Standardowa biblioteka szablonów (STL)

Wykład 11

Standardowa biblioteka szablonów (STL)

- Rdzeniem standardowej biblioteki C++ jest tzw. standardowa biblioteka szablonów
- STL umożliwia zarządzanie kolekcjami danych przy użyciu wydajnych algorytmów, bez konieczności dogłębnego poznawania ich sposobu działania
- STL oferuje grupę klas kontenerowych zaspokajających rozmaite potrzeby wraz z algorytmami, które na nich operują
- STL wzbogaca język C++ o nowy poziom abstrakcji
  - Możemy zapomnieć o programowaniu dynamicznych tablic czy drzew oraz algorytmów do ich przeszukiwania
- Ze względu na swoją elastyczność STL wymaga objaśnienia
  - Trzeba się zapoznać z jego składnikami
  - Oraz nauczyć się wydajnego korzystania z dostarczonych algorytmów

Składniki STL

- **Kontenery** – służą do zarządzania kolekcjami obiektów określonego typu
  - Poszczególne kontenery mają różne zalety oraz wady i odzwierciedlają zróżnicowane potrzeby wobec kolekcji w tworzonych programach
- **Iteratory** – służą do poruszania się po kolekcjach
  - Oferują one interfejs wspólny dla każdego dowolnego typu kontenerowego
  - Interfejs iteratorów jest bardzo podobny operacji na wskaźnikach (możemy np. używać ++, *, ->)
- **Algorytmy** – służą do przetwarzania elementów kolekcji
  - Mogą one wyszukiwać, sortować, modyfikować lub po prostu wykorzystywać elementy
  - Algorytmy korzystają z iteratorów przez co mogą być używane do dowolnego typu kolekcji

Koncepcja STL

- Koncepcja biblioteki STL oparta jest na odseparowaniu danych od operacji
- Dane zarządzenie są przez klasy kontenerowe
- Operacje natomiast definiowane są przez konfigurowalne algorytmy
  - Operacje specyficzne dla danego kontenera są oczywiście implementowane w kontenerze
- Do łączenia danych i operacji używane są iteratory
- Koncepcja STL jest w pewnym sensie sprzeczna z ideą programowania zorientowanego obiektowo
  - Zamiast łączyć dane i algorytmy, rozdziela je
  - Wynika to z dużych możliwości takiego podejścia, ponieważ możliwe są dzięki temu różne kombinacje kontenerów i algorytmów z nimi współpracujących
- Biblioteka STL stanowi dobry przykład programowania uogólnionego (*generic programming*)

Rodzaje kontenerów

- **Kontenery sekwencyjne** – reprezentują kolekcje uporządkowane, w których każdy element posiada określoną pozycję
  - Pozycja zależy od momentu i miejsca wstawienia, ale nie zależy od samej wartości elementu
  - Należą do nich
    - `array` – statyczna tablica (*C++11*)
    - `vector` – wektor
    - `deque` – kolejka dwustronna
    - `list` – lista
    - `forward_list` – lista jednokierunkowa (*C++11_*
    - Łańcuchy – `string`, `basic_string<>`
      - Bardzo zbliżone do wektorów, ale ich elementami są znaki
- **Kontenery asocjacyjne** – będące kolekcjami sortowanymi
  - Położenie elementu zależy od jego wartości zgodnie z określonym kryterium sortowania
  - Należą do nich
    - `set` – zbiór
    - `mulitset` – wielozbiór
    - `unordered_set` – nieposortowany zbiór (*C++11*)
    - `map` – mapa
    - `multimap` – multimapa
    - `unordered_map` – mapa nieposortowana (*C++11*)

Wspólne cechy kontenerów

- Wszystkie kontenery zapewniają semantykę wartości
  - Przy wstawianiu wykonywana jest kopia obiektu
  - Elementy kontenera mogą być wskaźnikami do obiektów
- Elementy w kontenerach mają określoną kolejność
  - Możemy wykonywać wielokrotne iteracje w tej samej kolejności po wszystkich elementach
- Operacje na kontenerach nie zapewniają bezpieczeństwa
  - Funkcja wywołująca musi zapewnić spełnienie wymagań przez parametry operacji
  - Funkcje biblioteczne STL na ogół nie rzucają wyjątków

Wspólne operacje

Operacja	Skutek	Operacja	Skutek
`ConType c`	Pusty kontener	`c1.swap(c2)`	Zamiana
`ConType c1(c2)`	Inicjalizuje c2	`swap(c1,c2)`	Zamiana funkcja globalna
`ConType c1(beg, end)`	Inicjalizuje zakresem	`c.begin()`	Iterator do pierwszego elem.
`c.~ConType()`	Zwalnia pamięć	`c.end()`	Iter. do ost. elem.
`c.size()`	Liczba elem.	`c.rbegin()`	Iter odwrotny p.
`c.empty()`	Czy pusty	`c.rend()`	Iter odwrotny k.
`c.max_size()`	Maksymalna liczba elem.	`c.insert(pos, ele)`	Wstawia kopie elem.
`==, !=, <, >,`	Operacje logiczne	`c.erase(beg, end)`	Usuwa elem. z zakresu
`c1 = c2`	Przypisanie	`c.clear()`	Opróżnia konten.

