**Тема17.Перевантаження операторів потокового введення/виведення. Форматування виводу. Обробка виключень.**

**Особливості механізмів перевантаження операторів введення-виведення даних**

У мові програмування C++ передбачено спосіб виконання операцій введення-виведення даних класів шляхом перевантаження операторів введення-виведення "<<" та ">>".

У мові C++ оператор "<<" називається оператором виведення або вставлення, оскільки він вставляє символи у потік. Аналогічно оператор ">>" називається оператором введення або вилучення, оскільки він вилучає символи з потоку. Оператори введення-виведення вже перевантажені (у заголовку ) для того, щоби вони могли виконувати операції потокового введення або виведення даних будь-яких вбудованих С++-типів. Розглянемо як можна визначити ці оператори для створення власних класів.

**Створення перевантажених операторів виведення даних**

Як простий приклад розглянемо механізм створення оператора виведення даних для класу **kooClass** на наступній програмі.

**#include <iostream>**

**#include <conio.h>**

**#include <string.h>**

**#include <stdio.h>**

**#include <Windows.h>**

**using namespace std;**

**class kooClass { public: int x, y, z; kooClass(int a, int b, int c)**

**{ x = a; y = b; z = c; };**

**};**

/**/ Перевизначений оператор введення даних**

**ostream &operator<<(ostream &stream, const kooClass &obj)**

**{ stream << obj.x << ", ";**

**stream << obj.y << ", ";**

**stream << obj.z << endl;**

**return stream;**

**}**

**main()**

**{system("color F0");**

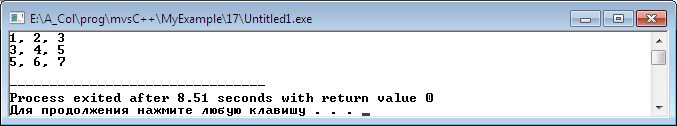
**kooClass ObjA(1, 2, 3), ObjB(3, 4, 5), ObjC(5, 6, 7);**

**cout << ObjA << ObjB << ObjC;**

**getch();**

**}**

Результат



Щоб створити операторну функцію виведення даних для об'єктів типу kooClass, необхідно перевантажити оператор виведення даних "<<". Один з можливих способів його реалізації має вигляд наданий після тексту "//ºПеревизначений оператор введення даних ". Ця операторна функція, вміст якої характерний для багатьох операторних функцій виведення даних, згідно з оголошенням, повертає посилання на об'єкт типу ostream. Це дає змогу декілька звичайних операторів виведення даних об'єднати в одному складеному виразі. Ця функція має два параметри. Перший є посиланням на потік, який використовується в лівій частині оператора "<<". Другим є об'єкт, який знаходиться у правій частині цього оператора. Саме тіло операторної функції складається з настанов виведення трьох значень координат, що містяться в об'єкті типу kooClass, і настанови повернення потоку stream.

Для будь-якій операторній функції виведення даних оператор перевантаження задається за схемою:

**ostream &operator<<(ostream &stream, class\_type obj)**

**{ // Код операторної функції виведення даних**

**return stream; // Повертає посилання на параметр stream**

**}**

У такій версії операторної функції жорстко закодований потік cout. Це обмежує перелік ситуацій, в яких її можна використовувати. Отже, програміст повинен передавати операторній функції виведення даних потік, який коректно працює в усіх конкретних випадках.

**Використання функцій-"друзів" класу для перевантаження операторів виведення даних**

У попередній програмі операторну функцію виведення даних не було визначено як член класу kooClass. Насправді ні будь-яка операторна функція виведення даних, ні функція їх введення не можуть бути членами класу. Причина в тому, що якщо операторна функція є членом класу, то лівий операнд (що опосередковано передається за допомогою показника this) повинен бути об'єктом класу, який генерує звернення до цієї операторної функції. І це змінити не можна. Проте при перевантаженні операторів виведення даних лівий операнд повинен бути потоком, а правий – об'єктом класу, дані якого підлягають виведенню. Отже, перевантажені оператори виведення даних не можуть бути функціями-членами класу. У зв'язку з тим, що операторні функції виведення даних не можуть бути членами класу, для якого вони визначаються, то виникає серйозне запитання: як перевантажений оператор виведення даних може отримати доступ до закритих членів класу?

У попередній програмі змінні x, y і z були визначені як відкриті, тому оператор виведення даних без перешкод міг отримати до них доступ. Для закритих членів класу оператор виведення даних можна зробити "другом" класу, і функція - "друг" отримує легальний доступ до його private-даних. Оголошення "другом" класу операторної функції виведення даних продемонструємо на прикладі класу kooClass.

