\*\*

\* Returns the account type.

\*

\* @return The account type.

\*

\* @throws None.

\*/

auto bank::BankAccount::getType() const -> AccountType {

return accType;

}

/\*\*

\* Sets the account number.

\*

\* @param number The account number.

\*

\* @throws None.

\*/

auto bank::BankAccount::setNumber(const long number) -> void {

accNo = number;

}

/\*\*

\* Sets the balance of the bank account.

\*

\* @param balance The new balance to be set.

\*

\* @return void

\*

\* @throws None

\*/

auto bank::BankAccount::setBalance(const double balance) -> void {

accBal = balance;

}

/\*\*

\* Sets the account type.

\*

\* @param type The new account type.

\*

\* @return void

\*

\* @throws None

\*/

auto bank::BankAccount::setType(const AccountType type) -> void {

accType = type;

}

/\*\*

\* Withdraws the specified amount from the bank account if there are sufficient funds.

\*

\* @param amount The amount to be withdrawn.

\*

\* @return True if the withdrawal was successful, false otherwise.

\*

\* @throws None

\*/

auto bank::BankAccount::withdraw(const double amount) -> bool {

const auto sufficientFunds = accBal >= amount;

if (sufficientFunds) {

accBal -= amount;

}

return sufficientFunds ;

}

/\*\*

\* Deposits the specified amount into the bank account.

\*

\* @param amount The amount to be deposited.

\*

\* @return The updated balance of the bank account after the deposit.

\*

\* @throws None

\*/

auto bank::BankAccount::deposit(const double amount) -> double {

return accBal += amount;

}

/\*\*

\* Transfers the specified amount from the given BankAccount to the current BankAccount.

\*

\* @param from The BankAccount to transfer the amount from.

\* @param amount The amount to be transferred.

\*

\* @return void

\*

\* @throws None

\*/

auto bank::BankAccount::transferFrom(BankAccount &from, const double amount) -> void {

if(from.withdraw(amount)) {

this->deposit(amount);

}

}

/\*\*

\* Returns the next account number.

\*

\* @return The next account number.

\*

\* @throws None

\*/

auto bank::BankAccount::NextNumber() -> long {

return nextNumber++;

}

**bankAccount.h**

//

// Created by dokto on 11.06.2024.

//

#pragma once

#include "AccountType.h"

namespace bank {

class BankAccount {

public:

void populate(int number, double balance);

[[nodiscard]] auto getNumber() const -> long;

[[nodiscard]] auto getBalance() const -> double;

[[nodiscard]] auto getType() const -> AccountType;

auto setNumber(long number) -> void;

auto setBalance(double balance) -> void;

auto setType(AccountType type) -> void;

auto withdraw(double amount) -> bool;

auto deposit(double amount) -> double;

auto transferFrom(BankAccount &from, double amount) -> void;

private:

auto NextNumber() -> long;

long accNo = 0;

double accBal = 0;

AccountType accType = AccountType::Checking;

long nextNumber = 123;

};

}

**CreateAccount.cpp**

//

// Created by dokto on 11.06.2024.

//

#include "CreateAccount.h"

#include "iostream"

#include "bank/AccountType.h"

namespace {

/\*\*

\* Creates a bank account with the given account number and a default balance of 100.

\*

\* @param num The account number.

\*

\* @return A bank::BankAccount object representing the newly created account.

\*

\* @throws None.

\*/

auto createBankAccount(const int num) -> bank::BankAccount {

bank::BankAccount tmp ;

tmp.populate( num, 100);

return tmp;

}

}

/\*\*

\* Executes the process of creating a bank account.

\*

\* This function creates a new bank account by calling the `creatBankAccount` function

\* and stores the result in the `bankAccount` variable. Then, it calls the `write`

\* function to display the details of the bank account.

