Sortowanie. Wskaźniki funkcyjne. Złożoność algorytmów

- a % b to reszta z dzielenia a przez b,
- int rand() zwraca pseudolosową liczbę całkowitą,
- time(NULL) zwraca liczbę sekund jakie upłynęły od 1 stycznia 1970 roku,
- srand(int seed) inicjalizuje generator liczb pseudolosowych,
- clock() może być użyte do mierzenia upływu czasu.

```
clock_t start = clock();
//tutaj jakiś algorytm, np. sortowanie
clock_t finish = clock();
double  duration = (double)(finish - start) / CLOCKS_PER_SEC;
cout << "Czas wyniosl " << duration << endl;
```

- swap(a, b) służy do zamiany zawartości zmiennych:

```
inline void swap(int &a, int &b) {
     int tmp = a;
     a = b;
     b = tmp;
}
```

Zadania

1. Niech każde uruchomienie aplikacji wyświetla losowy napis postaci "Czesc! Mam na imie XXX i chce opowiedziec o swoim problemie". W miejsce XXX wstaw wylosowane imię (spośród kilkunastu zdefiniowanych).
2. Dla pewnego N (które będzie można później zmieniać) utwórz N-elementową tablicę liczb typu int, a następnie nadaj im losowe wartości początkowe z zakresu [100, 999]. W tym celu zdefiniuj pomocniczą funkcję `void losuj(int *t, int N, int low, int high)`, przy czym low i high to odpowiednio parametry określające dolne i górny zakres losowanych liczb.

   Wskazówka: aby wylosować liczbę z zakresu 0..99 należałoby napisać: `rand() % 100;` natomiast liczbę 10..99 uzyskasz jako `10 + rand() % 90;`
3. Wyświetl 10 pierwszych i 10 ostatnich elementów tej tablicy.
4. Zaimplementuj algorytm sortowania bąbelkowego w postaci funkcji `void bubble_sort(int *t, int N)`. Wewnętrzna pętla tego algorytmu może wyglądać następująco:

```
for(int j=0; j < N-1; j++)
{
     if(t[j] > t[j+1])
     {
          swap(t[j], t[j+1]);
```

```
        }
    }
```

5. Zmierz czas wykonania algorytmu dla różnych wartości N i zbadaj charakter tej zależności (złożoność obliczeniową algorytmu).

Użycie qsort i wskaźniki funkcyjne

Nagłówek funkcji qsort:

```
void qsort(
  void *base, //gdzie w pamięci znajduje się tablica do posortowania
  size_t num, //ile elementów zawiera ta tablica
  size_t width, //ile bajtów zajmuje pojednyczny element
  int (*compare)(const void *a, const void *b) //funkcja porównująca
);
```

Zwróć uwagę na to, że jednym z argumentów funkcji qsort jest inna funkcja (a dokładniej wskaźnik funkcyjny **compare**). Taki trik pozwala na napiasanie jednego, uniwersalnego algorytmu sortowania, przy czym kluczową jego część, jaką jest funkcja porównująca elementy, dostarczamy z zewnątrz. Funkcja qsort ma dzięki temu możliwość wywołania dostarczonej przez nas funkcji i podejmowanie decyzji o tym, czy elementy mają być przestawione, czy nie.

```
int compare_int( const void *ptra, const void *ptrb )
{
     int a = * (int*)ptra;
     int b = * (int*)ptrb;
     return a<b ? -1 : (a>b) ? 1 : 0;
}
const int N = 100;
int t[N];
losuj(t, N, 100, 999);
qsort(t, N, sizeof(t[0]), compare_int);
```

Zadania

1. W jaki sposób są sortowane liczby w tablicy po wykonaniu powyższego kodu – rosnąco czy malejąco?
2. Zmień porządek sortowania na odwrotny, modyfikując funkcję compare_int(…).
3. * Posortuj liczby w taki sposób, że wcześniej w tablicy znajdą się wszystkie liczby parzyste, a dopiero po nich wszystkie nieparzyste. Np. 2, 6, 10, 100, 3, 7, 17, 105
4. Dostosuj funkcję porównującą i pozostałe elementy programu w taki sposób, żeby dawało się posortować struktury danych struct Student { int nralb; char imie[30]; }; Studenci będą umieszczeni w 100-elementowej tablicy, której wartości początkowe są losowe imiona (lub losowe znaki zamiast imion) oraz losowe liczby albumów.