Języki programowania, 8.01.2019
Zadanie 11 - ostatnie

Dzisiejsze zadanie ma charakter projektowy. Przychodzi do Was promotor (albo szef w pracy :)) i prosi o zbadanie za pomocą detektora promieniowania pewne nieznane źródło promieniotwórcze – należy zmierzyć jego czas połowicznego rozpadu. Nasz detektor, jaki mamy na wyposażeniu laboratorium, to jednak prosty licznik, który po prostu zlicza ilość rozpadów w danej jednostce czasu (czyli na „wyjściu" mamy tylko jedną liczbę). Wiadomo, że aby określić jaki jest czas połowicznego rozpadu tego źródła, trzeba zmierzyć rozkład promieniowania i „dopasować" do niego parametry rozkładu Poissona, który opisuje prawo rozpadu promieniotwórczego (ilość jąder rozpadających się w czasie). Zatem aby zmierzyć źródło promieniotwórcze potrzebujemy stworzyć histogram z dużej próbki danych z detektora.
W naszym zadaniu promotor/szef prosi każdego z Was o stworzenie takiego narzędzia do pracy z detektorem – histogramu, który wyglądałby jak przykład przedstawiony pod spodem, oraz jego przetestowanie. Narzędzie trzeba napisać w sposób uniwersalny, to znaczy możemy zadać dowolną liczbę binów histogramu jak i jego dowolny przedział (dolną granicę pierwszego binu i górną granicę ostatniego binu). Histogram powinien również przechowywać informacje o ilości wejść dla wartości które były mniejsze (underflow) i większe (overflow) od przedziału histogramu.

```
Printing histogram HistTest
underflow: 2
overflow: 5
0: *
1: *****
2: *****
3: *******
4: ********
5: ******
6: *********
```

gdzie liczba * oznacza jedno wejście w binie o zadanym numerze.

Histogramy możemy również np. przez siebie dzielić. Wtedy liczba zliczeń w danym binie nie będzie już całkowita i nie możemy stosować gwiazdek do wypisywania zawartości histogramu.

W naszym zadaniu napiszemy sobie histogram jako klasę szablonową, która będzie miała wyspecjalizowane funkcje (w zależności od tego co histogram przechowuje). Oto przykład metody `Dodaj` dla histogramów:

```
template<>
void Histogram<int>::Print()
{
	for(int i=0; i<nr_bins; i++)
	{
		cout<<i<<": ";
		for(int j=0; j<array_bins[i]; j++)
			cout<<"*";
		cout<<endl;
	}
	//to tylko przykład – tu robimy cokolwiek z obiektami typu Histogram, np. tworzymy wyjsciowy histogram gdzie w kazdym binie liczba zliczen jest suma z histogramow tablica[poz1] i tablica[poz2]
}
```

W naszej klasie możemy mieć wiele specjalizowanych metod (również wiele specjalizowanych metod o tej samej nazwie – dla różnych obiektów). Kompilator w trakcie kompilowania programu sam zdecyduje, którą metodę ma użyć

Uwaga!
W przypadku klas szablonowych wszystko (zarówno definicję klasy jak i metody klasy) piszemy tylko w pliku **.h** (czyli najczęściej dla jednej klasy jeden plik .h) – nie możemy pisać definicji metod w pliku **.cpp**!

W języku C++ oprócz klas szablonowych możemy również tworzyć globalne funkcje szablonowe. Np. załóżmy, że mamy kilka klas, które mają taką samą metodę (dajmy na to `int GetRozmiar()`). Chcielibyśmy stworzyć funkcję, która porównuje rozmiar dwóch obiektów dowolnej z tych klas i zwraca większą wartość. Nic trudnego, możemy dostawić więcej typów danych:

```
template<class T1, class T2>
int CompareRozmiar(T1 obiekt1, T2 obiekt2)
{
	if(obiekt1.GetRozmiar()>obiekt2.GetRozmiar())
		return obiekt1.GetRozmiar();
	else
```

```
        return obiekt2.GetRozmiar();
}
```

Oczywiście możemy dodawać więcej typów klas w nagłówku `template`, możemy je też dowolnie nazywać (np. `template<class A, class B, class C, class D>` itp.).

Zadanie do wykonania

Część I – klasa Histogram

1. Generujemy dużą próbkę liczb pseudolosowych z wybranych rozkładów (Gauss, Uniform czy Poisson) i zapisujemy wynik do pliku, np. **dane.Gauss.txt**
2. Tworzymy szablonową klasę `Histogram`. Zakładamy, że możemy przechowywać typu `double` oraz `int`. Następnie tworzymy plik z funkcją **main**, w którym będziemy otwierać zapisany wcześniej plik wpisywać dane do histogramu. Na końcu wypiszemy histogram na ekran. Nazwę wczytywanego pliku podajemy jako 1 parametr uruchomienia programu. Na koniec, aby przećwiczyć dziedziczenie i abstrakcyjne typy danych (ATD), należy również stworzyć abstrakcyjną klasę bazową `THistogram` dla klasy szablonowej `Histogram`.
3. Na koniec przekierowujemy „output” uruchomionego programu do pliku **out.txt**.
   3 p.

Część II

Modyfikujemy nasz program tak (a najlepiej robimy kopię działającej klasy i ją modyfikujemy), by używał dynamicznej struktury danych `std::vector` (która pełni rolę dynamicznej tablicy) ze standardowej biblioteki szablonów C++ (STL – Standard Template Library). Obok listy jest to druga bardzo ważna struktura danych, którą należy znać.

Zauważmy, że użycie STL powoduje, że nie musimy „wymyślać koła od nowa”, tylko używamy już gotowych i zoptymalizowanych struktur danych i algorytmów. W naszym przykładzie nie musimy się zatem martwić alokacją i zwalnianiem pamięci oraz niepotrzebne nam już będzie dodatkowe pole przechowujące ilość binów, gdyż to dostarcza nam „automatycznie” klasa `std::vector`.

2 p.

Przydatne linki:
http://www.cplusplus.com/reference/vector/vector/
http://pl.wikibooks.org/wiki/C++/Vector