Przeładowywanie operatorów, dziedziczenie

Wykład 6

Przeładowywane operatorów

- Jest bardzo wygodną metodą, zamiast definiować funkcje typu `add` itp. możemy użyć odpowiednich operatorów
- Przeładowanie operatora dokonuje się definiując własną funkcję o nazwie `operatorX`, gdzie `X` oznacza symbol interesującego nas operatora
  - Może być funkcją składową
  - Może być globalną funkcją – wyjątki!!!
- Lista operatorów, które można przeładowywać
  - `+, -, *, /, %, ^, &, |, ~, ! , =, <, >, +=, -=, *=, /=, %=, ^=, &=, |=, <<, >>, >>=, <<=, ==, !=, <=, >=, &&, ||, ++, --, ‘,’ , ->*, ->, new, delete, () , []`
  - Tylko jako metody: `=`, `()`, `[]`, `->`
- Natomiast następujące operatory nie mogą być przeładowywane
  - `.`, `.*`, `::`, `?:`

Przeładowywane operatorów...

- Nie można wymyślać swoich operatorów
  - Np. **`**`**
- Nie można zmieniać priorytetów operatorów
- Nie można też zmienić argumentowości operatorów, czyli tego czy są jedno- czy dwuargumentowe
- Nie można również zmieniać łączności operatorów
- Dla każdej klasy następujące operatory są generowane automatycznie
  - **`=`**, **`&`**(jednoargumentowy – pobranie adresu), **`new`**, **`,`** , **`delete`**

Funkcja operatorowa jako funkcja składowa

- Jeżeli definiujemy składową funkcje operatorową to przyjmuje ona zawsze o jeden mniej argument niż ta sam funkcja napisana w postaci funkcji globalnej
- Funkcja ta nie może być typu `static`, bo w jej działaniu bierze udział wskaźnik `this`
- Nie mogą istnieć dwie funkcje operatorowe pracujące na tych samych argumentach (zdefiniowane jako funkcja globalna i funkcja składowa)
- W tym wypadku po lewej stronie operatora zawsze musi stać obiekt klasy, dla której ten operator jest zdefiniowany
  - `Obiekt1 + Obiekt2;`
  - `Obiekt1.operator+(Obiekt2);`
- Przykład cpp_6.1

Funkcja operatorowa jako funkcja globalna

- Nie musi być funkcją zaprzyjaźnioną
- Jeżeli wymaga dostępu do zmiennych prywatnych to musi być zaprzyjaźniona
- Dzięki globalnym funkcjom operatorowym można zdefiniować operatory do klas już istniejących np. bibliotecznych
  - W przypadku takich klas muszą one udostępniać odpowiedni interfejs (w szczególności dostęp do składowych)
- Nie ma takiego ograniczenia jak dla funkcji operatorowej zdefiniowanej jako metoda
- Przykład cpp_6.2

Przemienność

- Funkcja operatorowa będąca funkcją składową klasy wymaga, aby obiekt stojący po lewej stronie operatora był obiektem tej klasy
  - `Fraction Fraction::operator*(int i);`
  - `aFraction = bFraction * 2; //OK`
  - `aFraction = 2 * bFraction; //Błąd`
- Zwykła funkcja globalna nie ma tego ograniczenia
  - `Fraction operator*(int i, Fraction K);`
    `//Argumenty mogą być w odwrotnej kolejności`
  - Oczywiście przy wywołaniach z niezgodnością typów muszą być zdefiniowane odpowiednie konwersje

Operatory, które muszą być funkcjami składowymi

- Operator przypisania `=`
  - Generowany automatyczne przez kompilator, tak że przepisuje obiekt składnik po składniku
    - Nie zawsze dobry – wskaźniki
  - Nie jest generowany automatycznie w sytuacjach
    - Jeżeli klasa ma składnik `const`
    - Jeżeli klasa ma składnik będący referencją
    - Jeżeli klasa ma składową klasę, w której operator przypisania jest prywatny
    - Jeżeli klasa ma klasę podstawową z prywatnym operatorem przypisania
  - Nie jest dziedziczony
  - Na ogół zawiera
    - Cześć destruktorową
    - Cześć konstruktorową
- Przykład cpp_6.3
- Test cpp_6.01

