Instrukcja

W czasie pisania programu pamiętaj o:

1. dbaniu o czytelność kodu (odpowiednie formatowanie kodu, nazewnictwo zmiennych adekwatne do ich znaczenia, komentarze),
2. dbaniu o czytelność interfejsu z użytkownikiem (w sposób jawny pytaj użytkownika jakie dane ma podać oraz opisuj wyniki, które zwracasz),
3. przed fragmentem implementującym poszczególne zadania umieść komentarz: `/*Zadanie X */` oraz wypisz na ekranie analogiczny komunikat (X jest numerem zadania): `std::cout << "Zadanie X"<< std::endl;`,
4. każde zadanie umieść w oddzielnej funkcji (w niej dopiero należy odwoływać się do zaimplementowanych funkcji i klas),
5. zaimplementuj menu wyboru zadania, a następnie wykorzystując pętle **do–while** oraz konstrukcję **switch** wykonaj odpowiedni fragment kodu,
6. w zadaniach wymagających udzielenia komentarza bądź odpowiedzi, należy umieścić go w kodzie programu (np. w postaci komentarza albo wydrukować na ekranie),
7. w zadaniach polegających na zaprojektowaniu klasy należy utworzyć jej instancję i wykorzystać zaimplementowaną funkcjonalność.

Wprowadzenie

> *Wszystko, co może pójść źle, pójdzie źle.*
>
> — Prawo Murphy'ego

Mechanizm obsługi wyjątków w C++ pozwala reagować na wystąpienia zdarzeń (w szczególności błędów) zmieniających prawidłowy przebieg wykonywania się programu. Instrukcje, których wykonanie może skutkować wywołaniem wyjątku, umieszczane są w bloku **try** (linia 2). Obsługa wyjątków z bloku **try** wykonywana jest w następujących zaraz po nim blokach **catch**. Dla każdego bloku **try** może przypadać jeden lub więcej bloków **catch** – w zależności od liczby możliwych typów wyjątków do obsługi.

Przykładowo, blok **catch**:

- w linii 3 pozwala na obsługę wyjątku `e` będącego liczbą całkowitą;
- w linii 6 pozwala na obsługę wyjątku `e` będącego instancją klasy `std::exception` (domyślna klasa bazowa dla wyjątków z STL);

- w linii 9 obsługiwałby byłby wyjątek bez nazwy typu `std::exception`;
- w linii 12 przechwytywałby każdy wyjątek nieobsłużony w żadnym z poprzednich bloków.

```
try {
    /* do something that may fail */
} catch(int e) {
    /* handle exception */
    cout << "Error code: " << e << endl;
} catch(const std::exception& e) {
    /* handle exception */
    cout << e.what() << endl;
} catch(const std::exception&) {
    /* handle exception */
    cout << "Standard exception." << endl;
} catch(...) {
    /* handle exception */
    cout << "Default exception"; }
}
```

Analogicznie jak w przypadku konstrukcji **if–else**, jeśli dany wyjątek został już przechwycony w jednym z wcześniejszych bloków **catch** to nie będzie on przechwytywany przez pozostałe.

Do zgłaszania wyjątków służy słowo kluczowe **throw**, jak przedstawiono poniżej:

```
if (fail()) {
    // something failed
     throw std::exception("Something failed!");
}
```

Po instrukcji **throw** podawany jest obiekt który zostanie zgłoszony – w tym przypadku jest to instancja klasy `std::exception`. W tym miejscu (linia 3) natychmiast przerywany jest standardowy przebieg działania programu – so najbliższego odpowiedniego bloku **catch**. Jeśli odpowiedni blok **catch** nie został odnaleziony program kończy swoje działanie.

Zgłaszany może być wyjątek dowolnego typu, jednak zazwyczaj jest to instancja klasy dziedziczącej po `std::exception`.

Dodatkowe informacje:

- wyjątki - `http://www.cplusplus.com/doc/tutorial/exceptions/`
- lista standardowych wyjątków - `http://en.cppreference.com/w/cpp/error/exception`

Zadania

Zadanie 1

Wykorzystując mechanizm wyjątków zabezpiecz szablon klasy `Vector` z poprzednich laboratoriów przed niepoprawnymi danymi.

W konstruktorze sprawdzaj czy podana liczba wymiarów wektora jest różna od zera. Jeśli jest równa zero rzuć wyjątek `std::domain_error` ze stosownym komunikatem o błędzie.
Uwaga: Pamiętaj, że w przypadku gdy konstruktor rzuca wyjątek to konieczne jest zwolnienie wszystkich zajętych wcześniej zasobów (zrób to zanim rzucony wyjątek). W przeciwnym wypadku mogłoby dojść do wycieku pamięci (ponieważ nie zostanie wywołany destruktor dla obiektu, który nie został ostatecznie utworzony).

