Programowanie II R

Zadania – seria 3.

Tablice statyczne i dynamiczne, wczytywanie plików

Zadanie 0: Anatomia pamięci (Wskaźniki i Adresy)

1. Stwórz zmienną całkowitą `x` oraz wskaźnik do zmiennej całkowitej (zmienną typu adres int) `ptr`. W zmiennej `ptr` zapisz adres zmiennej `x` (używając operatora `&`).
2. Wypisz na ekran następujące wartości i porównaj je ze sobą:
   - Wartość zmiennej `x`.
   - Adres zmiennej `x`.
   - Wartość przechowywaną we wskaźniku `ptr`.
   - Adres samego wskaźnika `ptr`. Czy jest on taki sam jak adres `x`?
3. Zmień wartość zmiennej `x` na `100`, ale zrób to wyłącznie za pomocą wskaźnika `ptr` (używając operatora `*`). Wypisz `x`, aby potwierdzić zmianę.
4. Stwórz referencję do zmiennej `x`. Wypisz jej adres i porównaj go z adresem `x`.

Dane wejściowe

Plik `random_walk_Step1e3_Ens1e4.txt` zawiera liczby całkowite rozdzielone spacjami. Każda liczba reprezentuje końcowe położenie cząstki po wykonaniu $N_{step} = 1e3$ kroków błądzenia losowego (wynik pojedynczej próby). Błądzenie odbywa się w jednym wymiarze, położenie początkowe $x = 0$, możliwe przesunięcia to ± 1 (z równym prawdopodobieństwem). Liczba prób: 1e4.

Zadanie 1: Pamięć statyczna i arytmetyka wskaźników

Wczytaj pierwsze 100 pomiarów z pliku do tablicy statycznej `int dane[100]`.

1. Podczas wczytywania danych z pliku, przypisuj wartości do komórek używając wyłącznie **arytmetyki wskaźników** (np. `ptr+1` to wskaźnik na kolejną zmienną w pamięci za `ptr`).
2. Po wczytaniu, wyświetl zawartość tablicy na ekranie, korzystając dla odmiany z operatora indeksowania `[]`.
3. Oblicz: wartość minimalną, maksymalną, średnią arytmetyczną oraz odchylenie standardowe populacji:

$$\sigma = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n}}$$

Wskazówki: Do wczytania danych z pliku użyj obiektu `std::fstream` (plik nagłówkowy `fstream`) i operatora `>>`, analogicznie do wczytywania danych ze standardowego wejścia (`std::cin`).

Uwagi: `dane[i]` to dokładnie to samo co `*(dane + i)`.

Zadanie 2: Tablice dynamiczne (`new` / `delete`)

Napisz analogiczny program do tego z Zadania 1. Użytkownik powinien móc zdecydować, ile prób (n) chce przeanalizować.

1. Pobierz od użytkownika liczbę n.
2. Alokuj pamięć dynamicznie używając operatora `new`.
3. Wczytaj n wartości z pliku i powtórz obliczenia statystyczne z Zadania 1. Porównaj wyniki z oczekiwanymi (średnia 0 i odchylenie standardowe $\sqrt{N_{step}}$) dla rosnącej liczby prób.
4. Na końcu programu zwolnij pamięć używając `delete[]`.

Wskazówki: Operator `new int[n]` alokuje ciągły obszar pamięci i zwraca wskaźnik na jej początek.

Uwagi: Brak zwolnienia pamięci (memory leak) może prowadzić do zużycia całej pamięci. Aby się przed tym zabezpieczyć, dobrze jest zastąpić zwykły wskaźnik jednym ze sprytnych wskaźników (smart pointer), np. `std::unique_ptr<int[]>`, które posprzątają pamięć bez wywoływania delete.

Zadanie 3: Bezpieczne kontenery statyczne (`std::array`)

Przepisz Zadanie 1, wykorzystując kontener `std::array<int, 100>`. Wykorzystaj pętle typu *range-based for*, np:

- `for (int x: dane)`
- `for (int &x: dane)`
- `for (auto x: dane)`
- `for (auto &x: dane)`

Wskazówki: Dołącz nagłówek `array`.

Uwagi: W większości zastosowań, `std::array` jest preferowane zamiast "C-style array" (np. `int dane[100];`)
.

Zadanie 4: Kontenery dynamiczne (`std::vector`)

Przepisz zadanie 2 przy użyciu `std::vector<int>`. Do dodania wartości na końcu wektora użyj metody `.push_back()`.

Wskazówki: Metody są to funkcje zdefiniowane dla danej klasy (typu zmiennej). Przykładowo, mając zmienną `std::vector<int> vec;` możemy wywołać metodę `vec.push_back(10)`.

Uwagi:

- W większości zastosowań, `std::vector` jest preferowane zamiast dynamicznej alokacji operatorem `new`.
- Wywołanie `push_back` może skutkować skopiowaniem całego wektora, aby zapewnić jego ciągłość w pamięci. Aby tego uniknąć można z góry zarezerwować miejsce w pamięci metoda `reserve`.

Opracowanie: Piotr Dziekan