Scalanie uporządkowanych

LABORKA © Piotr Ciskowski

- **Napisz funkcję o nazwie Scalaj – scalającą dwa ciągi liczb, o których wiemy, że każdy z nich jest posortowany rosnąco, w trzeci ciąg – tak, aby ten trzeci też był posortowany - jednak funkcja ma SCALAĆ, a nie SORTOWAĆ**

- **Funkcja przyjmuje trzy tablice liczb rzeczywistych (dwie wejściowe i jedna wynikowa) oraz trzy liczby całkowite („ilości” liczb w poszczególnych tablicach – rozmiary tablic) - i tylko scala, niczego nie wyświetla**

zadanie jedyne. scalanie uporządkowanych

- **Opracuj dwie wersje funkcji Scalaj:**

  1. sprawdzającą najpierw if-em,
     czy może wystarczy tylko dołączyć pierwszy ciąg do drugiego, tudzież nazad,
     a jeśli trzeba
     – scalającą ciągi for-em wykorzystującym if-a i skaczącym po obu ciągach
     (for w zasadzie idzie spokojnie po ciągu wynikowym,
     a skacze po ciągach scalanych ;-)

  2. na for'ze i while'u – skaczącą po obu ciągach
     (for skacze jakby po pierwszym ciągu,
     a while po drugim ;-)

- **Chyba dobrym pomysłem jest to sobie narysować ;-)**

- **Opracuj dwie wersje funkcji Scalaj:**

  1. „przechodzimy się” po obu tablicach źródłowych
     - wpisujemy mniejszy element do wynikowej
     - któraś tablica źródłowa zawsze wcześniej się skończy,
     - więc doklejamy do wynikowej resztę tej drugiej

  2. „przechodzimy się” po pierwszej tablicy
     - wpisujemy element do wynikowej,
     - ale przedtem…

- **Chyba dobrym pomysłem jest to sobie narysować ;-)**

- **Dopisz program testujący tę funkcję**
- **W trakcie testowania możesz wykorzystać zdefiniowane w programie ciągi, np. takie (przykład na liczbach całkowitych, zapisany MATLABowo):**

```
ciagA = [ 1 , 4 , 5 , 8 , 9 ]
ciagB = [ 1 , 2 , 3 , 4 , 5 , 6 ] ;
```

- **Dla powyższych ciągów program powinien zwrócić wynik:**

```
ciagC = [ 1 , 1 , 2 , 3 , 4 , 4 , 5 , 5 , 6 , 8 , 9 ]
```

No a później program powinien oczywiście pytać się o ciągi do scalenia

zadanie jedyne. scalanie uporządkowanych

- **wejście:**

```
ciagA = [ 1 , 4 , 5 , 8 , 9 ]
ciagB = [ 1 , 2 , 3 , 4 , 5 , 6 ] ;
```

- **wyjście:**

```
ciagC = [ 1
```

zadanie jedyne. scalanie uporządkowanych

- **losowe dane:**

ciągA:	**1**	**3**	**7**	**12**	**15**	**18**	**20**	**20**	**25**	
ciągB:	**4**	**7**	**11**	**15**	**19**	**22**	**25**	**29**	**30**	

- **schemat blokowy – wersja pierwsza:**

- **wersja pierwsza – str. 1:**

```
/* Scalanie dwóch uporządkowanych rosnąco ciągów liczb
   w trzeci też uporządkowany rosnąco (bez sortowania) */

// Zarówno program, jak i funkcje, zakładają numerację elementów macierzy od 1
// - po prostu pomijają pierwszy element

#include <iostream.h>
#include <conio.h>

void Scalanie ( int n , int m , int &nm, float pierwszy[] , float drugi[],
                                         float scalony[] )
{
  int i,j,k;
  if ( pierwszy[n] < drugi[1] )
  {
    for ( i=1 ; i<=n ; i++ ) scalony[i]   = pierwszy[i] ;
    for ( i=1 ; i<=m ; i++ ) scalony[i+n] = drugi[i]    ;  return ;
  }
  if ( drugi[m] < pierwszy[1] )
  {
    for ( i=1 ; i<=m ; i++ ) scalony[i]   = drugi[i]    ;
    for ( i=1 ; i<=n ; i++ ) scalony[i+m] = pierwszy[i] ;  return ;
  }
```

- **wersja pierwsza – str. 2:**

```
  nm = n+m ;

  for ( i=1,j=1,k=1 ; k<=nm ; k++ )
  { if ( pierwszy[i] < drugi[j] )
    { scalony[k] = pierwszy[i] ;
      if ( i==n )
      { nm=n  ;
        break ;
      }
      i++;
    }
    else
    { scalony[k] = drugi[j] ;
      if ( j==m )
      { nm=m  ;
        break ;
      }
      j++;
    }
  }
  if ( nm==n ) for ( ; j<=m ; j++ ) scalony[++k] = drugi[j]    ;
  if ( nm==m ) for ( ; i<=n ; i++ ) scalony[++k] = pierwszy[i] ;

  nm = n + m ;
} // koniec funkcji
```

zadanie jedyne. scalanie uporządkowanych
- rozwiązanie

- **wersja pierwsza – str. 3:**

```
void main()
{
  float pierwszy[100], drugi[100], scalony[200];
  int n,m,nm,i;
  clrscr();

  cout << "Podaj liczbę elementów pierwszego ciągu: ";  cin >> n;
  for (i=1; i<=n; i++)
  { cout << i << ": ";   cin >> pierwszy[i]; }
  cout << "\n Podaj liczbę elementów drugiego ciągu: "; cin >> m;
  for (i=1; i<=m; i++)
  { cout << i << ": ";   cin >> drugi[i];    }

  Scalanie (n,m,,nm,pierwszy,drugi,scalony);

  cout << "\n Ciąg scalony: " << endl;
  for(i=1; i<=nm; i++)
    cout << "scalony[" << i << "]= " << scalony[i] << endl;
  getch();
}
```

- **schemat blokowy – wersja druga:**

- **wersja druga:**