Szablony w C++

- Po co używać szablonów?
- C++ wymaga zmiennych, funkcji, klas itp. ze specyficznymi typami danych.
- Wiele algorytmów (np. quicksort) posiada ten sam kod ale dla różnych typów danych.
- Po co używać szablonów?
- Bez szablonów jedna z poniższych opcji musi zostać użyta:
  - Re-implementacja algorytmu dla każdego typu danych
  - Napisanie kodu ogólnego z użyciem Object lub *void
  - Użycie preprocesorów np.
    - **#define getmax(a,b) a>b?a:b**
- Po co używać szablonów?
- R-eimplementajca powoduje rednundację kodu co może prowadzić do błędów
- Napisanie kodu ogólnego pomija sprawdzanie wszystkich typów
- Preprocesory zamieniają tekst „na oślep” i może to wprowadzać błędy poprzez kod którego programista nawet nie widział podczas kompilacji.
- Przykład – bez szablonu

Zwróć wartość maksymalną dwu parametrów

```
int getmax(int a, int b) {
     return (a > b) ? a : b;
}
double getmax(double a, double b) {
     return (a > b) ? a : b;
}
float getmax(float a, float b) {
     return (a > b) ? a : b;
```

```
}
void* getmax(void *a, void *b) {
     return (*a > *b) ? a : b;
}

#define getmax(a,b) (a>b)?a:b;
```

- Przykład - szablon
- Zamiast wcześniejszego kodu można to zrobić za pomocą szablonu funkcji.

```
template <typename T>
T getmax(T a, T b) {
   return (a > b) ? a : b;
}
```

- Typ użyty przez tą funkcję jest definiowany w momencie wywołania.

```
int main() {
   cout << getmax(6,7) << endl;
}
```

- Przykład – różne typy
- Dla kompilatora, aby dopasować typ do szablonu, typ obu argumentów musi się zgadzać. Poniższy przykład spowoduje błąd:

```
template <typename T>
T getmax(T a, T b) {
   return (a > b) ? a : b;
}
int main() {
   cout << max(7,8.0) << endl;
}
```

- Przykład – różne typy
- Może być to rozwiązane na kilka sposobów:

```
template <typename T>
T getmax(T a, T b) {
   return (a > b) ? a : b;
}
int main() {
   cout << getmax((double)7,8.0) << endl;
   cout << getmax<double>(7,8.0) << endl;
}
```

- Lub:

```
template <typename T1, typename T2>
T1 getmax(T1 a, T2 b) {
   return (a > b) ? a : b;
}
```

- Przykład – różne typy

- Powoduje to nowy problem. Typ zwracany jest taki sam jak typ pierwszy.

```
int main() {
   cout << getmax(7,8.0) << endl;
   //jest inne od
   cout << max(7.0,8) << endl;
}
```

Przeciążanie szablonów funkcji

- Jak inne funkcje w C++, szablony funkcji również mogą być przeciążane. Na przykład z użyciem specyficznego typu.

```
int getmax(int a, int b) {
   return (a > b) ? a : b;
}
template <typename T>
T getmax(T a, T b) {
   return (a > b) ? a : b;
}
int main() {
   cout << getmax(5,6) << endl;      //funkcja
   cout << getmax(5.0,6.0) << endl; //szablon
}
```

Szablony klas

- Kolejnym powszechnym użyciem szablonów są szablony klas. Jest to szczególnie użyteczne dla klas kontenerowych, które są używane do przechowywania obiektów.

```
template <typename T>
class Koszyk {
   private:
   std::vector<T> items;
   public:
   void push(T const&);
   void pop();
```

```
   T top() const;
   bool empty() const { return items.empty();}
};
```

- Funkcje klasy powinny być zapisane z szablonem.

```
template <typename T>
void Koszyk<T>::push(T const &a) {
   items.push_back(a);
}
```

- Te klasy mogą zostać użyte tak aby zdefiniować przechowywany typ:

```
int main() {
   Koszyk<int> intKoszyk;
   Koszyk<float> floatKoszyk;
   Koszyk<string> stringKoszyk;
}
```

- Klasy mogą być specjalizowane dla szczególnych typów.

```
template <>
class Koszyk<int> {
   private:
   vector<int> items;
   public:
   void push(int);
   void pop();
   int top();
   bool empty() const {return items.empty();}
};
```

- Klasa może być również częściowo specjalizowana.

```
template <typename T1, typename T2>
class myClass{
   ...
};
template <typename T>
class myClass<T, T>{
   ...
};
template <typename T>
class myClass<T, int>{
   ...
};
template <typename T1, typename T2>
class myClass<T1*, T2*>{
   ...
```

```
};
```

- Może posiadać domyśle argumenty:

```
template <typename T, typename CONT=vector<T> >
class Koszyk {
   private:
   CONT items;
   public:
   void push(T const &);
   void pop();
   T top() const;
   bool empty() const {items.empty();}
};

int main() {
   Koszyk<int> vectorKoszyk; //z użyciem wektora
   Koszyk<int, deque<int> > dequeKoszyk; //z deque
}
```

Parametry szablonów w postaci wartości

- Parametry szablonu nie muszą być zdefiniowane jako typy. Parametry te mogą być także zwykłymi wartościami:

```
template <typename T, int MAXSIZE>
class Koszyk {
    private:
    T items[MAXSIZE];
    int numItems;
    public:
    Koszyk();
    void push(T const &);
    void pop();
    T top() const;
    bool empty() const;
    bool full() const;
};
```

- Pozwala to Koszykowi być tworzonym jako obiekt określonego typu i rozmiaru

```
int main() {
      Koszyk<int, 20> int20Koszyk;
      Koszyk<int, 40> int40Koszyk;
}
```

- Ważne jest, aby zauważyć, że te dwa przypadki są różnych typów. Koszyk <int, 20> jest inny niż Koszyk <int, 40>.

Parametr szablonu a szablon klasy

- Parametr szablonu może być również szablonem klasy. Może to być przydatne w celu uniknięcia następujących sytuacjach:

```
template <typename T, typename CONT=vector<T> >
class Koszyk {
   ...
};
int main() {
   Koszyk<int> vectorKoszyk; //z wektorem
   Koszyk<int, deque<int> > dequeKoszyk; //z deque
}
```

- Typ Koszyk jest zdefiniowany dwa razy:
  - **int and deque<int>**
- Byłoby lepiej, aby była możliwość określić Koszyk jako:

```
int main() {
   Koszyk<int, vector> vectorKoszyk; //jako wektor
   Koszyk<int, deque> dequeKoszyk; //z deque
}
```

- Aby to zrobić, musimy zdefiniować drugi parametr jako parametr szablonu szablonu. Poniższy kod definiuje to:

```
template <typename T, template<typename ELEM> class
   CONT = vector >
class Koszyk {
...
};
```

- Istnieje jednak problem z tym. Wektor SDT posiada więcej niż jeden parametr. Drugi jest to alokator. To musi zostać zdefiniowane w celu poprawnego działania szablonu:

```
template <typename T,template<typename ELEM, typename
   ALLOC = allocator<ELEM> >
     class CONT = vector >
class Koszyk {
...};
```

- Wszystkie te szablony pozwalają na utworzenie Koszyka jak poniżej:

```
int main() {
   Koszyk<int, vector> vectorKoszyk;
   Koszyk<int, deque> dequeKoszyk;
}
```