STL – Standardt Template Library

- STL - Standardowa Biblioteka Szablonów - zapewnia ogromną moc językowi C++. Ta biblioteka jest szeroko stosowana podczas tworzenia aplikacji C++. Dlatego w tym kursie omówimy najważniejsze aspekty korzystania z STL.

- Ideologia C++ polega na tym, że dane klasy są umieszczone na najwyższym poziomie hierarchii, a funkcje przekształcające te dane podlegają im (Bjarne Stroustrup).

- W STL jest odwrotnie. Na najwyższym poziomie hierarchii znajdują się algorytmy, które działają tak samo dla każdego typu danych (Alexander A. Stepanov).

- Połączenie tych dwóch przeciwstawnych pojęć rodzi pewne trudności w zrozumieniu. Dlatego bardzo ważne jest, aby to zrozumieć.

- STL zawiera szablony klas ogólnego przeznaczenia oraz funkcje implementujące.

- Jądro STL składa się z *kontenerów*, *algorytmów* i *iteratorów*.

- *Kontenery* – to są obiekty, przeznaczone do przechowywania innych obiektów, przy czym istnieje kilka rodzajów kontenerów.
  - *vector* – zawiera definicje tablicy dynamicznej (podstawowy kontener)
  - *deque* – kolejki (podstawowy kontener)
  - *list* – listy (podstawowy kontener)
  - *map* – kontener stowarzyszony (associative container) – umożliwia efektywne pobranie wartości na podstawie kluczy (para klucz/wartość)

- Każda klasa kontenera zawiera definicje operujących na nim funkcji (dodawania elementu w koniec, pobierania elementu, wstawiania elementu i t. d.).

- *Algorytmy* – operują na kontenerach. Istnieją algorytmy dla kopiowania, sortowania, wyszukiwania i zamiany elementów klasy-kontenera, i t. d. Wiele algorytmów operują na zakresach elementów wewnątrz kontenera.

- *Iteratory* – to obiekty, pełniące w stosunku do obiektów role wskaźników. Umożliwiają poruszanie się po zawartości kontenera w podobny sposób, w jaki wskaźniki pozwalają przemieszczać się w obrębie tablicy. Istnieje pięć rodzajów iteratorów:

Iterator	Opis
Random Access Iterator	Zapis i odczyt wartości. Umożliwia dostęp do elementów w losowej (w dowolnej) kolejności
Bidirectional Iterator	Zapis i odczyt wartości. Umożliwia poruszanie się do przodu i do tyłu w sposób sekwencyjny (inkrementowanie lub dekrementowanie).
Forward Iterator	Zapis i odczyt wartości. Umożliwia poruszanie się tylko do przodu w sposób sekwencyjny.
Input Iterator	Tylko odczyt wartości. Poruszanie się tylko do przodu.
Output Iterator	Tylko zapis wartości. Poruszanie się tylko do przodu.

- Uwaga! Analogicznie ze strumieniami I/O :
  - Wejście – to jest pobieranie danych z kontenera (odczyt)
  - Wyjście – to jest wyprowadzenie danych w kontener (zapis)

- Iterator o większych możliwościach może być używany zamiast iteratora o bardziej ograniczonej funkcjonalności. Na przykład, forward iterator może zastępować input iterator.

- Iteratory są obsługiwane tak samo, jak i wskaźniki. Możliwa jest ich inkrementacja/dekrementacja, operator pobrania wartości obiektu przez iterator *(*iter_val*). Iteratory deklaruje się za pomocą typu *iterator* zdefiniowanego w różnych kontenerach:

Przykład:
std::vector<type> :: *iterator* my_iter; //Iterator my_iter służy dla poruszania się po kontejneru vector<type>.

- STL obsługuje także odwrotne iteratory (*reverse iterators*). Są to *bidirectional iterators* lub *random access iterators*, przemieszczające się w obrębie sekwencji w odwrotnym kierunku. Oznacza to, że inkrementacja iteratora odwrotnego, wskazującego na ostatni element sekwencji, spowoduje przejście do przedostatniego elementu.

Example W44.

- Oprócz kontenerów, algorytmów i iteratorów, STL zawiera szereg innych standardowych komponentów. Najważniejsze wśród nich to *alokatory*, *predykaty* i *funkcje porównawcze*.

