Programowanie II R

Zadania – seria 11.

Szablony i koncepty.

Zadanie 1. `demere` – Paradoks kawalera de Méré.

Paradoks kawalera de Méré to problem z dziedziny rachunku prawdopodobieństwa postawiony przez francuskiego matematyka–amatora Antoine'a Gombauda (1607 – 1684). Dotyczy on serii rzutów symetrycznymi sześciennymi kostkami do gry i polega na rozstrzygnięciu, które spośród następujących dwóch zdarzeń jest bardziej prawdopodobne:

- zdarzenie A: wyrzucenie przynajmniej jednej szóstki w czterech rzutach jedną kostką,
- zdarzenie B: wyrzucenie przynajmniej jednej pary szóstek w dwudziestu czterech rzutach dwiema kostkami.

Analityczne rozwiązanie tego problemu podał Blaise Pascal (1623 – 1662).

Napisz szablon funkcji

```
template<typename RNG>
void DeMere(double &pA, double &pB, unsigned int N)
```

która posługując się metodą Monte Carlo znajduje przybliżone prawdopodobieństwa opisanych wyżej zdarzeń A i B. Funkcja `DeMere` przyjmuje argumenty `pA` i `pB` stanowiące referencje do zmiennych typu `double` oraz argument `N` odpowiadający liczbie losowych punktów do wykorzystania. Funkcja ta powinna przypisać obliczone prawdopodobieństwa zdarzeń A i B do zmiennych, odpowiednio, `pA` i `pB`. Parametr szablonu `RNG` ma reprezentować generator liczb losowych, który funkcja wykorzysta do wylosowania punktów.

Umieść koncept `std::uniform_random_bit_generator` zdefiniowany w pliku nagłówkowym `random` na liście wymagań szablonu `DeMere` względem typu `RNG`, aby upewnić się, że typ ten reprezentuje generator liczb losowych.

Korzystając z tego szablonu napisz program `demere`, który wypisuje na standardowe wyjście prawdopodobieństwa zdarzeń A i B znalezione przez Ciebie w literaturze oraz ich przybliżone wartości obliczone przez funkcję `DeMere` z wykorzystaniem następujących generatorów liczb losowych dostarczanych przez plik nagłówkowy `random`:

- `minstd_rand` – liniowy generator kongruentny *MINSTD*,
- `mt19937` – 32-bitowy generator *Mersenne Twister*,
- `ranlux24` – 24-bitowy generator *RANLUX*.

Zadanie 2. `loops` – Pętle w liście jednokierunkowej.

Napisz szablon klasy

```
template <typename T>
class LinkedList
```

reprezentującej listę jednokierunkową elementów typu `T`. Szablon powinien definiować wewnętrzną strukturę `Node` reprezentującą element listy, zawierającą pola określające jego wartość oraz wskaźnik na kolejny element listy. Ponadto szablon `LinkedList` powinien zawierać publiczne pole `Head` będące wskaźnikiem na czołowy element listy oraz publiczną metodę `Push` umieszczającą na liście nowy element.

Wykorzystując mechanizm konceptów zawęź zbiór dopuszczalnych typów `T` do typów liczbowych.

Następnie napisz szablon funkcji

```
template <typename T>
bool HasLoop(LinkedList<T> list)
```

zwracającej wartość `true`, gdy lista zawiera pętlę (tzn. gdy przynajmniej jeden z jej elementów pojawia się na niej dwa razy – należy tu jednak brać pod uwagę nie samą wartość elementu, ale również jego adres), oraz wartość `false` w przeciwnym przypadku.

Korzystając z tych szablonów napisz program `loops`, w którym stwórz dwie listy - jedną zawierającą pętlę i jedną bez pętli – a następnie przetestuj na nich działanie napisanych przez siebie szablonów.

Opracowanie: Bartłomiej Zglinicki.