Приклад. Демонстрація механізму використання функцій-"друзів" класу для перевантаження оператора виведення даних – опис класу kooClass змінено.

**class kooClass { private: int x, y, z;**

**public: kooClass(int a, int b, int c)**

**{ x = a; y = b; z = c; };**

**friend ostream &operator<<(ostream &stream, const kooClass &obj);**

**};**

**Створення перевантажених операторів введення даних**

Для перевантаження операторів введення даних використовують аналогічний підхід, який було застосовано при перевизначенні операторів виведення даних. В теорії перевантажений оператор виглядає так:

// Прийняття тривимірних координат x, y, z

// Перевантажений оператор введення даних для класу

istream &operator>>(istream &stream, kooClass &obj)

{ cout << "Введіть координати x, y і z: ";

// Перевантажений оператор введення даних

stream >> obj.x >> obj.y >> obj.z;

return stream;

// Повертає посилання на параметр stream }

Перевантажений оператор введення даних повинен повертати посилання на об'єкт типу istream. Окрім цього, перший параметр повинен бути посиланням на об'єкт типу istream. Цей тип належить потоку, що вказується зліва від оператора ">>". Другий параметр є посиланням на об'єкт, який приймає значення, що вводяться. Оскільки другий параметр – посилання на об'єкт, то його можна модифікувати при введенні інформації. Подібно до операторних функцій виведення даних, операторні функції їх введення не можуть бути членами класу, для оброблення даних якого вони призначені. Вони можуть бути тільки "друзями" цього класу або просто незалежними операторними функціями. За винятком того, що операторна функція введення даних повертає посилання на об'єкт типу istream, тіло цієї функції може містити все те, що програміст вважає за потрібне у неї помістити. Але логічніше використовувати оператори введення, а також виведення, все ж таки за прямим призначенням, тобто для виконання операцій введення даних.

Загальний формат перевизначеного оператора введення даних має такий вигляд:

**istream &operator>>(istream &stream, objectType &obj)**

**{ // код операторної функції введення даних**

**return stream;**

**// Повертає посилання на параметр stream**

**}**

# Потрібно зробити зауваження, що запропонована теоретична конструкція може викликати помилку при компіляції " Ambiguous overload for operator>>", яка викликана проблемами перетворення типів .

Нижче представлений працюючий код.

**#include <iostream>**

**#include <fstream>**

**#include <stdlib.h>**

**#include <conio.h>**

**#include <string.h>**

**#include <math.h>**

**#include <stdio.h>**

**#include <Windows.h>**

**using namespace std;**

**class kooClass**

**{**

**private:**

**public: int x, y, z;**

**kooClass(int a, int b, int c)**

**{ x = a; y = b; z = c; };**

**friend ostream &operator<<(ostream &stream, const kooClass &obj);**

**friend istream &operator>>(istream &stream, const kooClass &obj);**

**};**

**ostream &operator<<(ostream &stream, const kooClass &obj)**

**{ stream << obj.x << ", ";**

**stream << obj.y << ", ";**

**stream << obj.z << endl;**

**return stream;**

**};**

**istream &operator>>(istream &stream, kooClass &obj)**

**{ cout << "Enter x, y , z: ";**

**// Бажаний перевантажений оператор виведення**

**//stream>>obj.x >> obj.y>>obj.z;**

**char tmp[10];**

**stream>> tmp;**

**obj.x=atoi(tmp);**

**stream>> tmp;**

**obj.y=atoi(tmp);**

**stream>> tmp;**

**obj.z=atoi(tmp);**

**return stream;**

**// повертає посилання на поток stream**

**};**

**main()**

**{system("color F0");**

**kooClass ObjA(1, 2, 3), ObjB(3, 4, 5), ObjC(5, 6, 7);**

**cout << ObjA << ObjB << ObjC;**

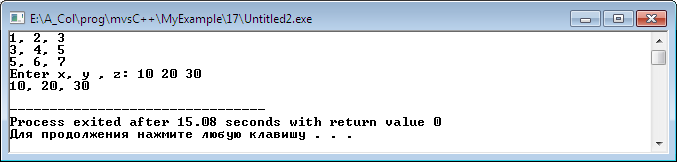
**cin >> ObjA;**

**cout << ObjA;**

**getch();**

**}**

Ось як виглядає результат виконання цієї програми:



**Форматоване виведення**.

Здійснюєтьсяфункцією:

* **printf** - виводить аргументи у стандартний потік stdout у відповідності із заданим форматом.

