Języki Programowania, 13.01.2021
Zadanie 12

Dzisiaj poznamy elementy języka C++11 (czyli nowszej implementacji C++). Co prawda istnieją jeszcze nowsze standardy, ale to C++11 przyniósł spore zmiany (nowe funkcjonalności i ułatwienia)

Kilka najwazniejszych nowych elementów:

1. Wyrażenia Lambda
   Czym one są?
   „Wyrażenia lambda umożliwiają tworzenie anonimowych funkcji, czyli funkcji które posiadają ciało lecz nie posiadają swojej nazwy.”
   http://cpp0x.pl/kursy/Kurs-C++/Poziom-X/Wyrazenia-lambda-C++11/423
   Przykład:
   ```
   function<double(double,double)> sum = [](double x, double y)
   { return x+y; };
   cout<<"Sum: "<<sum(4,5)<<endl<<endl;
   ```
2. Zmienne z automatycznie deklarowanym specyfikatorem
   ```
   auto c = 'a';       //zmienna c jest typu ‘char’
   auto num = 3.14;    //zmienna num jest typu ‘double’
   ```
3. Zmienne ‘auto’ w wyrażeniach Lambda
   ```
   auto sum_auto = [](double x, double y) { return  x+y; };
   cout<<"Sum auto: "<<sum_auto(4,5)<<endl<<endl;
   ```
4. Uproszczone iterowanie po kontenerach danych (`std::vector`, `std::list`, `std::map`, itp.)
   ```
   vector<int> vec = {1, 5, 3};
   for(auto &it : vec)
        cout<<it<<endl;
   ```
5. Tworzenie kontenerów zawierających wyrażenia Lambda
   ```
   function<double(double,double)> sum2 = [](double x, double y)
   { return x+y; };
   auto sub2 = [](double x, double y) { return x-y; };
   vector<decltype(sum2)> vecLambda = {sum2, sub2,  [](double x,
   double y) {return x*y; } };   //tworzenie wektora
   for(auto it : vecLambda)      //iteracja po gotowym wektorze
   {
      cout<<lamNames[idx]<<it(4,5)<<endl;
   }
   ```
6. Przechwytywanie typów przez Lambdę, pętla `for_each`
   ```
   const char *text = "Hello World";
   int upperCase = 0;
   for_each(text, text+sizeof(text), [&upperCase](char c)  {
   if(isupper(c))
      upperCase++;
   });
   cout<<"Ilosc wielkich liter "<<upperCase<<endl<<endl;
   ```
7. Funkcje zwracające typ `auto`
   ```
   template <class T>
   auto wrapperFun(T t) -> decltype(funkcja(t))
   {
     return funkcja(t);
   }
   ```

Co należy wykonać:

1. Piszemy klasę `Calculator`. Będzie ona obsługiwała 4 działania (+,-,*,/)
2. Klasa `Calculator` posiada następujące pola:
   - `fX`, `fY` – liczby typu `double`
   - `lamVec` – wektor wyrażeń Lambda na funkcje opisujące 4 działana **1p.**
3. Konstruktor domyślny – tworzy wyrażenia Lambda dla 4 działań, dodaje je do wektora, uruchamia w pętli funkcję `Run` 2**p.**
4. Funkcja `double Eval(function<double(double,double)> lambda)` – ma jedną linijkę – uruchamia podane w argumencie wyrażenie Lambda i zwraca jego wynik **0.5p**
5. Funkcja `void Run(bool &koniec)` – wykonuje jeden raz działania kalkulatora (prosi użytkownika o podanie z klawiatury wartości, iteruje po wektorze wyrażeń Lambda, zwraca wynik każdego działania dla zadanych liczb (za pomocą funkcji `Eval`), na koniec pyta się użytkownika, czy program ma się skończyć) **1p**
6. W funkcji głównej `main` wystarczy tylko użyć konstruktora 0**.5p**

```
int main()
{
  Calculator();

  return 1;
}
```