\*

\* @return void

\*/

auto bank::CreateAccount::process() -> void {

auto bankAccount1 = createBankAccount(1);

write(bankAccount1);

auto bankAccount2 = createBankAccount(2);

write(bankAccount2);

std::cout << "Transfer" << std::endl << std::endl;

bankAccount1.transferFrom( bankAccount2, 10);

write(bankAccount1);

write(bankAccount2);

}

/\*\*

\* Writes the details of a bank account to the standard output.

\*

\* @param bankAccount the bank account to write

\*

\* @return void

\*

\* @throws None

\*/

auto bank::CreateAccount::write(const bank::BankAccount &bankAccount) -> void {

std::cout << "Account number is " << bankAccount.getNumber() << std::endl;

std::cout << "Account balance is " << bankAccount.getBalance() << std::endl;

std::cout << "Account type is " <<

(bankAccount.getType() == bank::AccountType::Checking ? "Checking" : "Deposit") << std::endl << std::endl;

}

**CreateAccount.h**

//

// Created by dokto on 11.06.2024.

//

#pragma once

#include "bank/bankAccount.h"

namespace bank {

class CreateAccount {

public:

static auto process() -> void;

private:

static auto write(const bank::BankAccount &bankAccount) -> void;

};

}

### 3.2 Упражнение 2

В ходе выполнения данного упражнения, написана программа, которая переносит строку символов в другую переменную в обратном порядке.

## **3.2.1 Пошаговое описание алгоритма**

Ввод пользователем сообщения.

Вызов функции Reverse для изменения порядка на обратный.

Вывод сообщения введенного пользователем в обратном порядке.

### 3.2.2 Используемые классы и методы

В программе, написанной в данном упражнении, используются следующие методы:

- std::cout – служит для отображения на экране строк и значений переменных, переданных в метод в качестве параметров, с переходом на новую строку;

-std:;cin – ожидает следующего нажатия клавиши пользователем;

- main() – служит для запуска программы.

- Utils – класс, который взят из прошлой лабораторной работы и добавлен метод Reverse.

### 3.2.3 Контрольный пример

На рис.3.2 представлены результаты выполнения программы.

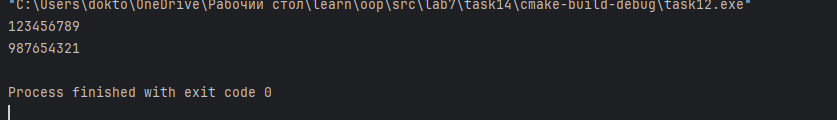
****

Рис.3.2 Контрольный пример для программы 2

Как видно из рисунка, пользователь вводит сообщение, затем выводится перевернутое сообщение.

# **4. Листинг программы**

**AccountType.h**

//

// Created by dokto on 11.06.2024.

//

#pragma once

namespace bank {

enum class AccountType {

Checking,

Deposit

};

}

**bankAccount.cpp**

//

// Created by dokto on 11.06.2024.

//

#include "bankAccount.h"

/\*\*

\* Populates the bank account with the given account number and balance.

\*

\* @param number The account number.

\* @param balance The account balance.

\*

\* @throws None.

\*/

void bank::BankAccount::populate(const int number, const double balance) {

accNo = number;

accBal = balance;

accType = AccountType::Checking;

}

/\*\*

\* Returns the account number.

\*

\* @return The account number.

\*

\* @throws None.

\*/

auto bank::BankAccount::getNumber() const -> long {

return accNo;

}

/\*\*

\* Returns the account balance.

\*

\* @return The account balance.

\*

\* @throws None.

\*/

auto bank::BankAccount::getBalance() const -> double {

return accBal;

}

/\*\*

\* Returns the account type.

\*

\* @return The account type.

\*

\* @throws None.

\*/

auto bank::BankAccount::getType() const -> AccountType {

return accType;

}

/\*\*

\* Sets the account number.

\*

\* @param number The account number.

\*

\* @throws None.

\*/

auto bank::BankAccount::setNumber(const long number) -> void {

accNo = number;

}

/\*\*

\* Sets the balance of the bank account.

\*

\* @param balance The new balance to be set.