Щоб зв’язати програму користувача зі стандартною бібліотекою, де знаходяться ці функції, необхідно на початку програми включити заголовний файл **stdio.h**.

***Формат виклику функції printf*:**

**printf("форматний\_рядок", [список\_аргументів])**

Де *список\_аргумен*тів = аргумент {, аргумент}

*форматний\_рядок* = “”” ([літерал] | % [маркер][ширина][.точність] тип)

{( [літерал] | % [маркер][ширина][.точність] тип}”””

Форматний рядок завжди береться в подвійні лапки. Символ "%" є символом початку задання формату введення чергового аргумента. При описі формату виведення також використовуються:

*маркер* – специфікатор вирівнювання виведених знаків, пропусків, десяткових точок, вісімкових і шістнадцяткових префіксів (наприклад, вирівнювання результату перетворення по лівій межі, обов'язкове зображення знака числа тощо);

ширина– загальна ширина поля виведення;

*точність* – максимальне число цифр дробової частини числа, які будуть виведені після коми;

*тип* – специфікатор типу аргументу (наприклад, d – ціле десяткове число, f – дійсне число).

Функція printf переводить дані з внутрішнього коду в символьне представлення відповідно до форматного рядка і виводить отримані символи на екран. Форматний рядок може включати довільний текст, керуючі символи та специфікації перетворення даних. Список аргументів є необов'язковим параметром даної функції. Дану функцію можна використовувати для виводу будь-якої комбінації символів, цілих та дійсних чисел, тощо.

Функція printf може використовуватися, наприклад, для виведення повідомлення на екран:

**printf ("**Enter the source data **\n");**

Для звертання до функції використовуються параметри, розташовані у круглих дужках. Найчастіше функція **printf** реалізується для виведення значень змінних. Першим аргументом у звертанні до функції ставиться рядок форматів (береться в лапки), а наступними, якщо вони є, — об’єкти, що виводяться.

Рядок форматів може включати звичайні символи, які копіюються при виведенні, і специфікації перетворення, що починаються із символу«% », за специфікаціями йде символ перетворення. Кожна специфікація перетворення відповідає одному з аргументів, що йдуть за форматним рядком, і між ними встановлюється взаємно однозначна відповідність, наприклад:

**printf (“Values a, b, c are equal: %d %d. %d \n”, а, b, с);**

тут літера **d** у специфікації перетворення вказує, що значення аргументу має бути представлено як десяткове ціле число.

Список форматних кодів має таку форму запису:

**% [прапорець] [довжина] [точність] [f | n] [h | l] тип ,**

де **прапорець** — символ, що керує вирівнюванням виведення і виведенням пропусків, десяткової крапки, ознак чисел вісімкової і шістнадцяткової систем числення. **Прапорець** може задаватися одним із символів:

**«-»** — вирівнювання вліво усередині заданого поля;

**«+»** — виведення знака числа;

**« »** (пропуск) — приєднання пропуску до виведеного числа, якщо число є додатним і має тип зі знаком;

**«#»** — виводиться ідентифікатор системи числення для цілих: **0** — для вісімкових чисел, **0х** чи **0Х** — для шістнадцяткових чисел;

**довжина** — визначає мінімальну кількість виведених символiв, якщо довжина більше виведеної кількості символів, то виведене значення доповнюється пропусками, у випадку, коли довжина менше кількості символів у виведеному значенні або вона не задана, виводяться всі символи значення (відповідно до поля точність, якщо воно є);

**точність** — задається цілим числом після крапки і визначає кількість виведених символів, кількість знаків після крапки; на відміну від поля довжини поле точність може привести до «зрізання» виведених даних.

**f | n** — дозволяють приглушити погодження за замовчуванням про використану модель пам’яті («далека», «близька» пам’ять);

**h | l** — предикати, що визначають відповідно аргументи типів **short і long**;

**тип** — задається одним із символів: **d** — десяткове ціле; **і** — десяткове, вісімкове чи шістнадцяткове ціле зі знаком; **с** — одиночний символ;**u** — беззнакове десяткове число; **х, X** — беззнакове шістнадцяткове число; **0** — вісімкове число; **s** — сприймає символи без перетворення до символу **«\n»** або пропуску, доки не буде досягнута задана довжина (при виведенні видає до потоку всі символи до символу **«\0»** або до досягнення специфікованої точності); **f, F** — значення з плаваючою крапкою; **е, Е** — значення у експоненціальній формі; **G, g** — значення зі знаком у формі **f** або **е**.