Nowy operator=

- Jest to funkcja składowa niestatyczna i nieszablonowa o nazwie operator=
  - `class_name & class_name :: operator= ( class_name && )`
  - Funkcja wywoływana jest kiedy pojawia się po lewej stronie =, a po jego prawe stoi rvalue
  - „Kradnie" zasoby obiektu stojącego po prawej stronie
    - np. dla std::string zostawia po prawej stronie obiekt pusty
- Generowana automatycznie w sytuacji kiedy
  - Nie ma konstruktora kopiującego (niedomyślnego)
  - Nie ma konstruktora przenoszalnego (niedomyślnego)
  - Nie ma kopiującego operatora=
  - Nie ma destruktora
  - Generowany jest wtedy publiczny i inline `T& T::operator=(T&&)`
- Jeżeli jest „trywialny" wykorzystuje do przenoszenia `std::memmove`
  - Trywialny znaczy
    - Generowany automatycznie
    - T nie posiada funkcji wirtualnych i wirtualnych klas bazowych
    - Trywialny jest przenaszalny operator= dla klas bazowych oraz składników
- Przykład 6.3a1

Operatory które muszą być funkcjami składowymi...

- Operator `[]`
  - Przeładowany operator `[]` powinien mieć działanie podobne do działania w stosunku do typów wbudowanych
    - Z tego powodu powinien być zadeklarowany
      `klasa& klasa::operator[](unsigned i);`
      czyli zwracać referencję do pojedynczego elementu tablicy o indeksie `i`
    - Możliwe będzie wtedy wykonanie
      - `a = tab[i];`
      - `tab[i] = a;`
- Przykład cpp_6.3a

Operatory które muszą być funkcjami składowymi...

- Operator `()`
  - Może przyjmować dowolną liczbę parametrów
  - Może posłużyć do indeksowanie wielowymiarowych tablic
    - `tab(1,2,3);`
  - Może też upraszczać zapis, nie musimy wywoływać funkcji tylko wystarczy sam operator `()`
  - Bardzo przydatny operator przy wykorzystaniu funktorów z algorytmami STL
  - Przykład cpp_6.3b i cpp_6.3c
- Operator `->`
  - Rzadko używany
  - Przydaje się gdy piszemy klasę, której obiekty pełnią rolę podobną do wskaźników
  - Wykorzystany między innymi przy tworzeniu klasy `unique_ptr` z STL-a
  - Zrobić przykład samodzielnie !!!

Operator `new` i `delete`

- W stosunku do klas funkcje przeładowujące te operatory są zawsze typu `static`, nawet jeśli tego nie zadeklarujemy
  - Przydają się kiedy chcemy uzyskać jakąś dodatkową funkcjonalność np. statystykę
  - Tworzymy obiekty w predefiniowanej wcześniej pamięci
  - Używamy niestandardowej biblioteki do tworzenia nowych obiektów
- Istnieją również globalne wersje tych operatorów
  - `void* operator new(size_t sz)`
  - `void operator delete(void* m)`
- Przykład cpp_6.3d, cpp_6.3e i cpp_6.3f

Operatory pre i post ++ --

- Operatory preinkrementacji ++ i -- działają jak zwykłe inne operatory jednoargumentowe
- Problem z operatorami postinkrementacji ++ i --, których w normalny sposób nie da się przeładować
  - Rozwiązano ten problem deklarując te operatory jak dwuargumentowe
  - `Point Point::operator++(int)`
  - Tworzony jest obiekt tymczasowy, o czym należy pamiętać przy optymalizacji
- Przykład cpp_6.4

Operatory << i >>

- Przy przeładowywaniu tych operatorów w stosunku do klasy **`iostream`** możemy je zdefiniować tylko jako globalne funkcje
  - Precyzyjniej w stosunku do klas **`istream`** oraz **`ostream`**
    - Będzie działać wtedy na standardowy WE/WY oraz z plikami
- Funkcja operatorowa musi pracować na zmiennych lub metodach globalnych
- Ewentualnie musi być zaprzyjaźniona z naszą klasą, jeżeli ma pracować na zmiennych prywatnych
- Przykład cpp_6.4