\*

\* @return void

\*

\* @throws None

\*/

auto bank::BankAccount::setBalance(const double balance) -> void {

accBal = balance;

}

/\*\*

\* Sets the account type.

\*

\* @param type The new account type.

\*

\* @return void

\*

\* @throws None

\*/

auto bank::BankAccount::setType(const AccountType type) -> void {

accType = type;

}

/\*\*

\* Withdraws the specified amount from the bank account if there are sufficient funds.

\*

\* @param amount The amount to be withdrawn.

\*

\* @return True if the withdrawal was successful, false otherwise.

\*

\* @throws None

\*/

auto bank::BankAccount::withdraw(const double amount) -> bool {

const auto sufficientFunds = accBal >= amount;

if (sufficientFunds) {

accBal -= amount;

}

return sufficientFunds ;

}

/\*\*

\* Deposits the specified amount into the bank account.

\*

\* @param amount The amount to be deposited.

\*

\* @return The updated balance of the bank account after the deposit.

\*

\* @throws None

\*/

auto bank::BankAccount::deposit(const double amount) -> double {

return accBal += amount;

}

/\*\*

\* Transfers the specified amount from the given BankAccount to the current BankAccount.

\*

\* @param from The BankAccount to transfer the amount from.

\* @param amount The amount to be transferred.

\*

\* @return void

\*

\* @throws None

\*/

auto bank::BankAccount::transferFrom(BankAccount &from, const double amount) -> void {

if(from.withdraw(amount)) {

this->deposit(amount);

}

}

/\*\*

\* Returns the next account number.

\*

\* @return The next account number.

\*

\* @throws None

\*/

auto bank::BankAccount::NextNumber() -> long {

return nextNumber++;

}

**bankAccount.h**

//

// Created by dokto on 11.06.2024.

//

#pragma once

#include "AccountType.h"

namespace bank {

class BankAccount {

public:

void populate(int number, double balance);

[[nodiscard]] auto getNumber() const -> long;

[[nodiscard]] auto getBalance() const -> double;

[[nodiscard]] auto getType() const -> AccountType;

auto setNumber(long number) -> void;

auto setBalance(double balance) -> void;

auto setType(AccountType type) -> void;

auto withdraw(double amount) -> bool;

auto deposit(double amount) -> double;

auto transferFrom(BankAccount &from, double amount) -> void;

private:

auto NextNumber() -> long;

long accNo = 0;

double accBal = 0;

AccountType accType = AccountType::Checking;

long nextNumber = 123;

};

}

**CreateAccount.cpp**

//

// Created by dokto on 11.06.2024.

//

#include "CreateAccount.h"

#include "iostream"

#include "bank/AccountType.h"

namespace {

/\*\*

\* Creates a bank account with the given account number and a default balance of 100.

\*

\* @param num The account number.

\*

\* @return A bank::BankAccount object representing the newly created account.

\*

\* @throws None.

\*/

auto createBankAccount(const int num) -> bank::BankAccount {

bank::BankAccount tmp ;

tmp.populate( num, 100);

return tmp;

}

}

/\*\*

\* Executes the process of creating a bank account.

\*

\* This function creates a new bank account by calling the `creatBankAccount` function

\* and stores the result in the `bankAccount` variable. Then, it calls the `write`

\* function to display the details of the bank account.

\*

\* @return void

\*/

auto bank::CreateAccount::process() -> void {

auto bankAccount1 = createBankAccount(1);

write(bankAccount1);

auto bankAccount2 = createBankAccount(2);

write(bankAccount2);

std::cout << "Transfer" << std::endl << std::endl;

bankAccount1.transferFrom( bankAccount2, 10);

write(bankAccount1);

write(bankAccount2);

}

/\*\*

\* Writes the details of a bank account to the standard output.