**Символи формату**

| **Символ**  **формату** | **Тип**  **виводимого значення** |
| --- | --- |
| c, С | **char; одиничний літерал (символ)** |
| d, i | **int**; цілі зі знаком |
| o | **int**; без знакові цілі у восьмирічній системі числення |
| u | **int**; цілі зі знаком |
| x, X | **int**; беззнакові цілі у шістнадцятирічній системі числення |
| f | **float**; дійсні числа типу **float** (фіксована крапка) |
| lf | **double**; дійсні числа типу **double** (фіксована крапка) |
| e, E | **double**; дійсні числа в експоненційній формі (плаваюча крапка) |
| g, G | **double**; дійсні числа |
| P | **void**; вказівник |
| % | знак % |

**Керуючі символи(*Escape* – послідовності)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Chr** | **Dec** | **Hex** | **Значення** |
| '\a' | '\07' | '\x07' | Звуковий сигнал |
| '\b' | '\08' | '\x08' | Повернення на одну позицію назад (Backspace) |
| '\f' | '\14' | '\x0c' | Перехід на нову сторінку |
| '\n' | '\12' | '\x0a' | Перехід на новий рядок |
| '\r' | '\15' | '\x0d' | Повернення каретки (повернення на початок рядка) |
| '\t' | '\11' | '\x09' | Горизонтальна табуляція |
| '\v' | '\13' | '\x0b' | Вертикальна табуляція |
| '\\' | '\134' | '\x5c' | Зворотна коса риска |
| '\'' | '\47' | '\x27' | Апостроф |
| '\"' | '\42' | '\x22' | Подвійні лапки |
| '\?' | '\77' | '\x77' | Знак питання |

Виведення результатів з використанням форматних кодів функції **printf** може мати вигляд:

**printf (” % 3.0 f % 6.1 f \ n ”, х, у);**

***Приклад .*** Обчислити значення функції **у = ах2 – sinx**, якщо **а =  10,5; х є [-1; 2]; hx = 0,5**.

**#include <stdio.h>**

**#include <math.h>**

**#include <conio.h>**

**main( )**

**{ float x, y, a(10.5);**

**printf ("\t Vivod rezultata\n");**

**for (x = -1; x <= 2; x += 0.5)**

**{** **у = a \* pow(x,2) - sin(x);     //у = a\*x\*x - sin(x);**

**printf (" \t x = % 4.1f    у = % 6.3f \n", x, y);**

**}**

**getch ();** *//затримка екрана*

**}**

Результати обчислення:  
**Vivod rezultata**  
**x = -1.0 у =11.342**  
**х = -0.5 у = 3.104**  
**х = 0.0  у = 0.000**  
**х = 0.5  у = 2.146**  
**х = 1.0  у = 9.659**  
**х = 1.5  у = 22.628**  
**х = 2.0  у = 41.091.**

**Виключні або виняткові ситуації**

У мові С++ існує механізм обробки виняткових ситуацій. У ході виконання програми можуть виявитися різні помилки. Вони можуть бути позв'язані з неправильним програмуванням (наприклад, вихід індексу масиву за межі припустимого чи переповнення пам'яті), а іноді їхня причина не залежить від програміста (скажемо, розрив зв'язку при мережевому з'єднанні). У кожній з цих ситуацій реакція програми непередбачена. Іноді вона завершує виконання, і лише після закінчення деякого інтервалу часу починають позначатися наслідку помилки, а частіше програма негайно припиняє роботу, піддаючи ризику дані, що знаходяться в пам'яті чи у файлі. Якщо не передбачити акуратне завершення роботи, використовуючи обробку виняткових ситуацій, результати можуть виявитися неприємними. В подальшому будемо називати винятковою ситуацією будь-яку подію, що вимагає особливої обробки. При цьому зовсім неважливо, чи є ця подія фатальною чи простою помилкою. Перевірка умов, що описують виняткову ситуацію, і реакція на її виникнення називається обробкою виняткової ситуації. Ця задача покладається на оброблювача виняткової ситуації.

**Механізм обробки виняткових ситуацій**

Обробка виняткових ситуацій у мові С++ є об‘єктно-орієнтованою. Це значить, що виняткова ситуація є об'єктом, що генерується при виникненні незвичайних умов, передбачених програмістом, і передається оброблювачу, що неї перехоплює. Об'єктом, що описує природу виняткової ситуації, може бути будь-як сутність — літерал, рядок, об'єкт класу, число і т.д. Виняткова ситуація не обов'язково повинна бути об'єктом якого-небудь класу. В основі обробки виняткових ситуацій у мові С++ лежать три ключових слова: **try**, **catch** і **throw**.