Typedefs

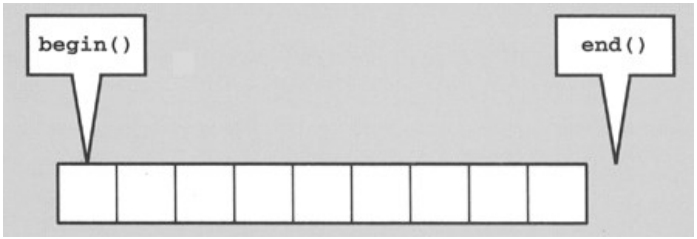
member type	definition	notes
value_type	The first template parameter (T)	
allocator_type	The second template parameter (Alloc)	defaults to: allocator<value_type>
reference	allocator_type::reference	for the default allocator: value_type&
const_reference	allocator_type::const_reference	for the default allocator: const value_type&
pointer	allocator_type::pointer	for the default allocator: value_type*
const_pointer	allocator_type::const_pointer	for the default allocator: const value_type*
iterator	a random access iterator to value_type	convertible to const_iterator
const_iterator	a random access iterator to const value_type	
reverse_iterator	reverse_iterator<iterator>	
const_reverse_iterator	reverse_iterator<const_iterator>	
difference_type	a signed integral type, identical to: iterator_traits<iterator>::difference_type	usually the same as ptrdiff_t
size_type	an unsigned integral type that can represent any non-negative value of difference_type	usually the same as size_t

Iteratory

- Iteratory są obiektami, które potrafią nawigować po elementach kontenerów
- Podstawowe operacje definiowane dla iteratorów
  - `operator*` - zwraca element z aktualnej pozycji
  - `operator++` - przesuwa iterator na pozycję następną
  - `operator==` i `!=` zwracają wartość logiczną czy iteratory reprezentują tą samą (inną) pozycję
  - `operator=` - przypisanie
- Każdy kontener definiuje co najmniej dwa typy iteratorów
  - `kontener::iterator` – przeznaczony do nawigowania w trybie odczytu i zapisu
  - `kontener::const_iterator` – przeznaczony do nawigowania w trybie tylko do odczytu
  - Zrealizowane jest to za pomocą instrukcji `typedef`

Nawigowanie po kontenerach za pomocą iteratorów

- Wszystkie klasy kontenerowe zapewniają takie same podstawowe metody, które umożliwiają nawigowanie po ich elementach
  - **`c.begin()`** – zwraca iterator reprezentujący początek elementów w kontenerze, początkiem jest pozycja pierwszego elementu
  - **`c.end()`** – zwraca iterator reprezentujący koniec elementów w kontenerze, końcem jest pozycja za ostatnim elementem
  - Obie te funkcje definiują zakres półotwarty **`[beg, end)`**
    - Zaletą jest brak specjalnej obsługi zakresów pustych oraz proste kryterium zakończenia iteracji



Algorytmy

- Algorytmy służą do przetwarzania elementów kolekcji
  - Sortowanie, kopiowanie, przestawianie …
  - Algorytmu są funkcjami globalnymi, a nie składowymi kontenerów
  - Pozwala to na jednokrotną implementację algorytmu dla wszystkich kontenerów, a nie dla każdego z osobna
  - Algorytmy pracują na zakresach (co najmniej jednym)
  - Algorytmy są oczywiście szablonami umieszczonymi w przestrzeni nazw **`std`**
- W celu używania algorytmów trzeba dołączyć plik nagłówkowy **`<algorithm>`**

Zakresy

- Zakres może obejmować cały kontener, ale nie musi
  - Dlatego w algorytmach podajmy początek i koniec zakresu
    - `sort(coll.begin(), coll.end());`
  - Algorytmy nie sprawdzają poprawności zakresów!!!
    - Funkcja wywołująca musi zapewnić poprawność zakresów, dla których wywołuje algorytm
  - Każdy algorytm przetwarza zakresy półotwarte
    - `[początek, koniec)`
    - Nie jest brany pod uwagę ostatni element zakresu!!!