Exceptions

Mechanizm wyjątków

try throw catch

Mechanizm wyjątków to sposób na zgłaszanie błędów w momencie ich wykrycia i natychmiastową ich obsługę. W języku C++ jest to realizowane za pomocą poniższych słów kluczowych:

- **try** - wraz z blokiem instrukcji definiuje fragment kodu podejrzany o wystąpienie wyjątku
- **throw** - instrukcja służąca do zgłoszenia wyjątku
- **catch** - definiuje zachowania programu po wykryciu wyjątku

składnia obsługi wyjątków

```
try
{
    // kod w którym może zdarzyć się wyjątek
}
catch( /* oczekiwany wyjątek do obsługi */ )
{
    // procedura obsługi wyjątku
}
```

throw

```
int main() {
    int dzielnik = 0;

    if (dzielnik == 0) {
        throw (std::string)"ratunku";
    }

    return 0;
}
```

```
        return 0;
```

Exception Unhandled

Unhandled exception at 0x771018A2 in wyjatki.exe: Microsoft C++ exception: std::basic_string<char,std::char_traits<char>,std::allocator<char> > at memory location 0x008FFD9C.

catch

```cpp
int main() {
    try {
        int dzielnik = 0;

        if (dzielnik == 0) {
            throw (std::string)"ratunku";
        }
    }
    catch (std::string wyjatek) {
        std::cout << wyjatek << "\n";
        std::cout << "poradzimy sobie z tym!\n";
    }

    std::cout << "juz wszystko dobrze :)";
    return 0;
}
```

```
ratunku
poradzimy sobie z tym!
juz wszystko dobrze :)
```

```cpp
int main() {
    try {
        try {
            try {
                bool wszystkoJestOk = false;
                if (!wszystkoJestOk) {
                    throw "ojejku!";
                }
            }
            catch (...) {
                std::cout << "nie daje rady!\n";
                throw;
            }
        }
        catch (...) {
            std::cout << "uh uh!\n";
            throw;
        }
    }
    catch (...) {
        std::cout << "dobra, teraz damy rade!\n";
    }
    return 0;
}
```

zagnieżdżone try catch

Zwróćcie też uwagę na to, że nie trzeba podawać typu wyjątku. Łapiemy wtedy wszystkie!

Jeżeli w bloku catch wywołamy throw, to rzucimy złapany wcześniej wyjątek.

```
nie daje rady!
uh uh!
dobra, teraz damy rade!
```

```cpp
int main() {
    try {
        mozeRzuciInt(false);
        mozeRzuciChar(true);
        mozeRzuciBool(false);
    }
    catch (int wyjatek) {
        std::cout << wyjatek << "\n";
        std::cout << "poradzimy sobie z tym intem!\n";
    }
    catch (char wyjatek) {
        std::cout << wyjatek << "\n";
        std::cout << "poradzimy sobie z tym charem!\n";
    }
    catch (bool wyjatek) {
        std::cout << wyjatek << "\n";
        std::cout << "poradzimy sobie z tym boolem!\n";
    }

    return 0;
}
```

wielokrotne catch

Zwróćcie uwagę, że funkcje mogą wyrzucać wyjątki wyżej, do funkcji je wywołującej!

```
a
poradzimy sobie z tym charem!
```

Język C++ nie zapewnia wbudowanych typów wyjątków (np. dla dzielenia przez 0), a w przypadku nieoczekiwanego zachowania programu generuje *“undefined behaviour”*. Zadaniem programisty jest przewidzenie i stworzenie odpowiednich typów wyjątków, np: out_of_range dla wyjścia poza zakres tablicy.

```cpp
class bladPierwszejFunkcji : public std::exception {};
class bladDrugiejFunkcji : public std::exception {};
class niktNieWieCzyjToBlad : public std::exception {};

void pierwszaFunkcja(bool czyRzuci) {
    if (czyRzuci)
        throw bladPierwszejFunkcji();
}

void drugaFunkcja(bool czyRzuci) {
    if (czyRzuci)
        throw bladDrugiejFunkcji();
}

void jakisTajniak(bool czyRzuci) {
    if (czyRzuci)
        throw niktNieWieCzyjToBlad();
}

int main() {
    try {
        pierwszaFunkcja(false);
        drugaFunkcja(true);
        jakisTajniak(false);
    }
    catch (bladPierwszejFunkcji) {
        std::cout << "uwazajcie na pierwsza funkcje!";
    }
    catch (bladDrugiejFunkcji) {
        std::cout << "uwazajcie na druga funkcje!";
    }
    catch (std::exception) {
        std::cout << "to jeszcz e ktos inny zawinil!";
    }

    return 0;
}
```

std::exception i polimorfizm

Czy jeżeli Tajniak popełni błąd, to zostanie on złapany?

Zwróćcie uwagę, że klasę, po której dziedziczymy, łapiemy na samym końcu. Inaczej to zawsze ten blok catch łapał by wyjątek!

```
uwazajcie na druga funkcje!
```

```cpp
class bladNiezawiazanychSznurowek : public std::exception {
public:
    const char * what() noexcept {
        return "Nastepnym razem zawiaz buty!";
    }
};

void ktosTuSieWywroci() {
    throw bladNiezawiazanychSznurowek();
}

int main() {
    try {
        ktosTuSieWywroci();
    }
    catch (bladNiezawiazanychSznurowek &e) {
        std::cout << e.what();
    }

    return 0;
}
```

what()

Dzieci std::exception mogą implementować wirtualną metodę what(), która zwraca komunikat o rzuconym wyjątku.

noexcept przy sygnaturze what() oznacza, że funkcja nie może rzucić żadnego wyjątku.

```
Nastepnym razem zawiaz buty!
```

Wyjątki rzucane przez składowe biblioteki standardowej (std::):

- bad_alloc - wyrzucany przez new w przypadku błędu alokacji,
- bad_cast - wyrzucany przez dynamic_cast w przypadku błędu przy rzutowaniu referencji,
- bad_exception - wyrzucany, gdy nie zostanie znaleziony odpowiadający blok catch,
- bad_typeid - wyrzucany przez typeid na przykład przy podaniu nullptr jako argumentu,
- logic_error - rodzic wyjątków dla błędów wewnętrznej logiki programu.

Wymagane #include <exception>.