Wyrażenia lambda

dr inż. Ireneusz Szcześniak

jesień 2017

1 Zadanie

Stworzyć kolejkę priorytetową liczb całkowitych. Włożyć liczby 2, 3, 1, a następnie wyciągnąć te liczby i wypisać.

1.1 Rozwiązanie

```
#include <iostream>
#include <queue>

using namespace std;

int
main(void)
{
  priority_queue<int> q;

  q.push(2);
  q.push(1);
  q.push(3);

  while(!q.empty())
    {
      cout << q.top() << endl;
      q.pop();
    }

  return 0;
}
```

2 Wywołanie funkcji przez wskaźnik

Wyrażenie, które jest tylko nazwą funkcji oznacza pobranie adresu funkcji. Używając tego adresu możemy wywołać funkcję. Jedyne operacje możliwe na wskaźniku do funkcji to: pobranie adresu funkcji i wywołanie funkcji.

```
#include <iostream>
#include <functional>

bool
foo(const int &a, const int &b)
{
  std::cout << "foo:␣a␣=␣" << a << ",␣b␣=␣" << b << '\n';
}

using foo_type = bool(const int &a, const int &b);

int
```

```
main()
{
  auto f1 = foo;
  // auto f1 = &foo;
  f1(10, 20);
  (*f1)(10, 20);

  bool (*f2)(const int &, const int &) = foo;
  // bool (*f2)(const int &, const int &) = &foo;
  f2(10, 20);
  (*f2)(10, 20);

  foo_type *f3 = foo;
  // foo_type *f3 = &foo;
  f3(10, 20);
  (*f3)(10, 20);

  std::function<bool(const int &, const int &)> f4 = foo;
  // std::function<bool(const int &, const int &)> f4 = &foo;
  f4(10, 20);
  // (*f4)(10, 20); // This doesn't fly.
}
```

3 Zadanie

Domyślnie kolejka priorytetowa zwraca największy element. Przerobić rozwiązanie wyżej z użyciem własnej funkcji porównującej elementy, żeby kolejka zwracała najmniejszy element.

3.1 Rozwiązanie

```
#include <functional>
#include <iostream>
#include <queue>

using namespace std;

bool
foo(const int &a, const int &b)
{
  return a < b;
}

int
main(void)
{
  // priority_queue<int, vector<int>,
  //                bool(*)(const int &, const int &)> q(foo);
  priority_queue<int, vector<int>,
                 function<bool(const int &, const int &)>> q(foo);

  q.push(2);
  q.push(1);
  q.push(3);

  while(!q.empty())
```

```
    {
      cout << q.top() << endl;
      q.pop();
    }

  return 0;
}
```

4 Zadanie

Domyślnie kolejka priorytetowa zwraca największy element, bo do porównania używa klasy funktora `std::less<T>`. Użyć klasy `std::greater<T>`, żeby kolejka zwracała najmniejszy element.

4.1 Rozwiązanie

```
#include <iostream>
#include <queue>
#include <vector>

using namespace std;

int
main(void)
{
  // HERE'S THE DIFFERENCE!
  priority_queue<int, vector<int>, greater<int>> q;

  q.push(2);
  q.push(1);
  q.push(3);

  while(!q.empty())
    {
      cout << q.top() << endl;
      q.pop();
    }

  return 0;
}
```

5 Zadanie

Do sortowania kolejki użyć funktora. Kolejka priorytetowa powinna zwracać najmniejszy element.

5.1 Rozwiązanie

```
#include <iostream>
#include <queue>

using namespace std;

struct CMP
{
  bool
  operator()(const int &a, const int &b)
  {
```

```
    return a > b;
  }
};

int
main(void)
{
  priority_queue<int, vector<int>, CMP> q;

  q.push(2);
  q.push(1);
  q.push(3);

  while(!q.empty())
    {
      cout << q.top() << endl;
      q.pop();
    }

  return 0;
}
```

6 Zadanie

Do sortowania kolejki użyć funktora, ale tym razem umożliwić w czasie uruchomienia wybór kierunku sortowania kolejki: malejący albo rosnący w zależności od argumentu wywołania konstruktora funktora.

6.1 Rozwiązanie

```
#include <iostream>
#include <queue>

using namespace std;

struct CMP
{
  bool m_order;

  CMP(bool order): m_order(order)
  {
  }

  bool
  operator()(const int &a, const int &b)
  {
    return m_order ? a < b : a > b;
  }
};

int
main(void)
{
  // priority_queue<int, vector<int>, CMP> q(false);
  priority_queue<int, vector<int>, CMP> q(CMP(false));

  q.push(2);
  q.push(1);
```

```
  q.push(3);

  while(!q.empty())
    {
      cout << q.top() << endl;
      q.pop();
    }

  return 0;
}
```