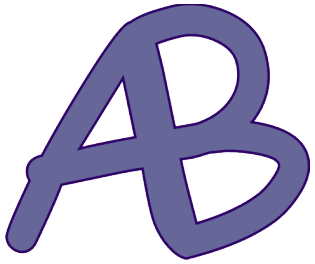

Wykład 9: Metody wirtualne i polimorfizm

Funkcje wirtualne

Funkcje wirtualne

Funkcje wirtualne to funkcje składowe, które przydają się szczególnie, gdy używamy obiektów posługując się wskaźnikami lub referencjami do nich.

Dla zwykłych funkcji z identycznymi nazwami to, czy zostanie wywołana funkcja z klasy podstawowej, czy pochodnej, zależy od typu wskaźnika, a nie tego, na co faktycznie on wskazuje.

```
1  class bazowa
2  {
3  public:
4      virtual void metoda_wirtaulan()
5      {
6          // treść metody
7      }
8  };
```

Klasy absreakcyjne

Funkcje wirtualne to funkcje składowe, które przydają się szczególnie, gdy używamy obiektów posługując się wskaźnikami lub referencjami do nich.

Dla zwykłych funkcji z identycznymi nazwami to, czy zostanie wywołana funkcja z klasy podstawowej, czy pochodnej, zależy od typu wskaźnika, a nie tego, na co faktycznie on wskazuje.

```
1  class bazowa
2  {
3  public:
4      virtual void metoda_wirtaulan()
5      {
6          // treść metody
7      }
8  };
```

Funkcje wirtualne

Dysponując funkcjami wirtualnymi będziemy mogli użyć polimorfizmu - używać metod klasy pochodnej wszędzie tam, gdzie spodziewana jest klasa podstawowa.

W ten sposób będziemy mogli korzystać z metod klasy pochodnej korzystając ze wskaźnika, którego typ odnosi się do klasy podstawowej. W tej chwili może się to wydawać niepraktyczne, lecz za chwilę przekonasz się, że funkcje wirtualne niosą naprawdę sporo nowych możliwości.

Funkcje wirtualne

Bez użycia metod wirtualnych:

Mając klasę bazową wyprowadzamy od niej klasę pochodną:

```
1   class Baza
2   {
3   public:
4       void pisz()
5       {
6           std::cout << "Tu funkcja pisz z klasy Baza" << std::endl;
7       }
8   };
9
10  class Baza2 : public Baza
11  {
12  public:
13      void pisz()
14      {
15          std::cout << "Tu funkcja pisz z klasy Baza2" << std::endl;
16      }
17  };
```

Źródło: https://pl.wikibooks.org

Funkcje wirtualne

Jeżeli teraz w funkcji main stworzymy wskaźnik do obiektu typu Baza, to możemy ten wskaźnik ustawiać na dowolne obiekty tego typu. Można też ustawić go na obiekt typu pochodnego, czyli Baza2:

```
19  int main()
20  {
21      Baza   *wsk;
22      Baza    objB;
23      Baza2   objB2;
24
25      wsk = &objB;
26      wsk -> pisz();
27
28      wsk = &objB2;
29      wsk -> pisz();
30      return 0;
31  }
```

Źródło: https://pl.wikibooks.org

Funkcje wirtualne

Jeżeli teraz w funkcji main stworzymy wskaźnik do obiektu typu Baza, to możemy ten wskaźnik ustawiać na dowolne obiekty tego typu. Można też ustawić go na obiekt typu pochodnego, czyli Baza2:

```
19  int main()
20  {
21      Baza   *wsk;
22      Baza    objB;
23      Baza2   objB2;
24
25      wsk = &objB;
26      wsk -> pisz();
27
28      wsk = &objB2;
29      wsk -> pisz();
30      return 0;
31  }
```

Źródło: https://pl.wikibooks.org

Funkcje wirtualne

Po skompilowaniu na ekranie zobaczymy dwa wypisy: „Tu funkcja pisz z klasy Baza”. Stało się tak dlatego, że wskaźnik jest do typu Baza. Gdy ustawiliśmy wskaźnik na obiekt typu pochodnego a następnie wywołaliśmy funkcję składową, to kompilator sięgnął po funkcję pisz z klasy bazowej.

