Programowanie 1

Zadanie 14

Piotr Błaszyński

24 stycznia 2023

Przygotować program do przetestowania kilku ciekawych (niekoniecznie podstawowych) elementów języka C++. Należy uzupełnić funkcje demonstrujące działanie następujących elementów języka: łańcucha znaków (std::string), wektora (std::vector), mapy (std::map), iteratorów (::iterator) oraz prostych generatorów.
Funkcja main (i pliki nagłowkowe) wyglądają następująco:

```
#include "pch.h"
#include <vector>
#include <map>
#include <algorithm>
#include <iterator>
#include <iostream>
#include <string>
#include <cstdlib>
#include <ctime>
//reszta kodu
int main(int argc, char* argv[])
{
  char tmp;
  srand(static_cast<unsigned int>(time(0)));
  DemoVectors();
  DemoFunctors();
  DemoStrings();
  DemoMap();
  std::cout << std::endl;
  std::cin >> tmp;
  return 0;
}
```

Kody poszczególnych funkcji, oznaczone miejsca do dodania własnego kodu:

```
void DemoVectors()
{
  std::vector <int> vec; //Tworzymy wektor o nieznanym
     rozmiarze poczatkowym
  std::vector <int>::iterator MyRes;//iterator moze
     rowniez byc wynikiem funkcji szukajacej
  vec.push_back(11);//dokladamy pojedyncze elementy na
     koniec wektora
  vec.push_back(22);
  vec.push_back(13);
  vec.push_back(25);
  vec.push_back(17);
  vec.push_back(29);
  reverse(vec.begin(), vec.end());//wykonujemy
     odwrocenie kolejnosci elementow
  copy(vec.begin(), vec.end(), std::ostream_iterator<int
     >(std::cout, "\n")); //wypisanie wektorow
  res = find(vec.begin(), vec.end(), 11); //szukamy
     elementu o wartosci 11
  if (res != vec.end()) std::cout << *res << std::endl;
     //jesli znajdziemy to wypisujemy
  res = find(vec.begin(), vec.end(), 19);//funkcja
     zwraca iterator do znalezionego elementu lub vec.
     end()
  /* uzupelnic o wyswietlenie znalezionego elementu lub
     informacji o braku elementu*/
}
```

```
struct ltstr
{
  bool operator()(const char* s1, const char* s2) const
  {
    return strcmp(s1, s2) < 0;
  }//musimy utworzyc funkcje porownujaca, ktora
     wykorzystamy przy mapie
};

void DemoMap()
{
```

```
std::map<const char*, int, ltstr> months;

months["styczen"] = 31;  months["luty"] = 28;
months["marzec"] = 31;   months["kwiecien"] = 30;
months["maj"] = 31;      months["czerwiec"] = 30;
months["lipiec"] = 31;   months["sierpien"] = 31;
months["wrzesien"] = 30; months["pazdziernik"] = 31;
months["listopad"] = 30; months["grudzien"] = 31;

std::cout << "lipiec ma: " << months["lipiec"] << " dni." << std::endl;
std::map<const char*, int, ltstr>::iterator month = months.find("lipiec");
std::map<const char*, int, ltstr>::iterator prev_month = month;
auto next_month = month;//zamiast dlugiego zapisu mozna uzyc auto
next_month++;
prev_month--;
std::cout << "Poprzedni (w porzadku alfabetycznym) jest " << (*prev_month).first << std::endl;//first to pierwszy element mapy
std::cout << "Nastepny (w porzadku alfabetycznym) jest " << (*next_month).first << std::endl;
next_month++;
prev_month--;
std::cout << "Poprzedni (w porzadku alfabetycznym) jest " << (*prev_month).first << std::endl;
std::cout << "Nastepny (w porzadku alfabetycznym) jest " << (*next_month).first << std::endl;
month = months.begin();
std::cout << "Pierwszy (w porzadku alfabetycznym) jest " << (*month).first << std::endl;
std::cout << "I ma " << (*month).second << " dni" << std::endl;
month++;
do
{
  std::cout << "Nastepny (w porzadku alfabetycznym) jest " << /* uzupelnic */ << std::endl;
```

```
    std::cout << "I ma " << /* uzupelnic */ << " dni" <<
        std::endl;
    month++;
  } while (month != months.end());
}
```

```
void DemoStrings()
{
  std::string str1, str2;//deklarujemy napisy o
     nieokreslonej dlugosci
  std::string name("Piotr");
  str1 = "Moge w koncu robic normlane przypisania do
     stringow.";
  str2 = "Teraz lancuchy znakow sa traktowane jak
     normalne zmienne.";
  std::cout << str1 << std::endl;
  std::cout << str2 << std::endl;
  try {//probujemy wykonac to co ponizej
    str1.replace(str1.find("normla"), 6, "normal");
  }
  catch (...) {//A jak sie nie uda
    std::cout << "Podano bledny argument do funkcji
       replace";
  }//Wyjatkow uzywa sie do obslugi sytuacji WYJATKOWYCH
  std::cout << name << std::endl;
  name = str1 + " " + str2; //laczenie odbywa sie przy
     pomocy normalnego dodawnaia
  std::cout << name << std::endl;
  reverse(name.begin(), name.end());//odwracanie
     standardowym algorytmem
  std::cout << name << '\n'; //mozna zamiennie uzywac
  reverse(name.begin(), name.end());
  name += " I Jak?";
  std::cout << name.c_str() << std::endl;//jak wyciagnac
      wartosc do funkcji ktore potrzebuja char *
  /* uzupelnic */
  // wypisac lancuchy znakow a10 a9 a8 ... a2 a1 a0
}
```

```
void DemoFunctors() //Jak mniej wiecej dzialaja funktory
{
```

```
  std::vector<int> vec(10); // tworzymy 10-cio
     elementowy wektor liczb calkowitych
  generate(vec.begin(), vec.end(), rand);//wypelniamy go
      losowymi wartosciami
  std::for_each(vec.begin(), vec.end(), [](int &i) {i %=
      100; });//to co ponizej
  std::vector <int>::iterator i; //tworzymy iterator do
     wykorzystania w petli
  for (i = vec.begin(); i < vec.end(); i++) //zaczynamy
     od poczatku wektora
    *i %= 10; //iterator mozna traktowac jak wskaznik
  copy(vec.begin(), vec.end(), std::ostream_iterator<int
     >(std::cout, "\n"));//kopiujemy nasz wektor na
     wyjscie
  std::cout << std::endl;
  sort(vec.begin(), vec.end()); //a teraz go sortujemy,
     prosze zauwazyc,
  // ze nie musielismy nawet tworzyc funkcji
     porownujacej nasz wektor
  /* uzupelnic */
  // jeszcze raz pokazujemy nasz wektor
}
```