W przeciążeniach operatora indeksowania `[]` dodaj sprawdzanie czy podany indeks nie znajduje się poza zasięgiem danego wektora. Jeśli indeks jest niepoprawny rzuć wyjątek `std::out_of_range` ze stosownym komunikatem o błędzie.

Dla funkcji wykonujących operacje matematyczne na dwóch wektorach (operatory `+` oraz `-`) dodaj sprawdzanie czy wektory mają tę samą liczbę wymiarów. Jeśli liczby te są różne to również rzuć wyjątek `std::out_of_range` ze stosownym komunikatem o błędzie.

Po przygotowaniu odpowiedniej implementacji przetestuj zaktualizowaną przez siebie definicję szablonu klasy `Vector`. Do poniższego kodu dodaj odpowiedni blok (lub bloki) **try-catch** oraz wyświetlanie informacji o błędzie w przypadku podania niepoprawnych danych.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "Vector.hpp"

using namespace std;

int main() {
    srand(time(nullptr));
    int n;
    cout << "Podaj rozmiar wektora V1: ";
    cin >> n;
    Vector<int> v1(n);
    for (size_t i = 0; i < n; i++)
        v1[i] = rand() % 100;
    cout << "V1: " << v1 << endl << endl;

    cout << "Ktory element V1 chcesz wyswietlic? (od 0 do " << (n-1) << ")";
    cin >> n;
    cout << n << "-ty element V1 to: " << v1[n] << endl << endl;

    cout << "Podaj rozmiar wektora V2: ";
    cin >> n;
```

```
    Vector<int> v2(n);
    for (size_t i = 0; i < n; i++)
        v2[i] = rand() % 100;
    cout << "V2: " << v2 << endl << endl;

    cout << "V1 + V2: " << (v1 + v2) << endl;
    cout << "V1 - V2: " << (v1 - v2) << endl;

    cin.ignore(2);
    return 0;
}
```

Uruchom program, a następnie spróbuj wprowadzić jakieś niepoprawne dane aby sprawdzić czy wyjątki są odpowiednio obsługiwane.

Zadanie 2

Z wykorzystaniem wyjątków zabezpiecz klasę `BinomialSolver` z laboratoriów nr 1 przed podaniem funkcji kwadratowej nie posiadającej rozwiązań z dziedziny liczb rzeczywistych. Zdefiniuj nową klasę `no_real_solution` dziedziczącą po `std::runtime_error`. Nowy wyjątek powinien być rzucany w metodzie (np. w konstruktorze) wyznaczającej miejsca zerowe w przypadku, gdy równanie nie posiada rozwiązań z dziedziny liczb rzeczywistych.

Przetestuj działanie programu dla funkcji:

- $x^2 + 5x + 3$ – posiada dwa miejsca zerowe,
- $x^2 + 2x + 1$ – posiada jedno miejsce zerowe ($x_1 = x_2$),
- $6x^2 + 3x + 9$ – nie posiada miejsc zerowych.

Wskazówka 1 Przykład definicji własnej klasy reprezentującej wyjątki: `http://www.cplusplus.com/doc/tutorial/exceptions/#standard_exceptions`

Wskazówka 2 Zwróć uwagę, że `std::runtime_error` nie posiada konstruktora domyślnego.

Zadanie 3*

Wykorzystując fragmenty kodu z laboratoriów nr 4 (zadanie z liczeniem miejsc zerowych i pochodnej różnych funkcji) zabezpiecz swoją implementację przed niepoprawnymi danymi. Przeanalizuj wszystkie funkcje matematyczne, które zostały zaimplementowane pod kątem domeny i w przypadku niepoprawnej wartości argumentu x zgłoś odpowiedni wyjątek. Na przykład dla funkcji $1/x$ niepoprawną wartością jest $x = 0$ – mamy błąd dzielenie przez 0.

Zaimplementuj i zabezpiecz również kolejne funkcje (analogicznie do poprzednich):

a) $tan(x)$
b) $log(x)$

Kod związany ze sprawdzaniem poprawności oraz zgłaszaniem wyjątków możesz umieścić, albo w odpowiednim setterze (należy pamiętać o jego odpowiednim przeciążeniu), albo bezpośrednio przed liczeniem wartości funkcji (preferowana jest wersja pierwsza).

Przeanalizuj kod związany z zaimplementowanymi algorytmami numerycznymi i tam również umieść zgłaszanie odpowiednich wyjątków.

W programie głównym korzystając z konstrukcji **try–catch** obsłuż wyrzucane wyjątki i wyświetl użytkownikowi stosowny komunikat, a następnie podejmij odpowiednie działanie (np. ponownie zapytaj o wartość).

Wskazówka W przypadku, gdy wartość x nie należy do domeny implementowanej funkcji, można zgłosić wyjątek typu `std::domain_error`.