\*

\* @param bankAccount the bank account to write

\*

\* @return void

\*

\* @throws None

\*/

auto bank::CreateAccount::write(const bank::BankAccount &bankAccount) -> void {

std::cout << "Account number is " << bankAccount.getNumber() << std::endl;

std::cout << "Account balance is " << bankAccount.getBalance() << std::endl;

std::cout << "Account type is " <<

(bankAccount.getType() == bank::AccountType::Checking ? "Checking" : "Deposit") << std::endl << std::endl;

}

**CreateAccount.h**

//

// Created by dokto on 11.06.2024.

//

#pragma once

#include "bank/bankAccount.h"

namespace bank {

class CreateAccount {

public:

static auto process() -> void;

private:

static auto write(const bank::BankAccount &bankAccount) -> void;

};

}

### 3.3 Упражнение 3

В ходе выполнения данного упражнения, написана программа, которая считывает текст из одного файла и переписывает в другой файл, переводя все буквы в верхний регистр.

## **3.3.1 Пошаговое описание алгоритма**

Пользователем вводится название входного и выходного файла.

Сообщение из одного файла передается в другой с верхним регистром.

В случае ошибки выводится сообщение об ошибке.

### 3.3.2 Используемые классы и методы

В программе, написанной в данном упражнении, используются следующие методы:

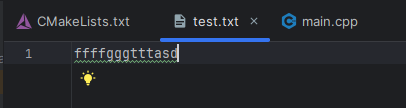
- std::cout – служит для отображения на экране строк и значений переменных, переданных в метод в качестве параметров, с переходом на новую строку;

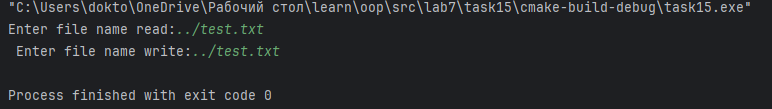
- std::cin – ожидает следующего нажатия клавиши пользователем;

- main() – служит для запуска программы.

### 3.3.3 Контрольный пример

На рис.3.3 представлены результаты выполнения программы.

****

****

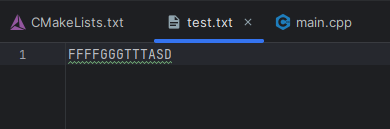
****

Рис.3.3 Контрольный пример для программы 3

Как видно из рисунка, пользователем вводятся названия файлов и в выходном файле появляется сообщение из входного файла в верхнем регистре.

# **4. Листинг программы**

**WorkFile.cpp**

//

// Created by dokto on 11.06.2024.

//

#include "WorkFile.h"

auto WorkFile::readToFile() -> void {

std::string fileName;

std::cout << "Enter file name read: ";

std::cin >> fileName;

if (std::ifstream myfile(fileName); myfile.is\_open()) {

std::string line;

while (getline(myfile, line)) {

fileData += line;

}

myfile.close();

}else {

throw std::runtime\_error("File not found");

}

}

auto WorkFile::saveToFile() const -> void {

std::string fileName;

std::cout << "Enter file name write: ";

std::cin >> fileName;

std::ofstream myfile(fileName);

if(!myfile.is\_open()){throw std::runtime\_error("File not found");}

std::string tmp;

for(const auto &i: fileData) {

tmp += static\_cast<char>(toupper(i));

}

myfile << tmp;

myfile.close();

}

**WorkFile.h**

//

// Created by dokto on 11.06.2024.

//

#pragma once

#include <string>

#include <fstream>

#include <iostream>

class WorkFile {

public:

auto readToFile() -> void;

auto saveToFile() const -> void;

private:

std::string fileData;

};

**Main.cpp**

#include <iostream>

#include "workFile/WorkFile.h"

int main()

{

try {

WorkFile workFile;

workFile.readToFile();

workFile.saveToFile();

}catch (std::logic\_error &ex) {

std::cout << ex.what() << std::endl;

}

}

**Четвертое приложение**

**Utils.h**

//

// Created by Юлий Максимов on 11.06.2024.

//

#pragma once

#include <string>

#include <utility>

namespace utils {

class Utils {

public:

/\*\*

\* Returns the greater of two values.

