Zadanie 1

Sprawdź działanie obsługi wyjątków/błędów (exception handling) w języku C++

- Utwórz dowolną funkcję, która zwraca wyjątek w postaci:

```
throw "To jest wyjatek!";
```

- Funkcję tę wywołaj w funkcji `main()` i przechwyć wyjątek:

```
try
{
    ....
}
catch (char* s)
{
    ....
}
```

- Podejrzyj debuggerem co zawiera zmienna `s`, a następnie wydrukuj jej wartość na ekran w sekcji `catch`. Sprawdź co się stanie jeśli:
  - nie umieścisz wywołania funkcji w bloku `try - catch`
  - zmienisz typ wyjątku w bloku catch na inny: np. `int`
- Dodaj jeszcze funkcję, która zwróci jako wyjątek liczbę całkowitą. Funkcję tę wywołaj wewnątrz pierwszej funkcji. Wyjątek przechwyć wewnątrz funkcji `main()`.
- Utwórz klasę abstrakcyjną `Except` która ma metodę wirtualną do drukowania informacji o wyjątku (np. `PrintInfo()`)
- Utwórz klasę konkretną np. `Except1` która będzie posiadała implementację metody `PrintInfo()`
- W funkcji main umieść instrukcje służące do przechwytywania wyjątków:

```
try
{
    ....
}
catch ( Except& e)
{
    e.PrintInfo();
}
```

- W sekcji try wywołaj jakąś funkcję, która rzuca wyjątek `Except1`:

```
throw Except1();
```

- Spróbuj zrobić to samo dla nowej klasy `Except2` która będzie drukowała informacje w której linii kodu został rzucony wyjątek (użyj zmiennej preprocesora `__LINE__`). (Inne popularne zmienne preprocesora to: `__FILE__`, `__DATE__`, `__TIME__`.)
- Sprawdź działanie sekcji `catch(...)` do przechwytywania wszystkich wyjątków. Dopisz ją poniżej już istniejącej sekcji `catch` i spróbuj rzucić wyjątek który nie dziedziczy po typie `Except`.

Zadanie 2

Wykorzystanie prostych wzorców (template'ów)

- Napisz wzorce (template'y) funkcji `mymin` i `mymax` liczące odpowiednio minimum i maksimum z dwóch argumentów. Sprawdź działanie tych funkcji dla różnych typów np. `int`, `double`.
- Sprawdź działanie funkcji `mymax` dla następującego wywołania:

```
cout << mymax( 1., 2 );
```

- Jeśli to niezbędne popraw kod programu, aby powyższe wywołanie funkcji było poprawne.

- Co jest konieczne aby można było wykorzystać powyższe funkcje również do klasy np. `Wektor2D`?
- Napisz klasę `Wektor2D`, tak aby zadziałał następujący kod:

```
Wektor2D w1(1, 3), w2(3, 5);
Wektor2D w3 = mymax(w1, w2);
```

(Wynik działania sprawdź pod debuggerem)

- Przerób klasę `Wector2D` tak aby był to wzorzec klasy sparametryzowany typem składowych wektora:

```
template< class T >
Wector2D
{
    . . .
};
```

- Sprawdź działanie takiej klasy dla różnych typów (`int`, `double`), np.:

```
Wektor2D<double> wt1(10., 20.), wt2(10.,40.);
Wektor2D<double> wt3 = mymax( wt1, wt2 );
```

- Dodaj do klasy `Wektor2D` `operator` << pozwalający na wykonanie następującego kodu:

```
Wektor2D<double> wt1(10., 20.), wt2(10.,40.);
cout << mymax( wt1, wt2 );
```

- Zapewnij aby jedynie wektory typu int miały niepowtarzalną postać wydruku. (Należy skonkretyzować `operator`<< dla typu `int`.)
- Przerób klasę `Wector2D` tak aby była ona również sparametryzowana liczbą składowych:

```
template< class T, int N >
Wektor
{
    . . .
};
```

- W konstruktorze domyślnym zainicjalizuj każdą ze składowych wektora kolejną liczbą naturalną. Nie używaj dynamicznej alokacji pamięci (przynamniej na początku). Sprawdź działanie takiej klasy, np:

```
WektorNT<int,20> wnt;
cout << wnt;
```