

Tematy

- Dedukcja typów, operator decitype
- Wyrażenia Lambda
- Inicjalizacja (uniform initialization)
- delete i default
- nullptr
- Delegowanie konstruktorów
- Zmiany w bibliotece STL





Dedukcja typów, operator decltype

```
    wcześniej konieczność specyfikacji typu

 auto
auto x = 0; //x typu int, bo 0 jest całkowite auto c = 'a'; //char
auto d = 0.5; //double
void func(const vector<int> &vi) {
  vector<int>::const iterator ci = vi.begin();
  auto cj = vi.begin();
                                  C++11 zmieniło znaczenie słowa auto
}
                                (wcześniej dotyczyło przechowywania jak:
 operator decitype
                               automatic, static, register, external)
const vector<int> vi;
typedef decltype (vi.begin()) ConstIt;
                                                            //int
ConstIt another_const_iterator;
                                                            //double
int fun1(int a, double b) { return a + b; }
auto fun2(int a, double b) { return a + b; }
                                                            //int
                                                           //int
auto fun3(int a, double b) -> int { return a + b; }
auto fun4(int a, double b) -> decltype(a) { return a + b; }
```

Lambda zamiast funkcji

Warszawska

```
    Lokalna definicja funkcji

int upCnt;
void upperCnt(char c) {
  if (isupper(c))
  upCnt++;
int main() {
  char s[] = "Hello World!";
  for_each(s, s + sizeof(s), upperCnt);
  cout << "Wielkich liter w :" << s << "jest " << upCnt << endl;</pre>
                                                           przeniesione ciało funkcji
//modified by the lambda
                                                           "funkcja" otrzymuje
  int Uppercase = 0;
  for_each(s, s + sizeof(s), [&Uppercase](char c) 
                                                           referencje na zmienną
     if (isupper(c)) Uppercase++;}
);
cout << "Wielkich liter w :" << s << "jest " << Uppercase << endl;</pre>
return 0;
                                         Wielkich liter w: Hello World!jest 2
                                         Wielkich liter w: Hello World!jest 2
  Warszawska
```

Obiekty funkcyjne (funktory)

- Obiekt funkcyjny: obiekt, który łączy w sobie cechy obiektu i funkcji: można go utworzyć/wygenerować, przekazać jako argument, "wywołać", itp.
- Formalnie obiektem funkcyjnym jest dowolny obiekt, dla którego jest zdefiniowany operator wywołania funkcji: operator()(params)

```
template<typename T>
class Inside {
    const T &a, &b; // Przedział domknięty [a, b]
public:
    Inside(const T& aa, const T& bb) : a(aa), b(bb) { }
    bool operator()(const T& x) const // wołanie funkcji
    {
        return x >= a && b >= x;
    }
    // ...
};
Politechnika
```

Obiekty funkcyjne (funktory) (cd1)

1 2 4 0 4 4 3 3 2 4 0 0 1 2 1 1 0 2 2 1 Liczba zer i jedynek w A: 9



varszawska

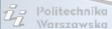
5

5

Funkcje (wyrażenia) lambda

- Wyrażenia lambda pozwalają utworzyć obiekt funkcyjny anonimowy i przekazać go do algorytmu – konsumenta
- W poprzednim przykładzie: zamiast definiować klasę Inside i według niej utworzyć odpowiedni obiekt funkcyjny można pisać:

- To wyrażenie korzysta z otoczenia tylko poprzez parametr x
- Można określić związanie wyrażenia lambda z otoczeniem podając odpowiednią listę capture w [] (określa to tak zwane domknięcie wyrażenia lambda (closure))



Funkcje (wyrażenia) lambda (cd1)

Możliwe formy funkcji lambda (od max do min):

```
[capt] (pars) mutable except attrib -> ret {body}
[capt] (pars) -> ret {body}
[capt] (pars) {body}
[capt] {body}
```

- [capture]
 - [] domknięcie puste (lambda nie korzysta z otoczenia)
 - [=] domknięcie wg wartości (zmienne lokalne przechwycone przez wartość)
 - [&] domknięcie przez referencję
 - [&x, y] zmienna x przechwycona przez referencję, y przez wartość
- Wyrażenia lambda powinny być możliwie proste

Uwaga! Lambda nie jest funkcją, lecz obiektem funkcyjnym (obiekt pozwala przechowywać także jego stan pomiędzy kolejnymi wywołaniami).

Warszawska

Funkcje (wyrażenia) lambda (cd2)

```
int main() {
    array<int, 20> A; // Zawartosć nieokreslona
    for (auto& elem : A) elem = rand() % 5;
    for (auto& elem : A) cout << elem << ' ';</pre>
    cout << endl;</pre>
    cout << "Zera i jedynki wg lambda: "</pre>
        << count_if(A.begin(), A.end(),
        [](int x){ return x >= 0 && 1 >= x; })
        << endl;
    // Przechwyt zmiennych lokalnych przez wartość
    for (int a = 0, b = 4; a <= b; ++a, --b) {
        cout << "W przedziale [" << a << ".." << b << "]: "</pre>
          << count_if(A.begin(), A.end(),
                                                        12404433240012110221
          [=](int x){return x >= a && b >= x; })
                                                        Zera i jedynki wg lambda: 9
                                                        W przedziale [0..4]: 20
                                                        W przedziale [1..3]: 12
   Politechnika
                                                        W przedziale [2..2]: 5
   Varszawska
```