Metody sortowania: sortowanie przez wstawianie

Na dzisiejszych zajęciach zajmiemy się jednym z najprostszych algorytmów sortowania. Sortowanie jest jedną z najważniejszych i najbardziej podstawowych operacji wykonywanych na danych. Sortowanie dotyczy nie tylko zagadnień informatyki; spotykamy je w życiu codziennym. Porządkowanie list nazwisk w kolejności alfabetycznej, układanie kart w talii, porządkowanie spisu czynności do wykonania względem terminu ich ukończenia to tylko kilka przykładów.

Rozważmy następującą sytuację: gracz trzyma w dłoni karty do gry i chce je uporządkować. Załóżmy dla uproszczenia rozważań, że są to karty w jednym kolorze: *9, 10, J, Q, K, A* (ćwierć talii 24 kart). Jest to sytuacja dogodna do omówienia algorytmu sortowania, znanego jako tzw. „sortowanie przez wstawianie“ (ang. *insertion sort*). Nasz gracz otrzymał karty w następującej kolejności:

10	J	K	9	A	Q

Sortowanie takiego pliku kart omawianą metodą polega na tym, że gracz stopniowo dokłada karty do uporządkowanej części kart (początkowo zawierającej jedną kartę) spośród pozostałych kart przez wstawianie, po jednej, kolejnych kart w odpowiednie miejsce. Spójrzmy na karty i rozważmy *pierwsze dwie* (karty będą sortowane w porządku rosnącym):

10	J	K	9	A	Q

Widzimy, że te dwie karty są ustawione w prawidłowej kolejności – walet znajduje się za mniejszą od niego dziesiątką (ich pozycja w całym rozdaniu będzie musiała ulec zmianie – przed tymi kartami będzie *„9“* - ale w tej chwili wystarczające jest jedynie utrzymanie właściwego porządku w rozważanym podzbiorze kart). Dalszy ciąg procesu sortowania opiera się na dołączaniu kolejnych kart do uporządkowanego zbioru. Dołóżmy zatem do naszego zestawu dwóch uporządkowanych kart kolejną kartę:

10	J	K	9	A	Q

Znowu mamy sytuację, w której trzy karty, znajdujące się w części uporządkowanej, występują we właściwej kolejności. Dołączmy kolejną kartę.

10	J	K	9	A	Q

Po raz pierwszy natrafiamy na nietrywialny przypadek dołączania karty do zbioru uporządkowanego: musimy przesunąć wszystkie karty o wartości wyższej niż *„9“* (czyli karta właśnie dołączana – ostatnia) o jedną pozycję w prawo, a następnie w powstałe wolne miejsce wstawić nową kartę. W rozważanej sytuacji oznacza to wykonanie następujących czynności:

- Wyjęcie z talii karty „9”, następnie ustawienie kart w poniższym porządku:

puste	10	J	K	A	Q

- Wstawienie karty „9” w „puste“ miejsce, co kończy proces dołączania karty do części uporządkowanej.

9	10	J	K	A	Q

Dołączenie kolejnej karty do zbioru uporządkowanego

9	10	J	K	A	Q

jest kolejnym przypadkiem trywialnym, gdyż as jest na prawidłowym, co do kolejności, miejscu

w zbiorze kart uporządkowanych. Do zakończenia zadania pozostało nam dołączenie ostatniej karty – damy.

9	10	J	K	A	Q

Zgodnie z podaną wcześniej zasadą, należy przesunąć o jedną pozycję w prawo karty większe od damy, stojące na ostatnich pozycjach; króla i asa

9	10	J	puste	K	A

i wstawienie w zwolnione miejsce wyjętej z talii damy

9	10	J	Q	K	A

W ten sposób problem posortowania kart został rozwiązany. Podsumujmy założenia algorytmu „sortowanie przez wstawianie”:

- sortowane elementy *dzielą się* na dwie grupy: *uporządkowaną* i *nieuporządkowaną*.
- w toku sortowania do grupy elementów posortowanych, zawierającej na początku jeden element, dołączane są po jednym, elementy z grupy nieuporządkowanej.
- nowo dodawany element ma być wstawiony w odpowiednie miejsce w grupie uporządkowanej. Grupa uporządkowana jest rozszerzana o kolejny element w następujący sposób: dodawany element zajmuje puste miejsce, pozostawione przez blok elementów większych od niego (przy sortowaniu rosnącym), po ich przesunięciu o jedną pozycję w prawo.
- algorytm kończy się, gdy dokonaliśmy dołączenia wszystkich elementów grupy nieuporządkowanej do grupy elementów uporządkowanych.