Można jednak określić żeby kompilator nie sięgał po funkcję z klasy bazowej, ale sam się zorientował na co wskaźnik pokazuje. Do tego służy przydomek virtual, a funkcja składowa nim oznaczona nazywa się wirtualną. Różnica polega tylko na dodaniu słowa kluczowego virtual:

Funkcje wirtualne

```
1   class Baza
2   {
3   public:
4       virtual void pisz()
5       {
6           std::cout << "Tu funkcja pisz z klasy baza" << std::endl;
7       }
8   };
9
10  class Baza2 : public Baza
11  {
12  public:
13      virtual void pisz()
14      {
15          std::cout << "Tu funkcja pisz z klasy Baza2" << std::endl;
16      }
17  };
```

Źródło: https://pl.wikibooks.org

Funkcje wirtualne

Funkcja wirtualna nie może być funkcją składową typu static. Przydomek static mówił, że funkcja jest wywoływana nie na rzecz jakiegoś konkretnego obiektu, ale na rzecz całej klasy. Mechanizm wywoływania funkcji wirtualnej jest jednak taki, że o tym, którą wersję funkcji wywołać, decyduje się na podstawie obiektu. Musi być więc wywołanie na rzecz obiektu (znanego z przezwiska, lub pokazywanego wskaźnikiem) - ale obiektu.

Funkcje wirtualne

Gdy funkcja jest oznaczona jako wirtualna, kompilator nie przypisuje na stałe wywołania funkcji z tej klasy, na którą pokazuje wskaźnik, już podczas kompilacji. Pozostawia decyzję co do wyboru właściwej wersji funkcji aż do momentu wykonania programu - jest to tzw. późne wiązanie.

Program skorzysta z informacji zapisanej w obiekcie a określającej klasę, do jakiej należy dany obiekt. Dopiero po odczytaniu informacji o klasie danego obiektu wybierana jest właściwa metoda.

Źródło: https://pl.wikibooks.org

Funkcje wirtualne

Wirtualny destruktor

Niewirtualny destruktor może powodować wycieki pamięci i niezdefiniowane zachowanie programu, ponieważ będzie wywoływany ten, który odpowiada typowi wskaźnika, a nie ten, który odpowiada rzeczywistemu typowi obiektu.

```
1   class Klasa
2   {
3   public:
4       virtual ~Klasa();
5   };
```

Klasy abstrakcyjne

Klasy abstrakcyjne

Klasy abstrakcyjne to klasy dla których nie można stworzyć obiektu – stworzone tylko po to, aby po nich dziedziczyć.

Często zdarza się że mamy kilka klas które mają pewną ilość cech wspólnych, aczkolwiek między nimi samymi nie zachodzi relacja dziedziczenia (żadna z klas nie jest szczególnym przypadkiem innej klasy) – można wydzielić pola wspólne w abstrakcyjną klasę bazową.

Klasy abstrakcyjne

Najważniejszą zaletą klas abstrakcyjnych jest możliwość wywołań polimorficznych.

Poza tym:

- jeśli postanowimy zmienić jakieś pole wspólne, to zmiany dokonujemy tylko w klasie bazowej,
- jeśli pojawią się nowe wspólne cechy które potrzebujemy to dodajemy je tylko w jednej klasie

Klasy abstrakcyjne

Aby klasa była abstrakcyjna to musi mieć przynajmniej jedną metodę czysto wirtualną - czyli metodę wirtualną która nie ma ciała.

```
1   class A
2   {
3     public:
4       virtual void metodaCzystoWirtualna() =0;
5   };
```

Literatura:

W prezentacji wykorzystano przykłady i fragmenty:

- Gręboszj. : ***Symfonia C++, Programowanie w języku C++ orientowane obiektowo***, Wydawnictwo Edition 2000.
- Jakubczyk K.: *Turbo Pascal i Borland C++ Przykłady*, Helion.

Warto zajrzeć także do:

- Sokół R. : ***Microsoft Visual Studio 2012 Programowanie w Ci C++,*** Helion.
- Kerninghan B. W., Ritchie D. M.: ***język ANSI C***, Wydawnictwo Naukowo Techniczne.

Dla bardziej zaawansowanych:

- Gręboszj. : ***Pasja C++,*** Wydawnictwo Edition 2000.
- Meyers S.: ***język C++ bardziej efektywnie***, Wydawnictwo Naukowo Techniczne