Якщо програміст підозрює, що визначений фрагмент програми може спровокувати помилку, він повинний занурити цю частину коду в блок *try*. Необхідно мати на увазі, що зміст помилки (за винятком стандартних ситуацій) визначає сам програміст. Це значить, що програміст може задати будь-яку умову, що приведе до створення виняткової ситуації. Після цього необхідно вказати, у яких умовах варто генерувати виняткову ситуацію. Для цієї мети призначене ключове слово *throw*. І нарешті, виняткову ситуацію потрібно перехопити й обробити в блоці *catch*. Ось як виглядає ця конструкція.

**try {**

**// Тіло блоку**

**try if(умова)throw виняткова\_ситуація**

**}**

**catch(тип1 аргумент)**

**{**

**// Тіло блоку catch**

**}**

**catch(тип2 аргумент)**

**{**

**// Тіло блоку catch**

**}**

**. . .**

**catch(типN аргумент)**

**{**

**// Тіло блоку catch**

**}**

Розмір блоку try не обмежений. У нього можна занурити як один оператор, так і цілу програму. Один блок ***try*** можна зв'язати з довільною кількістю блоків ***catch***. Оскільки кожен блок catch відповідає окремому типу виняткової ситуації, програма сама визначить, який з них виконати. У цьому випадку інші блоки catch не виконуються. Кожен блок catch має аргумент, що приймає визначене значення. Цей аргумент може бути об'єктом будь-якого типу.

Якщо програма виконана правильно й у блоці try не виникло жодної виняткової ситуації, усі блоки ***catch*** будуть зігноровані. Якщо в програмі виникла подія, що програміст вважає небажаною, оператор ***throw*** генерує виняткову ситуацію. Для цього оператор throw повинний знаходитися усередині блоку ***try*** або усередині функції, викликуваної усередині блоку ***try***.

Якщо в програмі виникла виняткова ситуація, для якої не передбачені перехоплення й обробка, викликається стандартна функція ***terminate()***, що, у свою чергу, викликає функцію ***abort().*** Утім, іноді виняткова ситуація не є небезпечною. У цьому випадку можна виправити помилку (наприклад, привласнити нульовому знаменнику ненульове значення) і продовжити виконання програми.

**Приклади обробки виняткової ситуації**

Розглянемо найпростіший приклад обробки виняткової ситуації

**#include <iostream>**

**#include <Windows.h>**

**using namespace std;**

**int main()**

**{system("color F0");**

**int n = 10, m = 0;**

**printf("Begin\n");**

**try {**

**printf("In try block\n");**

**if(m==0) throw "Divide by zero";**

**else n=n/m;**

**printf("The rest of the block is not executed!");**

**}**

**catch (const char\* s)**

**{**

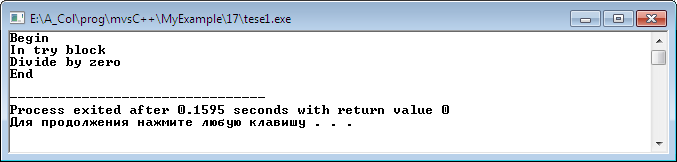
**printf("%s\n",s);**

**}**

**printf("End\n"); return 0;**

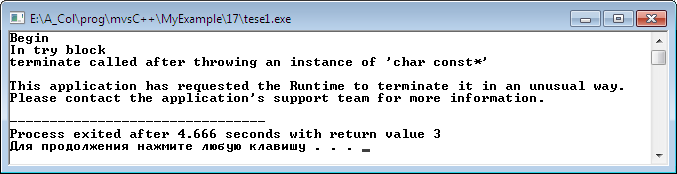
**}**

Ця програма виводить на екран наступні рядки.



Простежимо за потоком керування при виконанні цієї програми. Спочатку з'являються і ініціалізуються дві цілочисельні змінні (одна з них дорівнює нулю). Потім виводиться повідомлення про початок виконання програми, і потік керування входить у блок ***try.*** Після виводу рядка повідомлення про вхід у блок try, потік керування переходить до перевірки рівності ***m==0***. Оскільки ця рівність є істиною, генерується виняткова ситуація (у даному випадку — константний рядок). Керування негайно передається блоку ***catch***, аргументом якого є константний символьний вказівник, ігноруючи всі інші оператори в блоці ***try***. У цій програмі блок catch не робить жодних спроб виправити помилку. Замість цього він просто видає повідомлення — рядок, отриманий як аргумент — і передає керування оператору, що слідує за блоком. На закінчення функція printf() виводить на екран рядок **End**, і програма завершує свою роботу. Тип виняткової ситуації повинний збігатися з типом аргументу розділу catch.

