Szablony funkcji

- W C++ istnieje możliwość zdefiniowania funkcji działającej na nieustalonym z góry typie danych. To tak zwany szablon funkcji. Zamiast konkretnego typu zmiennej wpisuje się jakiś identyfikator. Konkretny typ przypisujemy do identyfikatora przed użyciem szablonu. Wygląda to następująco

```
template <typename T>
T dodaj(T a, T b)
{
    return a + b;
}
```

- funkcja może być użyta do dodawania liczb dowolnego typu (dla którego zdefiniowany jest operator +), i zwraca taki sam typ.

Szablony funkcji

- pierwsza linijka wyjaśnia kompilatorowi rolę nowego identyfikatora `T` (bez średnika). Tradycyjnie do oznaczania typów używa się dużych liter lub słów zaczynających się od dużych liter.
- użycie szablonu jest następujące:

```
...
z = dodaj<int>(x, y);
...
h = dodaj<double>(u, w);
...
```

Szablony funkcji

- jeżeli w momencie użycia szablonu typ T jest jasny, nie potrzeba go jawnie podawać, np:

```
...
int x, y, z;
...
z = dodaj(x, y);
...
```

(możemy pominąć <int>).

Szablony funkcji

- Szablonów można używać z różnymi typami zmiennych, nie tylko typami fundamentalnymi. W typowym zastosowaniu typem zmiennej używanym w szablonie funkcji jest zdefiniowana przez użytkownika klasa. W takim przypadku zamiast słowa `typename` używamy słowa `class`
- w powyższym przykładzie możemy używać dowolnej klasy, dla której zdefiniowany jest (przez przeładowanie) operator +.
- typów ukrytych pod identyfikatorami może być więcej, składnia jest następująca:

```
template<typename S, typename T>
S add(S x, T y)
{
    return x + y;
}
```

Szablony funkcji

- wywołanie takiego szablonu funkcji jest następujące:

```
...
float x = 7.f; int y = 3;
...
cout << add<float, int>(x, y) << endl;
...
```

- Przykład: `Prog1.cpp`

Szablony funkcji

- Warto zapamiętać: szablony funkcji zaoszczędzają nam pisania kodu źródłowego. Nie mają natomiast wpływu na skompilowany kod wynikowy. Każde wywołanie szablonu, z nowym typem zmiennej, lub typami, powoduje, że kompilator wstawia do programu odpowiednią wersję funkcji (to jest tak zwana instancja szablonu). Wywołanie szablonu z takim samym typem zmiennej, lub typami, co wcześniej, nie dodaje nowego kodu, wykorzystana jest funkcja wstawiona wcześniej.
- Biblioteka STL, o której zaraz powiemy zawiera wiele gotowych szablonów, w tym szablonów funkcji

Szablony

- Podobnie jak szablony funkcji, w C++ występują szablony klas. Wyobraźmy sobie często stosowaną strukturę, stos. Napiszmy klasę realizującą stos zmiennych `int`

```
class IntStack
{
  static const int ssize = 100;
  int stack[ssize];
  int top;
public:
  IntStack() : top(0) {}
  void push(int i) {stack[top++] = i;}
  int pop() {return stack[--top];}
};
```

Szablony

- jest to typowa implementacja stosu, z dwoma podstawowymi metodami.
- Przykład: `Prog2.cpp`
- nie ma potrzeby przepisywania tego kodu jeżeli potrzebujemy stosu obiektów innych niż `int`. Służą do tego szablony

```
template<class T>
class IntStack
{
  static const int ssize = 100;
  T stack[ssize];
  int top;
public:
  IntStack() : top(0) {};
  void push(const T& i) {stack[top++] = i;}
  T pop() {return stack[--top];}
  int size() {return top;}
};
```

Szablony

- Przykład `Prog2a.cpp`
- Dostępna jest możliwość tzw. specjalizacji szablonu. Jeżeli dla pewnego konkretnego typu `T` potrzebujemy szczególnej realizacji funkcji,jest to możliwe, także w ramach szablonu.
- `Prog2b.cpp`
- C++ oferuje gotową bibliotekę szablonów STL. Nie jest to część standardu języka C++, ale jest projektem towarzyszącym C++ i też jest standardem.
- Szablony zgromadzone w STL można podzielić na 3 grupy: kontenery, iteratory i algorytmy.
- Kontenery to struktury danych, w których występuje pewna ilość pewnych obiektów tej samej klasy. Widzieliśmy tego typu struktury, były to np. listy. Kontener przechowuje obiekty, oraz oferuje pewną dodatkową funkcjonalność.

Szablony

- Przykładowa, typowa funkcjonalność kontenerów:
  - tworzenie kontenera (poprzez konstruktor)
  - dodawanie/usuwanie obiektów z kontenera
  - dostęp do obiektów
  - sortowanie
  - raportowanie rozmiaru
  - opróżnianie kontenera
- Dostępne w STL kontenery mogą się różnić wbudowaną funkcjonalnością. Więcej wbudowanej funkcjonalności może być wygodniejsze dla programisty, ale wymaga więcej zasobów od systemu operacyjnego oraz spowalnia działanie.
- Kontenery mogą być typu `value`, to znaczy przechowywać własne kopie obiektów, lub typu `reference`, to znaczy przechowywać wskaźniki do obiektów istniejących niezależnie. W tym drugim przypadku kontener nie zajmuje sią tworzeniem ani usuwaniem składników.