\*

\* @tparam T the type of the values being compared

\*

\* @param var1 the first value to compare

\* @param var2 the second value to compare

\*

\* @return the greater of var1 and var2

\*

\* @throws None

\*/

template <typename T>

static auto greater(T var1, T var2) -> T;

/\*\*

\* Swaps the values of two variables.

\*

\* @tparam T the type of the variables

\*

\* @param var1 the first variable

\* @param var2 the second variable

\*

\* @return void

\*

\* @throws None

\*/

template <typename T>

static auto swap(T &var1, T &var2) noexcept -> void;

/\*\*

\* Checks if the value is a string.

\*

\* @tparam T the type of the value

\*

\* @param var the value to check

\*

\* @return true if the value is a string, false otherwise

\*

\* @throws None

\*/

template <typename T>

static auto isToString(T var) -> bool;

};

template<typename T>

auto Utils::greater(T var1, T var2) -> T {

return var1 > var2 ? var1 : var2;

}

template<typename T>

auto Utils::swap(T &var1, T &var2) noexcept -> void {

std::swap(var1,var1);

}

template<typename T>

auto Utils::isToString(T var) -> bool {

if(std::is\_same\_v<T, std::string>) {

return false;

}

if(std::is\_same\_v<T, int> ||

std::is\_same\_v<T, double> ||

std::is\_same\_v<T, bool> ||

std::is\_same\_v<T, unsigned long>)

{

return true;

}

return false;

}

}

**Main.cpp**

#include <iostream>

#include "utils/utils.h"

/\*\*

\* The main function of the C++ program.

\*

\* @return an integer representing the exit status of the program

\*

\* @throws None

\*/

int main()

{

constexpr int x = 1;

constexpr unsigned long y = 2;

const std::string z = "3";

std::cout << "int is toString: " << (utils::Utils::isToString(x) ? "true" : "false") << std::endl;

std::cout << "unsigned long is toString: " << (utils::Utils::isToString(y) ? "true" : "false") << std::endl;

std::cout << "string is toString: " << (utils::Utils::isToString(z) ? "true" : "false") << std::endl;

}

### 3.4 Упражнение 4

В ходе выполнения данного упражнения, написана программа, которая добавляет статический метод.

## **3.4.1 Пошаговое описание алгоритма**

Инициализируются три переменные.

Переменные выводятся на экран после метода IsItFormattable.

### 3.4.2 Используемые классы и методы

В программе, написанной в данном упражнении, используются следующие методы:

- std::cout – служит для отображения на экране строк и значений переменных, переданных в метод в качестве параметров, с переходом на новую строку;

- std::cin – ожидает следующего нажатия клавиши пользователем;

- main() – служит для запуска программы.

- Utils – класс, который взят из прошлой лабораторной работы и добавлен метод IsItFormattable.

### 3.4.3 Контрольный пример

На рис.3.4 представлены результаты выполнения программы.

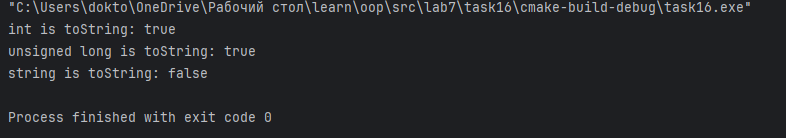
****

Рис.3.4 Контрольный пример для программы 4

Как видно из рисунка, выводятся результаты после метода IsItFormattable.

# **4. Листинг программы**

**Utils.h**

//

// Created by Юлий Максимов on 11.06.2024.

//

#pragma once

#include <string>

#include <utility>

namespace utils {

class Utils {

public:

/\*\*

\* Returns the greater of two values.

\*

\* @tparam T the type of the values being compared

\*

\* @param var1 the first value to compare

\* @param var2 the second value to compare

\*

\* @return the greater of var1 and var2

\*

\* @throws None

\*/

template <typename T>

static auto greater(T var1, T var2) -> T;

/\*\*

\* Swaps the values of two variables.