Przykładowa implementacja algorytmu „insertion sort” dla sortowania rosnącego tablicy znakowej wygląda następująco (plik *i-sort.cpp*):

```
#include<iostream>
using namespace std;

void insertion_sort(char tablica[], unsigned n);
void wypisz(char tablica[], unsigned n);

int main()
{
char litery[]={'C', 'H' ,'R', 'I', 'S', 'T', 'M', 'A', 'S'};
unsigned i, ilosc = sizeof(litery)/sizeof(char);

   cout<<"Program posortuje tablice z literkami metoda \"przez wstawianie\""<<endl;
   cout<<"W tablicy mamy "<<ilosc<<" liter, sa to:"<<endl;
   wypisz(litery, ilosc);
   cout<<endl;

   cout<<"Po sortowaniu tablica zawiera litery w nastepujacej kolejnosci:"<<endl;
   insertion_sort(litery, ilosc);
   wypisz(litery, ilosc);
}

void insertion_sort(char tablica[], unsigned n)
{
unsigned i;

   for(i=1; i<n; i++)
   {
   // Granica fragmentu tablicy, ktory porzadkujemy
```

```
   unsigned j=i;
   // Zapamietaj element "krancowy"
   char kopia = tablica[j];

                // Wewnatrz fragmentu tablicy przesun elementy "za duze" o 1 pozycje
                while( (j>0) && ( tablica[j-1] > kopia))
                {
                   // Przesun element j-1 tablicy o jeden indeks wyzej (do j)
                   tablica[j] = tablica[j-1];
                   // Zaktualizuj indeks w tablicy
                   j--;
                }
                // Wpisz element "krancowy" w odpowiednie miejsce
                tablica[j] = kopia;
   }
}

void wypisz(char tablica[], unsigned n)
{
unsigned i;

   // Wypisujemy znak po znaku - mamy w tablicy pojedyncze litery, a nie lancuch !!!
   for(i=0; i<n; i++) cout<< tablica[i]<<" ";
   cout<<endl;
}
```

Program w trakcie wykonania wypisze na ekran, co następuje:

```
Program posortuje tablice z literkami metoda "przez wstawianie"
W tablicy mamy 9 liter, sa to:
C H R I S T M A S

Po sortowaniu tablica zawiera litery w nastepujacej kolejnosci:
A C H I M R S S T
```

Tablica *litery[]* zawiera znaki do posortowania. Jak wynika z listy inicjalizacyjnej, jest to tablica znaków, nie posiadająca znaku *NULL* na końcu tekstu. Jest to dopuszczalne, ale należy pamiętać, że nie wolno w takim wypadku traktować tekstu, zawartego w tablicy jako łańcucha znaków! Oznacza to na przykład, że instrukcja *cout<<litery;* spowodowałaby niepoprawne działanie programu, jakkolwiek instrukcja *cout<<litery[0];* działałaby poprawnie. Na czym polega tak znaczna różnica? Pierwsza instrukcja oznacza żądanie wypisania łańcucha znaków, ale ponieważ tablica *litery[]* nie zawiera znaku *NULL,* strumień będzie wypisywał wszystko, nie zatrzymując się na końcu tekstu, gdyż nie ma prawidłowego znaku końca. Druga instrukcja interpretowana jest jako rozkaz wypisania pojedynczego elementu tablicy, co nie spowoduje błędu – zostanie wypisany jeden znak.

Ostatnim elementem, wymagającym komentarza jest instrukcja:

```
ilosc = sizeof(litery)/sizeof(char);
```

Aby wyznaczyć ilość elementów tablicy, można podzielić ilość bajtów, jaką zajmuje tablica (*sizeof (litery)*), przez ilość bajtów, jaką zajmuje element tablicy (*sizeof (char)*). Metody tej można użyć jedynie w zakresie ważności danej tablicy, w szczególności nie zadziała ona dla argumentu tablicowego funkcji. Ponadto, *sizeof(char)* wynosi jeden i może zostać pominięte w wyrażeniu.

Zadanie

Państwa zadaniem jest napisanie programu, który posortuje w porządku malejącym tablicę 10 liczb, podanych z klawiatury przez użytkownika.