Szablony

- Najbardziej popularnym kontenerem zawartym w STL jest `vector`, czyli lista, zajmiemy się nim za moment.
- Przypomnijmy, wszystkie klasy dostępne w STL są szablonami. W momencie ich użycia specyfikujemy, jakiego typu elementy będą przechowywać.

Szablony

- Iteratory to narzędzie do poruszania się po kontenerach. Pamiętamy, że w liście dwukierunkowej mając dostęp do konkretnego elementu mieliśmy łatwy dostęp do elementów następnego i poprzedniego. Było to zrealizowane przy pomocy wskaźników. Iteratory zachowują się podobnie do wskaźników.
- Kontenery mają wbudowane iteratory, jako typ zmiennej. Różne kontenery mogą mieć iteratory o różnych funkcjonalnościach. Jednak podstawowe operacje na iteratorach są wspólne dla różnych typów. Powtórzmy, iteratory w działaniu przypominają wskaźniki, i tak można o nich myśleć.
- Każdy kontener zawiera przynajmniej dwa typy iteratorów: `iterator` oraz `const_iterator`. Ten drugi uniemożliwia zmianę wskazywanego elementu.
- Dla iteratorów dostępne są operacje `*`, `++`, `==`, `!=`, `=`. Łatwo się domyśleć, jak działają.

Szablony

- W każdym kontenerze dostępne są funkcje:
  - `begin()`
  - `end()`
  - `cbegin()`
  - `cend()`
- `begin()` zwraca iterator wskazujący na pierwszy element kontenera. `end()` zwraca iterator wskazujący na adres następny po ostatnim elemencie kontenera. Nie jest to dziwne, porównując wartość iteratora z wartością zwracaną `end()` sprawdzamy, czy przejrzany został już cały kontener.
- Funkcje `cbegin()` i `cend()` zachowują się tak samo, ale zwracają iterator typu `const_iterator`.
- Przykład `Prog2c.cpp`, `Prog2d.cpp`, `Prog2e.cpp`.

Szablony

- Przyjrzyjmy się dokładniej szablonowi `vector`. Jest to uporządkowana lista, do której elementów jest dostęp przy pomocy operatora `[]`, tak jak do elementów tablicy (niezależnie od możliwego dostępu przy pomocy iteratorów). Elementy można wstawiać w dowolne miejsce i usuwać z dowolnej pozycji. Szablon zawiera iterator typu *random access*, najbardziej uniwersalny
- Podobne do `vector` kontenery to `list`, `array` i `deque`. Każdy z nich ma swoją specyfikę, dostępne metody i szybkość działania
- Przykład: `Prog3.cpp`.

Szablony

- Szablon `vector` daje możliwość wstawiania i usuwania elementów w dowolnym punkcie listy, ale wstawianie i usuwanie na końcu są znacznie szybsze. Szablon `list` nie pozwala na wstawianie czy usuwanie elementów w dowolnych miejscach, ale działa szybciej niż `vector`. Należy pamiętać o tego typu sprawach.
- Wymienione powyżej kontenery to tak zwane *sequence containers*. Są też inne typy, takie jak `set`, `map` czy `stack`
- Nie będziemy omawiać wszystkich istniejących kontenerów. Pracując z konkretną strukturą danych należy znaleźć właściwy dla niej kontener i szczegółowo się z nim zapoznać.
- Zwróćmy uwagę, że w STL nie ma kontenera `tree`. Wynika to z filozofii nazewnictwa. Nazwy są ogólne, i poszczególne kontenery mogą realizować różne modele danych.
- Na przykład drzewo można zrealizować jako `set` lub `map`

Szablony

- Używając iteratora, przeglądanie wektora v6 z poprzedniego przykładu możemy zrealizować następująco

```
vector<int>::const_iterator it;
  it = v6.cbegin();
  while (it != v6.cend())
  {
       cout << *it << " ";
      ++it;
  }
```

Szablony

- Algorytmy to szablony funkcji, realizujące typowe operacje, na przykład sortowanie czy wyszukiwanie.
- Szablony funkcji realizujące różnego rodzaju często stosowane operacje dostępne są w bibliotece `algorithm`, będącej częścią STL.
- Przykład `Prog4.cpp`.
- Zauważmy ciekawą pętlę. To jest tak zwana pętla *range based loop* wprowadzona w jednej z ostatnich rewizji standardu C++. Parametrem tej pętli jest para *typ_elementu : zakres*. W naszym przypadku zakresem jest wektor. Zakresem może być jakakolwiek uporządkowana struktura dla których podstawowe funkcje takie jak `begin()` i `end()` są zdefiniowane.