\*

\* @tparam T the type of the variables

\*

\* @param var1 the first variable

\* @param var2 the second variable

\*

\* @return void

\*

\* @throws None

\*/

template <typename T>

static auto swap(T &var1, T &var2) noexcept -> void;

/\*\*

\* Checks if the value is a string.

\*

\* @tparam T the type of the value

\*

\* @param var the value to check

\*

\* @return true if the value is a string, false otherwise

\*

\* @throws None

\*/

template <typename T>

static auto isToString(T var) -> bool;

};

template<typename T>

auto Utils::greater(T var1, T var2) -> T {

return var1 > var2 ? var1 : var2;

}

template<typename T>

auto Utils::swap(T &var1, T &var2) noexcept -> void {

std::swap(var1,var1);

}

template<typename T>

auto Utils::isToString(T var) -> bool {

if(std::is\_same\_v<T, std::string>) {

return false;

}

if(std::is\_same\_v<T, int> ||

std::is\_same\_v<T, double> ||

std::is\_same\_v<T, bool> ||

std::is\_same\_v<T, unsigned long>)

{

return true;

}

return false;

}

}

**Main.cpp**

#include <iostream>

#include "utils/utils.h"

/\*\*

\* The main function of the C++ program.

\*

\* @return an integer representing the exit status of the program

\*

\* @throws None

\*/

int main()

{

constexpr int x = 1;

constexpr unsigned long y = 2;

const std::string z = "3";

std::cout << "int is toString: " << (utils::Utils::isToString(x) ? "true" : "false") << std::endl;

std::cout << "unsigned long is toString: " << (utils::Utils::isToString(y) ? "true" : "false") << std::endl;

std::cout << "string is toString: " << (utils::Utils::isToString(z) ? "true" : "false") << std::endl;

}

### 3.5 Упражнение 5

В ходе выполнения данного упражнения, написана программа, которая добавляет метод Display, который использует оператор as для определения передачи объекта как параметра в интерфейс IPrintablе.

## **3.5.1 Пошаговое описание алгоритма**

Переменные инициализируются.

Метод Display.

Вывод результата на экран.

### 3.5.2 Используемые классы и методы

В программе, написанной в данном упражнении, используются следующие методы:

- std::cout – служит для отображения на экране строк и значений переменных, переданных в метод в качестве параметров, с переходом на новую строку;

-std::cin – ожидает следующего нажатия клавиши пользователем;

- main() – служит для запуска программы.

- Utils – класс, который взят из прошлой лабораторной работы и добавлен метод Display.

### 3.5.3 Контрольный пример

На рис.3.5 представлены результаты выполнения программы.

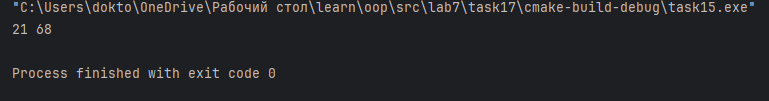
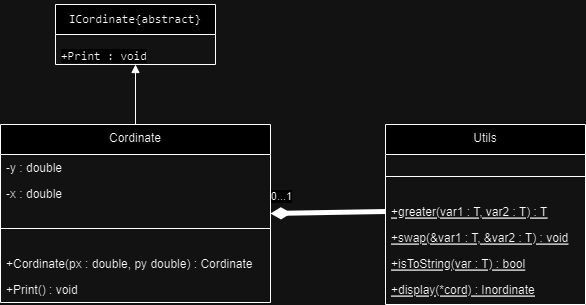
****

Рис.3.5 Контрольный пример для программы 4

Как видно из рисунка, выводятся результаты после метода Display.

**3.5.4 Диаграмма**



# **3.5.5 Листинг программы**

**Cordinate.h**

//

// Created by dokto on 11.06.2024.

//

#pragma once

#include <iostream>

#include <cord/ICord.h>

class Cordinate final : public ICordinate{

public:

/\*\*

\* Constructs a Cordinate object with the default coordinates (0.0, 0.0).

