Konwersje jawne

- Podczas wykorzystywania polimorfizmu możemy natrafić na pewien problem
  - Posługując się obiektem za pomocą uogólnienia nie znamy jego rzeczywistego typu
  - W niektórych sytuacjach potrzebna nam jest dokładana informacja o faktycznym typie obiektu
    - Kiedy?
- W C++ możemy jednak odzyskać informację o typie obiektu za pomocą mechanizmu *RTTI* (*run-time type information*)
  - Spróbować przekształcić obiekt na jego rzeczywisty typ korzystając z rzutowania
  - Ustalić klasę obiektu

Rzutowanie `dynamic_cast`

- Posługując się operatorem rzutowania `dynamic_cast` możemy przywrócić obiektowi jego rzeczywisty typ
  - `dynamic_cast<Klasa&>(obiekt);`
  - Typ do którego przekształcamy obiekt musi być referencją lub wskaźnikiem
- Przy rzutowaniu możemy bezpośrednio odwołać się do składnika klasy do której rzutujemy
  - `dynamic_cast<Klasa&>(obiekt).fun();`
  - Jeżeli rzutowanie nie będzie możliwe to zostanie wyrzucony wyjątek (`std::bad_cast`)
- Z rzutowaniem `dynamic_cast` związany jest dodatkowy narzut czasowy w stosunku do `static_cast`
- Przykład cpp_7.8

Zapytanie o typ

- Za pomocą operatora **`typeid`** możemy ustalić typ obiektu podczas wykonania programu
  - Musimy dołączyć nagłówek **`typeinfo`**
  - **`cout << typeid(obj).name();`**
  - **`if(typeid(obj) == typeid(Klasa))`**
- Operator **`typeid`** zwraca obiekt **`std::type_info`**, dla którego zdefiniowane są między innymi
  - Funkcja **`name()`** - zwraca nazwę klasy (dokładna postać zależy od implementacji)
  - Operatory **`==`** i **`!=`** pozwalające na porównanie dwóch typów
- Przykład cpp_7.9

Mechanizm *RTTI* i projektowanie klas

- Mechanizm *RTTI* należy używać z rozwagą (w szczególności **`typeid`**)
  - Na pewno nie należy używać instrukcji warunkowych, w których **`typeid`** służy do określania jaką funkcję wywołać
- Używanie tego mechanizmu sprzyja tworzeniu nieprawidłowo zaprojektowanego kodu, dlatego należy go unikać
- Jeżeli tylko jest możliwe to należy używać funkcji wirtualnych i projektować klasy tak, żeby możliwe było wywołanie odpowiedniej funkcji składowej
- W szczególnym przypadku możemy posłużyć się rzutowaniem **`dynamic_cast`**
- Przykład cpp_7.10 (podobnie jak cpp_7.8)

Dziedziczenie wielokrotne

- Klasa może dziedziczyć po więcej niż jednej klasie bazowej, wtedy mówimy o dziedziczeniu wielokrotnym
  - Nie jest to cecha dostępna we wszystkich językach programowania zorientowanego obiektowo
- Tego typu dziedziczenie jest rzadziej stosowane
  - Zaletą jego jest możliwość połączenia ze sobą zupełnie niezależnych od siebie klas
  - `class C : public A, public B {...}`
- Przykład cpp_7.11

Aspekty dziedziczenia wielokrotnego

- Dana klasa podstawowa może znaleźć się na liście dziedziczenia tylko raz
- Definicja klasy umieszczona na liście pochodzenia musi być wcześniej znana, nie wystarcza deklaracja zapowiadająca typu `class A;`
- Na liście pochodzenia przed nazwami klas pojawiają się określenia sposobu dziedziczenia
  - Domniemanie jest identyczne jak w dziedziczeniu jednokrotnym, czyli `private`

Konstruktory przy dziedziczeniu wielokrotnym

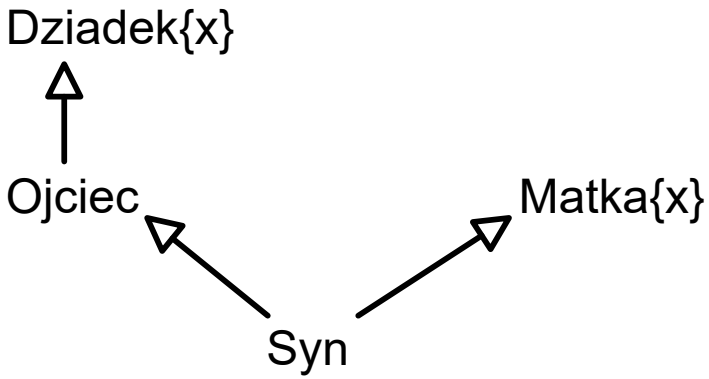
- Konstruktor klasy pochodnej na liście inicjalizacyjnej może zawierać wywołania konstruktorów swoich bezpośrednich klas podstawowych
  - ```
    class C : public A, public B {
              C() : B(), A() {...}
    }
    ```
- Kolejność wywoływania konstruktorów uzależniona jest od kolejności umieszczenia nazw klas podstawowych na liście pochodzenia
- Przykład cpp_7.12
    ```

Wieloznaczność przy dziedziczeniu wielokrotnym

- Z wieloznacznością mamy do czynienia w sytuacji gdy wyrażenie odnoszące się do składnika klasy podstawowej równie dobrze może odnosić się do innego składnika drugiej klasy podstawowej
- `A{x,...}, B{x,...}`
  `class C : public A, public B`
- Aby odniesienie do składnika było możliwe musimy użyć operatora zakresu `::`
  - `A::x;` lub `B::x;`
- Należy pamiętać, że najpierw sprawdzana jest jednoznaczność, a dopiero później dostęp!!!
  - Czyli jeśli nawet składnik jednej klasy jest prywatny to mimo to kompilator zaprotestuje
- Przykład cpp_7.13

Wieloznaczność, a pokrewieństwo

- Bliższe pokrewieństwo usuwa wieloznaczność
- Wieloznaczności nie ma bo droga do nazwy **x** w klasie podstawowej **Matka** jest znacząco krótsza niż do identycznego składnika w klasie **Dziadek**



Wieloznaczność, a pokrewieństwo...

- Jeżeli mimo wszystko musimy posłużyć się kwalifikatorem zakresu to nie musimy podawać dokładnego określenia, w której klasie interesujący nas obiekt się znajduje
- Wystarczy podać zakres od którego zaczną się poszukiwania
  - `A::x == C::x`
  - `D::x == E::x`
  - `F::x == G::x`