# C++: Funkcje wirtualne

## Wskaźnik (referencja) do klasy podstawowej, a obiekty klasy pochodnej

- Wskaźnik do pokazywania na obiekty klasy podstawowej może być wykorzystany do pokazywania na obiekty klasy pochodnej
- Referencja do nazywania obiektów klasy podstawowej może być wykorzystana do nazywania obiektów klasy pochodnej

- Przy takim postępowaniu obiekt klasy pochodnej "udaje" obiekt klasy podstawowej
- Odwrotne postępowanie nie jest dopuszczalne, bo obiekt klasy podstawowej nie ma jak "udawać" obiektu klasy pochodnej

```
struct A{
  int i;
}
struct B : public A{
  int j;
}
A a;
B b;
A* ptrA=&b; //  OK
A& refA=b;  //  OK
B* ptrB=&a; // Błąd, bo ptrB->j nie ma sensu
B& refB=a;  // Błąd, bo refB.j nie ma sensu
```

## Wywołanie funkcji składowych na rzecz obiektu klasy pochodnej wskazywanego przez referencję do klasy podstawowej

Do klas **samochod** i **taksowka** dodajmy klasę

```
class autostrada_platna{
   const float dystans;
   public:
   autostrada_platna(float
jakisDystans):dystans(jakisDystans){
   };
   void przepusc(samochod &jakis_samochod){
      jakis_samochod.jedz(dystans);
   };
};
```

Co wskaże taksometr taksówki po wykonaniu takiego kodu?

```
...
autostrada_platna krakow_katowice(78);
taksowka kwd1214=taksowka("Polonez",80800,23.5,0.0);
krakow_katowice.przepusc(kwd1214);
...
```

### Mechanizm wirtualności

Jeśli chcemy, by przy wywoływaniu funkcji składowej o doborze jej wersji decydował nie typ referencji wskazującej na obiekt ale typ obiektu, na który referencja wskazuje, poprzedzamy ją słowem **virtual**.

```
virtual void jedz(float ile_kilometrow);
```

Podstawowym pożytkiem z wirtualności jest możliwość tworzenia uniwersalnego kodu, który działa poprawnie bez żadnych modyfikacji na klasach pochodnych napisanych kiedykolwiek później

Za wirtualność się płaci:

- jeśli w klasie jest choć jedna funkcja wirtualna, to obiekt tej klasy jest nieco większy
- podejmowanie decyzji o tym, którą funkcję w danym przypadku wykonać trwa pewien czas

- Słowo **virtual** pojawia się tylko przy deklaracji funkcji wewnątrz ciała klasy.
- Jeśli definicja funkcji składowej jest poza ciałem klasy, to przy definicji funkcji nie powtarza się już słowa **virtual**.

```
class X{
  public:
  // słowo virtual stawiamy przy deklaracji
  void virtual jakas_funkcja(argumenty);
  ....
};
......
// nie stawiamy go przy definicji
void X::jakas_funkcja(argumenty){
  ....
};
```

- Funkcja składowa jest wirtualna wtedy, gdy w definicji klasy przy jej

deklaracji stoi słowo **virtual**, lub gdy w jednej z klas podstawowych tej klasy funkcja o identycznym nagłówku zadeklarowana jest jako **virtual**. O identycznym nagłówku - to znaczy z identyczną nazwą, typem i kolejnością argumentów i typem rezultatu.

- Słowo **virtual** może wystąpić tylko raz w klasie podstawowej i nie musi już powtarzać się przy analogicznych funkcjach w klasach pochodnych. (Chociaż może).
- Wszystkie funkcje o takim nagłówku w następnych pokoleniach (nawet tych jeszcze nie wymyślonych) będą automatycznie wirtualne.
- Wirtualną może być tylko funkcja składowa. Funkcja globalna nie może być wirtualna.
- Funkcja wirtualna nie może być funkcją składową typu **static**.

