Programowanie i metody numeryczne

Zadania – seria 2.

Język C++: wskaźniki, referencje, tablice, wektory.

Zadanie 1. `polar` – Zamiana współrzędnych biegunowych na kartezjańskie.

Rozważmy na płaszczyźnie dwa układy współrzędnych: układ współrzędnych kartezjańskich OXY oraz układ współrzędnych biegunowych z biegunem O i osią biegunową OX. Współrzędne kartezjańskie $(x, y) \in \mathbb{R}^2$ punktu P o współrzędnych biegunowych $(r, \varphi) \in \mathbb{R} \times [0, 2\pi)$ dane są wzorami

$$\begin{cases} x = r\cos\varphi, \\ y = r\sin\varphi. \end{cases}$$

Napisz funkcję

```
void Polar(double r, double phi, double *x, double &y)
```

przeliczającą współrzędne biegunowe zadanego punktu płaszczyzny na jego współrzędne kartezjańskie. Funkcja ta przyjmuje cztery argumenty: dwie zmienne zmiennoprzecinkowe odpowiadające kolejno współrzędnym r i φ, wskaźnik do zmiennej zmiennoprzecinkowej odpowiadającej współrzędnej x oraz referencję zmiennej zmiennoprzecinkowej odpowiadającej współrzędnej y. Zadaniem funkcji jest umieszczenie w zmiennych wskazywanych przez dwa ostatnie argumenty współrzędnych kartezjańskich punktu o współrzędnych biegunowych określonych dwoma pierwszymi argumentami.

Korzystając z tej funkcji napisz program `polar`, który wczytuje ze standardowego wejścia współrzędne biegunowe pewnego punktu płaszczyzny z kątem wyrażonym w stopniach, a następnie wypisuje na standardowe wyjście współrzędne kartezjańskie tego punktu.

Zadanie 2. `threestat` – Statystyka zbioru trzech liczb.

Napisz następujące przeciążone funkcje:

- ```
  void Ord(double &a, double &b, double &c)
  void Ord(double *a, double *b, double *c)
  ```

  Funkcje te przyjmują jako argumenty trzy liczby zmiennoprzecinkowe, przekazane, odpowiednio, przez referencje i przez wskaźniki, a następnie zamieniają ich wartości w taki sposób, by były one posortowane rosnąco (tzn. wartość drugiego argumentu ma być większa od wartości pierwszego, a wartość trzeciego – od wartości drugiego).

- ```
  void MaxMin(double &a, double &b, double &c, double *&pMax, double *&pMin)
  void MaxMin(double *a, double *b, double *c, double **pMax, double **pMin)
  ```

  Funkcje te przyjmują jako argumenty trzy liczby zmiennoprzecinkowe oraz dwa wskaźniki, przekazane, odpowiednio, przez referencje i przez wskaźniki, a następnie wpisują do wskaźników `pMax` i `pMin` adresy tych z przekazanych zmiennych, które mają, odpowiednio, największą i najmniejsza wartość.

- ```
  double& Biggest(double &a, double &b, double &c)
  double* Biggest(double *a, double *b, double *c)
  ```
  ```

Funkcje te przyjmują jako argumenty trzy zmienne zmiennoprzecinkowe, przekazane, odpowiednio, przez referencje i przez wskaźniki, oraz zwracają, odpowiednio, referencję lub wskaźnik do tej z nich, która ma największą wartość.

Korzystając z tych funkcji napisz program `threestat`, który wczytuje ze standardowego wejścia trzy liczby zmiennoprzecinkowe, a następnie dwukrotnie wypisuje je na standardowe wyjście w kolejności rosnącej, największą i najmniejszą z nich oraz ponownie te liczby, zastępując jednak największą z nich liczb sumą wszystkich trzech liczb. Za każdym razem program powinien posłużyć się inną spośród przeciążonych funkcji.

Zadanie 3. `arrays` – Operacje na tablicach i wektorach.

Napisz następujące przeciążone funkcje operujące na tablicach oraz wektorach:

- ```
  bool ArrIsDiff(const int data[], size_t size)
  bool ArrIsDiff(const std::vector<int> &data)
  ```

  Funkcja powinna zwracać wartość `true`, jeśli wszystkie elementy kolekcji `data` są różne lub wartość `false` w przeciwnym przypadku.

- ```
  int ArrNumDiff(const int data[], size_t size)
  int ArrNumDiff(const std::vector<int> &data)
  ```

  Funkcja powinna zwracać liczbę różnych elementów kolekcji `data` (np. 2 w przypadku kolekcji 1, 5, 1, 1, 5).