- *Alokatory*. Dla każdego kontenera jest zdefiniowany *alokator*, który zarządza pamięcią, przydzieloną dla kontenera. Domyślny alokator to obiekt klasy *allocator*, ale programista może zdefiniować własny obiekt, jeśli wymaga tego specjalistyczna aplikacja. W większości wypadków domyślny alokator jest w pełni wystarczający.

- Niektóre algorytmy i kontenery wykorzystują specjalne funkcje zwane *predykatami*. Istnieją predykaty unarne (jednoargumentowe) i binarne (dwuargumentowe). Funkcje te zwracają wynik *true* lub *false*.

- Niektóre algorytmy i klasy używają specjalnych predykatów binarnych, służących do porównywania dwóch elementów.

- Typowe nagłówki STL: <vector>, <deque>, <algorithm>, <functional>, <utility> - patrz MSDN.

➢ **Nazwy typów używanych w klasach kontenerów STL:**

size_type	size_t
reference	Referencja do elementu
const_reference	Referencja do elementu zadeklarowana jako const
iterator	Iterator
const_iterator	Iterator zadeklarowany jako const
reverse_iterator	Iterator odwrotny
const_reverse_iterator	Iterator odwrotny zadeklarowany jako const
value_type	Typ wartości przechowywanej w kontenerze
allocator_type	Typ alokatora
key_type	Typ klucza
key_compare	Typ funkcji porównującej dwa klucze
value_compare	Typ funkcji porównującej dwie wartości

Vector

- Obsługuje tablicy dynamiczne – mogą być powiększane w razie potrzeby.
- Specyfikacja szablonu dla klasy *vector*:

  template <*class* T, *class* Allocator=*allocator*<T>> *class vector*

  T – typ przechowywanych danych, Allocator – określa alokator, przy czym domyślną wartością jest alokator standardowy.

- Konstruktory:

```
//tworzy pusty vector
explicit vector(const Allocator &a=Allocator());

//tworzy vector o num elementów o wartości val
explicit vector(size_type num,  const T &val = T(), const Allocator &a=Allocator());

//tworzy vector zawierający takie same elementy jak ob
vector(const vector<T, Allocator> &ob);
```

//tworzy wektor zawierający elementy z zakresu wyznaczonego przez
//iteratory *start* i *end*
template<clasInIter> *vector*(InIter *start*, InIter *end*, const Allocator &a=Allocator());

- Każdy obiekt, który będzie przechowywany w kontenerze *vector*, musi mieć zdefiniowany domyślny konstruktor, a również w zależności od funkcjonalności:

  - konstruktor sparametryzowany
  - destruktor (jeśli trzeba zwolnić pamięć)
  - konstruktor kopii
  - operator =
  - operatory ==, !=, >, <, >=, <=

- Dla kontenera *vector* są zdefiniowane operatory: **==, <, <=, !=, >, >=, [ ]**

- Do najczęściej stosowanych funkcji składowych należą:

  size(), *begin*(), *end*(), *push_back*(), *insert*(), *erase*(), *clear*().

size()	Zwraca aktualną ilość elementów tablicy
begin()	Zwraca iterator wskazujący początek wektora
end()	Zwraca iterator wskazujący koniec wektora (pierwszą wolną pozycje tablicy)
push_back(…)	Umieszcza wartość w pierwszej wolnej pozycji
insert(…)	Dodawanie elementów w środku wektora
erase(…)	Usuwanie elementów z tablicy
clear()	Usuwanie wszystkich elementów z tablicy bez dealokacji pamięci

Algorytmy

<algorithm>

- Operują na kontenerach. Mimo że każdy kontener zapewnia obsługę podstawowych operacji, to algorytmy pozwalają wykonywać bardziej rozbudowane i złożone akcji. Umożliwiają także jednoczesne prace z dwoma kontenerami różnych typów.

- Wszystkie algorytmy są szablonami funkcji. Oznacza to że można ich stosować podczas pracy z dowolnym typem kontenera.

- Lista algorytmów jest podana w MSDN. Rozważmy kilka najbardziej popularnych algorytmów.

Example W45.