Funkcje „specjalizowane”

- Czasami funkcja wygenerowana przez szablon może być nieodpowiednia
  - np. `max(char*, char*)`
- Istnieje wtedy możliwość zdefiniowanie normalnej funkcji, która będzie pracować w odpowiedni sposób na takich danych
- W takiej sytuacji kompilator wykorzysta tą specjalizowaną wersję funkcji, dopiero jeżeli takowej nie znajdzie to skorzysta z szablonu
  - Dopasowanie musi być dokładne tzn. `char* != const char*`
- Przykład cpp_9.6

Dopasowywanie argumentów

- Dopasowanie dokładne
  - Kompilator szuka funkcji o odpowiedniej nazwie z dokładnie takimi samymi argumentami
- Poszukiwanie szablonu, z którego można wyprodukować funkcję o argumentach takiego samego typu jak wywołanie
  - Dopasowanie wszystkich argumentów musi być idealne (bez konwersji standardowych)
- Kontynuacja poszukiwania wśród funkcji (nie szablonów)
  - Konwersje standardowe
  - Konwersje zdefiniowane przez programistę

Jeden szablon w wielu plikach

- Ponieważ szablony na ogół umieszczamy w plikach nagłówkowych to może się zdarzyć, że powstaną w osobnych modułach programu takie same funkcje
  - Taka sytuacja nastąpi, jeżeli w jednym pliku powstanie funkcja np. `int max(int, int)` w wyniku jej wywołania i w innym też
  - Wtedy aby program został poprawnie skonsolidowany („zlinkowany") to linker musi być „inteligentny" tzn. usunąć nadmiarowe definicję takich samych funkcji
- Przykład cpp_9.7 i cpp_9.8

Częściowa „specjalizacja"

- Faktycznie w przypadku funkcji nie jest to częściowa specjalizacja
- Cały czas mamy do czynienia z przeładowaniem nazw
  - Ciągle te same reguły
    - Jakie?
- `template<class T> void f(T a);`
- `template<class T> void f(T* a);`
- `template<class T> void f(const T* a);`
- Przykład cpp_9.8a