### Stałe funkcje składowe, a wirtualność

- Stałe funkcje składowe to formalnie inne funkcje niż ich niestałe odpowiedniki
- W klasie mogą być obecne równocześnie dwie funkcje o tej samej nazwie, tej samej liście argumentów, tym samym zwracanym typie, różniące się jedynie tym, że jedna jest określona jako stała, a druga nie.
- Mechanizm wirtualności działa dla takich funkcji oddzielnie

### Wczesne i późne wiązanie

- Jeżeli decyzja, którą funkcję składową wywołać może odbyć się w czasie kompilacji, nazywamy to wiązaniem w czasie kompilacji lub krócej: **wczesnym wiązaniem**.
- Jeśli w trakcie kompilacji nie da się rozstrzygnąć, którą wersję funkcji wirtualnej wywołać, kompilator wygeneruje taki kod, który decyzję o powiązaniu wywołania funkcji z określoną wersją funkcji wirtualnej będzie podejmował dopiero na etapie wykonywania programu.
- Mówimy, że jest to wiązanie na etapie wykonania lub **późne wiązanie**.

```
samochod kr52222;
taksowka kn32432;
// Wczesne wiązanie
```

```
kr52222.jedz(100); // Wiadomo, że chodzi o funkcję z klasy
samochod
samochod* s=&kn32432;
// Późne wiązanie
s->jedz(30);       // Pod wskaznikiem s może być tak samochód
jak i taksówka
```

### Automatyczne wczesne wiązanie dla funkcji wirtualnej

Nie każde wywołanie funkcji wirtualnej oznacza późne wiązanie. Są sytuacje gdzie kompilator może sobie uprościć sprawę i zastosować zwykłe, wczesne wiązanie.

- wywołanie na rzecz obiektu

```
samochod kvj2020=samochod("Corolla",1200,20.7);
samochod* s=new taksowka("Corolla",1200,20.7);
kvj2020.jedz(100.5);  // wczesne wiązanie
s->jedz(23.7);         // późne wiązanie
```

- jawne użycie kwalifikatora zakresu

```
s->samochod::jedz(23.7);          // wczesne wiązanie
```

- wywołanie z konstruktora lub destruktora klasy podstawowej - to dlatego, że konstruktor klasy podstawowej pracuje wówczas, gdy obiekt klasy pochodnej nie jest jeszcze w całości skonstruowany

Przy wywołaniu funkcji wirtualnej, kwalifikator zakresu stosujemy tylko wtedy, gdy chodzi nam o sięgnięcie do składników klasy podstawowej - z funkcji składowych klasy pochodnej.

### Funkcje wirtualne, a mimo to `inline`

- W sytuacjach, gdy chodzi nam o rzeczywisty polimorfizm (późne wiązanie) przydomek **`inline`** będzie zignorowany.
- W sytuacjach, gdy już na etapie kompilacji wiadomo dla jakiego obiektu następuje wywołanie funkcji, implementacja będzie **`inline`**

### Polimorfizm

Możliwość zróżnicowania zachowania obiektów klas pochodnych wskazywanych wskaźnikiem klasy macierzystej określa się mianem **polimorfizm**.

Polimorfizm (wielość form) zostaje urzeczywistniony tam, gdzie zachodzi późne wiązanie funkcji wirtualnej.

Polimorfizm ujawnia się tylko przy wywołaniach za pomocą wskaźnika lub referencji.

### Wirtualny destruktor

- Wirtualny destruktor jest dopuszczalny, choć to wyjątek, bo nie spełnia warunku, że nazwa jest ta sama.
- Jednak destruktor w danej klasie może być tylko jeden, więc nie ma niejednoznaczności.

Jeśli klasa deklaruje jedną ze swych funkcji jako `virtual`, to destruktor tej klasy też powinien być `virtual`.

### Konstruktor nie może być wirtualny

- Wirtualny konstruktor nie jest możliwy.
- To dlatego, że konstruktora nie da się wywołać poprzez wskaźnik lub referencję, gdyż w momencie wołania konstruktora obiektu jeszcze nie ma.
- Mimo to można kreować obiekty wirtualnie na wzór innego obiektu bez wirtualnych konstruktorów.