\*

\* @throws None

\*/

Cordinate() {

x=0.0;

y=0.0;

}

/\*\*

\* Constructs a Cordinate object with the given coordinates.

\*

\* @param px the x-coordinate

\* @param py the y-coordinate

\*/

Cordinate(const double px, const double py) {

x=px;

y=py;

}

/\*\*

\* Prints the x and y coordinates of the Cordinate object to the standard output.

\*

\* @throws None

\*/

void Print() const override {

std::cout << x << " " << y << std::endl;

}

private:

double x{};

double y{};

};

**ICord.h**

//

// Created by dokto on 11.06.2024.

//

#pragma once

class ICordinate {

public:

virtual ~ICordinate() = default;

virtual void Print() const = 0;

};

**Utils.h**

//

// Created by Юлий Максимов on 11.06.2024.

//

#pragma once

#include <string>

#include <utility>

#include <cord/ICord.h>

namespace utils {

class Utils {

public:

/\*\*

\* Returns the greater of two values.

\*

\* @tparam T the type of the values being compared

\*

\* @param var1 the first value to compare

\* @param var2 the second value to compare

\*

\* @return the greater of var1 and var2

\*

\* @throws None

\*/

template <typename T>

static auto greater(T var1, T var2) -> T;

/\*\*

\* Swaps the values of two variables.

\*

\* @tparam T the type of the variables

\*

\* @param var1 the first variable

\* @param var2 the second variable

\*

\* @return void

\*

\* @throws None

\*/

template <typename T>

static auto swap(T &var1, T &var2) noexcept -> void;

/\*\*

\* Checks if the value is a string.

\*

\* @tparam T the type of the value

\*

\* @param var the value to check

\*

\* @return true if the value is a string, false otherwise

\*

\* @throws None

\*/

template <typename T>

static auto isToString(T var) -> bool;

/\*\*

\* Displays the coordinates of a given ICordinate object.

\*

\* @param cord Pointer to the ICordinate object to display.

\*

\* @return void

\*

\* @throws None

\*/

static auto display(const ICordinate \*cord) -> void;

};

inline auto Utils::display(const ICordinate \*cord) -> void {

cord->Print();

}

template<typename T>

auto Utils::greater(T var1, T var2) -> T {

return var1 > var2 ? var1 : var2;

}

template<typename T>

auto Utils::swap(T &var1, T &var2) noexcept -> void {

std::swap(var1,var1);

}

template<typename T>

auto Utils::isToString(T var) -> bool {

if(std::is\_same\_v<T, std::string>) {

return false;

}

if(std::is\_same\_v<T, int> ||

std::is\_same\_v<T, double> ||

std::is\_same\_v<T, bool> ||

std::is\_same\_v<T, unsigned long>)

{

return true;

}

return false;

}

}

**Main.cpp**

#include <iostream>

#include "utils/utils.h"

#include "cord/Cordinate.h"

#include "memory"

/\*\*

\* The main function of the C++ program.

\*

\* @return 0 indicating successful execution.

\*

\* @throws None.

\*/

int main()

{

const std::unique\_ptr<ICordinate> cord = std::make\_unique<Cordinate>(21.0,68.0);

utils::Utils::display(cord.get());

return 0;

}

# **5. Полученные результаты**

В ходе выполнения данной лабораторной работы нами были получены следующие результаты:

* В ходе работы программы 1 были созданы методы с параметры и добавлены в класс.
* В ходе программы 2 были использованы методы со ссылочными параметрами.
* В ходе программы 3 были преобразованы символы файла в верхний регистр.
* В ходе программы 4 была проведена проверка реализация интерфейса.
* В ходе программы 5 была произведена работа с интерфесами.

# **6. Выводы**

В ходе выполнения данной лабораторной работы:

* были изучены переменные ссылочного типа в языке C++;
* были изучены интерфейсы в языке С++.