- ```
  int* ArrDiffOnly(const int data[], size_t size)
  std::vector<int> ArrDiffOnly(const std::vector<int> &data)
  ```

  Funkcja powinna zwracać kolekcję elementów złożoną z różnych elementów kolekcji `data` (np. 1, 5 w przypadku kolekcji 1, 5, 1, 1, 5).

Korzystając z tych funkcji napisz program `arrays`, który powinien prosić użytkownika o podanie liczby całkowitej n, następnie o podanie n liczb całkowitych, utworzyć tablicę i wektor wypełnione tymi liczbami oraz wypisać wynik działania każdej z napisanych funkcji na tej tablicy i na tym wektorze.

Zadanie 4. `bubblesort` – Sortowanie bąbelkowe.

Sortowanie bąbelkowe to jeden z algorytmów sortowania kolekcji danych. Polega on na wielokrotnym przechodzeniu przez kolekcję i porównywaniu dwóch stojących obok siebie elementów – gdy są one ustawione w złej kolejności, zostają zamienione miejscami. Proces sortowania kończy się, gdy podczas przejścia przez kolekcję nie dokonano ani jednej zamiany.

Napisz funkcję `bubble_sort` przyjmującą jako argument wektor elementów typu `double` i zwracającą wektor złożony z posortowanych w kolejności rosnącej elementów wyjściowego wektora. Funkcje powinna wykorzystywać algorytm sortowania bąbelkowego.

Korzystając z tej funkcji, napisz program `bubblesort`, który wczytuje ze standardowego wejścia oddzielone spacjami liczby całkowite i wypisuje na standardowe wyjście te same liczby w kolejności rosnącej.

Zadanie 5. `selectionsort` – Sortowanie przez wybieranie.

Sortowanie przez wybieranie (ang. *selection sort*) to jeden z najprostszych znanych algorytmów sortowania. W celu posortowania pewnej kolekcji elementów w kolejności rosnącej metodą przez wybieranie należy najpierw odszukać najmniejszy element tej kolekcji i zamienić go miejscami z jej pierwszym elementem – element najmniejszy znajdzie się wówczas na swojej docelowej pozycji. Czynności te należy następnie powtórzyć dla pozostałych elementów należących do sortowanej kolekcji: szukamy najmniejszego z nich i zamieniamy go
  ```

miejscami z elementem na drugiej pozycji – w ten sposób dwa pierwsze elementy kolekcji będą na właściwych pozycjach. Procedurę tę należy powtarzać tak długo, aż wszystkie elementy kolekcji znajdą się na swoich miejscach (czyli – innymi słowy – aż zbiór elementów wciąż wymagających posortowania stanie się jednoelementowy).

Napisz funkcję

```
void SelectionSort(std::vector<int> &data)
```

która przyjmuje jako argument referencję wektora liczb całkowitych, a następnie sortuje ten wektor rosnąco, posługując się algorytmem sortowania przez wybieranie.

Korzystając z tej funkcji napisz program `selectionsort`, który wczytuje ze standardowego wejścia liczby całkowite do napotkania znaku końca pliku, a następnie wypisuje te liczby na standardowe wyjście w kolejności rosnącej.

Zadanie 6. `binsearch` – Wyszukiwanie binarne.

Wyszukiwanie binarne to jeden z algorytmów wyszukiwania elementu w kolekcji danych, oparty na metodzie *divide et impera* (łac. *dziel i rządź*). Można go stosować wyłącznie dla posortowanych kolekcji. Polega on w uproszczeniu na dzieleniu kolekcji na coraz mniejsze przedziały tak długo, aż długość przedziału będzie równa 1 – wówczas pojedynczym sprawdzeniem można ustalić, czy szukany element należy do konkretnego przedziału. Algorytm ten można realizować rekurencyjnie.

Napisz funkcję `bin_search` przyjmującą trzy argumenty: pierwszym ma być poszukiwany element, drugim – wektor elementów typu `int`, zaś trzecim i czwartym – liczby całkowite `min` i `max`. Funkcja powinna posortować wektor, a następnie zwrócić indeks, pod którym szukany element występuje w tym wektorze, biorąc pod uwagę tylko indeksy od `min` do `max` włącznie. Jeśli szukany element występuje wielokrotnie, funkcja może zwrócić dowolny z jego indeksów, jeśli zaś nie występuje wcale, funkcja powinna zwrócić `-1`.

Korzystając z tej funkcji, napisz program `binsearch`, który przyjmuje jako argument wywołania jedną liczbę całkowitą oraz wczytuje ze standardowego wejścia oddzielone spacjami liczby całkowite i wypisuje na standardowe wyjście położenie przekazanej jako argument wywołania liczby wśród posortowanych rosnąco liczb wczytanych ze standardowego wejścia.

Opracowanie: Bartłomiej Zglinicki.