C++11 i C++14 auto i decltype, a zwracany typ w szablonach funkcji

- W przypadku szablonów rozwiązuje problem wyznaczenia typu zwracanego
  - Typ deklarowany jak **`auto`**
  - Z informacją dla kompilatora w jaki sposób typ zwracany ma zostać wyznaczony
    - Wymagane w przypadku C++11, opcjonalne w przypadku C++14 (działa automatyczna dedukcja typów)
  - ```
    template<typename T, typename U>
    auto add(T t, U u) -> decltype(t + u)
    { return t + u; }
    ```
- Przykład cpp_9.8b
    ```

Szablony funkcji uwagi

- Szablon funkcji nie powinien pracować na zmiennych globalnych
- Dwa (lub więcej) szablony o takiej samej nazwie mogą istnieć - jest to po prostu przeładowanie nazw
  - Nie powinny generować funkcji o takich samych argumentach
- Możemy tworzyć funkcję z szablonu i od razu deklarować jakiego typu ma ona być (kompilator nie będzie wtedy decydował na podstawie parametrów wywołania)
  - `a = max<int>(a, b);`
  - `swap<double>(f, g);`

Szablony klas

- Podobnie jak szablony funkcji w języku C++ mamy możliwość definiowania szablonów klas
- Szablon klasy to nic innego jak narzędzie do automatycznego pisania różnych wersji bardzo podobnych klas
  - Szablon klasy to nie sama klasa, ale przepis jak taką klasę stworzyć
- Definiowanie szablonu klasy
  - `template<typename T> class Box {/*…*/};`

Definiowanie szablonu klasy

- Nazwa szablonu klasy musi być unikatowa
  - Nie może być taka jak nazwa innej klasy, szablonu, funkcji typu wyliczeniowego …
  - Nie istnieje przeładowanie klas
- Szablony mogą być definiowane tylko w zakresie globalnym (oczywiście mogą się znajdować w przestrzeniach nazw)
  - Nie można szablonów klas zagnieżdżać
- Klasy szablonowe powstałe z jednego szablonu nie mają nic wspólnego ze sobą (np. dziedziczenie czy przyjaźń)

Parametry szablonu klasy

- W szablonach funkcji kompilator mógł na podstawie argumentów wywołania określić jaką wersję funkcji wygenerować
- Parametry szablonu klasy muszą być podane przy tworzeniu obiektów danego typu klasy
  - Typ parametru(-ów) szablonu klasy jest jakby częścią jego nazwy, ponieważ klasy nie mogą być przeładowane
  - Parametry szablonu umieszcza się w nawiasach <>
    - Np. `box<int> a;`
- Przykład cpp_9.9

Parametry szablonu klasy…

- Parametrów szablonu klas może być więcej niż jeden
  - Parametry umieszczamy na liście (podobnie jak dla funkcji)
    - Np. `template<typename T1, typename T2> class Box{…};`
- Parametrami szablonu klas mogą być
  - Typ
  - Stałe wyrażenia
    - Stała dosłowna typu całkowitego, adresy (obiektu globalnego, funkcji globalnej, składnika statycznego klasy)

Parametry szablonu klasy…

- Parametrem aktualnym szablonu klas nie może być
  - Stała dosłowna będąca łańcuchem
  - Adres elementu tablicy
  - Adres zwykłego niestatycznego składnika klasy
- Jeżeli dwa wyrażenia będące parametrem aktualnym szablonu, mają taką samą wartość to, uznawane są za identyczne
- Przykład cpp_9.10

Funkcje składowe szablonu klas

- Definiowanie funkcji w ciele szablonu
- Definiowanie na zewnątrz szablonu klasy
  - Taką funkcję składową definiujemy w podobny sposób do szablonu funkcji
  - `template<typename T> typ_zwaracany nazwa_sz_klasy<T>::nazwa_funkcji(args) {…}`
  - `<T>` - używane jest do rozróżnienia między rożnymi funkcjami składowymi dla różnych wersji szablonu klasy
- Przykład cpp_9.11

Kiedy produkowane są klasy z szablonu

- Oczywiście jeśli definiujemy obiekt klasy
  - `box<int> a;`
- Również przy definiowaniu wskaźnika mogącego pokazywać na obiekt klasy szablonowej
  - `box<int> *a;`
  - Jest to potrzebne chociażby w sytuacji kiedy wielkość obiektu ma znaczenie
- Podobnie przy deklaracji funkcji, która jako argument przyjmuje klasę szablonową
  - `void fun(int a, box<int> b);`
- Jeżeli klasa szablonowa używana jest jako klasa podstawowa
  - `class boxA: public box<int> {…};`

Szablon funkcji z argumentem będącym szablonem klasy

- Dlaczego takiej funkcji nie zrobić w postaci funkcji składowej?
  - Nie wszystkie funkcje mogą być funkcjami składowym np. funkcje operatorowe takie jak <<
- W takiej sytuacji definiujemy sobie szablon funkcji, który jako argument przyjmuje obiekt klasy, ale powstały na podstawie typ szablonu funkcji
  - ```
    template<typename T>
    ostream& operator<<(ostream &o, klasa<T>& K);
    ```
- Przykład cpp_9.12
    ```

Obiekt klasy szablonowej będący składnikiem innej klasy szablonowej

- Podobna sytuacja do poprzedniej, parametr tym razem szablonu klasy zostanie wykorzystany do stworzenia odpowiedniego składnika klasy
  - ```
    template<typename T> class K1
    {...};

    template<typename T> class K2
    {
            K1<int> a;
            K1<T> b;
    };
    ```
- Przykład cpp_9.13
    ```