Поглянемо, що відбудеться, якщо цією умовою зневажити. Порушення угоди про тип виняткової ситуації можна викликати заміною оператору **catch (const char\* s)** на **(const char s).** Отримаємо результат:



У цій програмі ми зробили цілком ―природну‖ помилку — забули поставити зірочку в оголошенні аргументу. Тепер блок catch очікує виняткову ситуацію, що представляє собою константний символ, а не вказівник. Ця помилка приводить до аварійного завершення роботи програми. Покажемо, що відбудеться, якщо виняткова ситуація генерується усередині функції, яка викликається в блоці try. Виняткова ситуація, згенерована усередині функції **#include <iostream>**

**#include <Windows.h>**

**using namespace std;**

**int Denominator(int);**

**int main()**

**{ system("color F0");**

**int n = 10, m; printf("Begin\n");**

**try { printf("In try block\n");**

**m=Denominator(0);**

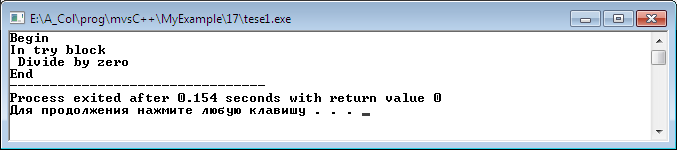
**printf("The rest of the block is not executed!"); }**

**catch (const char\* s) { printf("%s\n",s); }**

**printf("End"); return 0; }**

**int Denominator(int i) { if(i==0)throw " Divide by zero"; return i; }**

Результат:



У цій програмі виняткова ситуація генерується у функції Denominator(), яка викликається в блоці try. Завдяки цьому результати роботи програми цілком збігаються з попередніми. Якщо блок try знаходиться усередині функції, обробка виняткової ситуації виконується при кожнім виклику.

**Розміщення блоку try усередині функції**

Приклад.

**#include <iostream>**

**#include <stdio.h>**

**#include <Windows.h>**

**using namespace std;**

**int Denominator(int);**

**int main()**

**{ system("color F0");**

**int n = 10, m; printf("Begin\n");**

**m=Denominator(0);**

**n = Denominator(11);**

**printf("End"); return 0; }**

**int Denominator(int i)**

**{ printf("In Denominator \n");**

**try {**

**printf("In try block\n");**

**if(i==0) throw("Divide by zero!");**

**if(i>10) throw 10;**

**printf("The rest of the block is not executed!");**

**}**

**catch (const char\* s)**

**{**

**printf("%s\n",s);**

**}**

**catch (int n)**

**{**

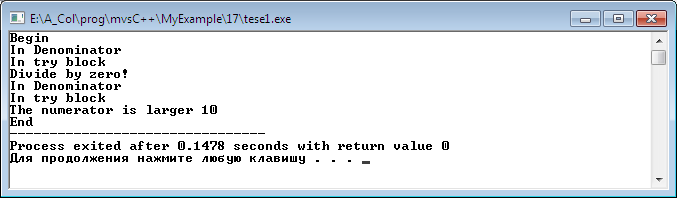
**printf("The numerator is larger %d\n",n);**

**}**

**return i;**

**}**

Результат



*Для самостійного вивчення*: Поглибити матеріал лекції за наданою літературою. Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

*Література* [1, 4, 6, 11, 12, 18]

*Контрольні запитання*.

1. Опишіть механізмів перевантаження операторів введення-виведення даних.
2. Що таке виключні ситуації.
3. Синтаксис виключних ситуацій.
4. Як контролюються виключні ситуації?
5. Як відбувається перехват виключних ситуацій?
6. Яка стандартна бібліотека застосовується при форматованому введенні-виведенні даних?
7. Запишіть специфікацію формату функції **printf**?
8. За що відповідають параметри **флаг, ширина** і **точність**? Чому вони вказані у квадратних дужках?
9. Як вказати кількість значущих розрядів цілої і дробової частини дійсного числа з фіксованою крапкою?
10. Для чого застосовуються керуючі символи (*Escape* – послідовності)? Наведіть приклади.