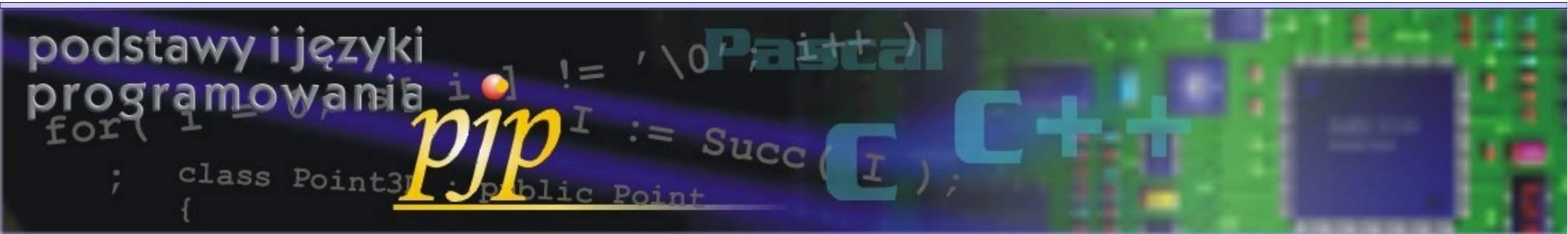
Języki programowania obiektowego
Nieobiektowe elementy języka C++

Roman Simiński
roman.siminski@us.edu.pl
www.programowanie.siminskionline.pl

Tablice — koncepcja, deklaracje, podstawowe zastosowania

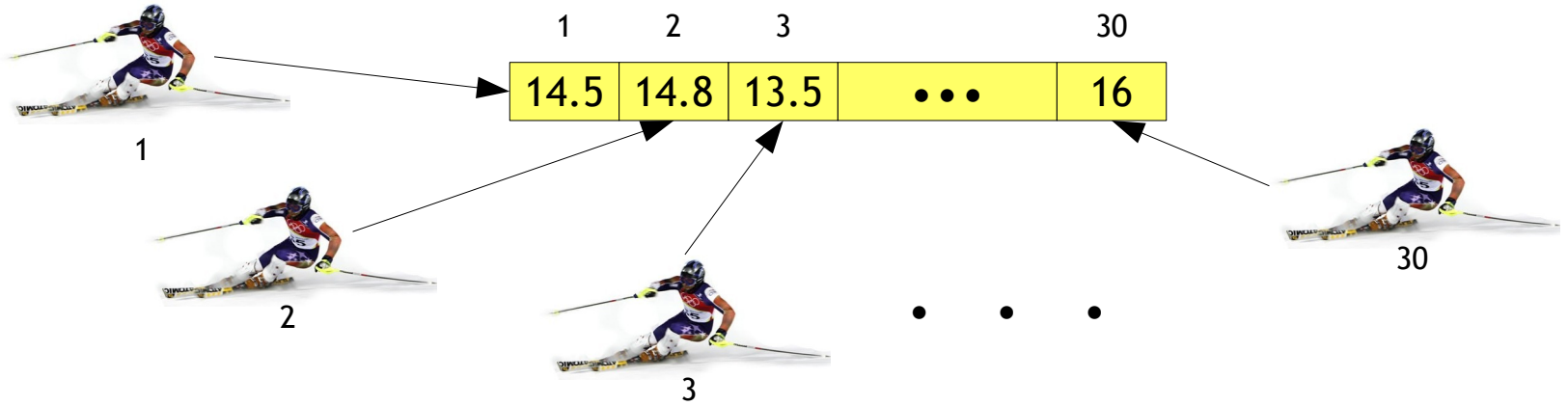


Tablice — koncepcja

- *Tablica* jest *zmienną* złożoną z elementów *tego samego typu*.
- Obejmuje ona *ciągły obszar pamięci* operacyjnej dokładnie tak duży, aby zmieścić wszystkie jej elementy.
- Termin *tablica* w języku potocznym jest zmiennikiem sformułowania *zmienna tablicowa*.

Tablice stosuje się wtedy, gdy trzeba zgromadzić wiele obiektów tego samego typu w jednym miejscu, i w sposób wygodny przetwarzać je, według jednolitego schematu.

Przykład — czasy przejazdów 30 zawodników startujących w slalomie można zapamiętać w tablicy.



Tablice — uwaga na standardy C89 i C99

Według standardu C89 i C++:

- tablica zawsze składa się z *ustalonej*, i *znanej na etapie kompilacji* liczby elementów,
- liczba elementów tablicy *nie ulega zmianie* w trakcie działania programu — tablice są *statyczne*.

W standardzie C99 istnieją tablice *VLA* (ang. *variable length array*):

- liczba elementów tablicy może być *zdefiniowany w trakcie wykonania programu* — może być określona *wartością zmiennej*, ta wartość *nie musi być znana* na etapie kompilacji,
- liczba elementów tablicy *nie ulega zmianie* w trakcie działania programu — raz stworzona tablica *zachowuje swój rozmiar*.

Chociaż w językach C/C++ nie występuje mechanizm *tablic dynamicznych*, istnieją jednak mechanizmy pozwalające na łatwą implementacje tablic posiadających właściwości dynamiczne.

Deklaracja zmiennych tablicowych

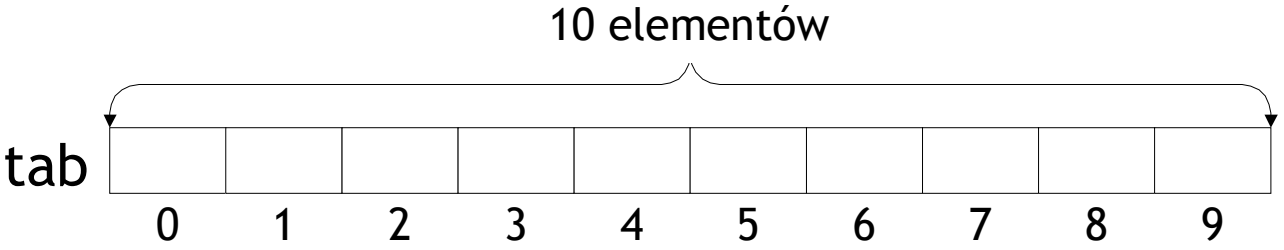
Ogólna postać deklaracji tablicy — zmiennej tablicowej:

```
<typ_elemetu> nazwa_zmiennej_tablicowej[ <wyrażenie_ stałe> ]
```

wyrażenie_ stałe — wyrażenie określające liczbę elementów tablicy, wartość tego wyrażenia musi być znana na etapie kompilacji (za wyjątkiem standardu C99).

Deklaracja 10-cio elementowej tablicy liczb całkowitych:

```
int tab[ 10 ];
```



Elementy tablicy numerowane są zawsze od 0. Zatem jeżeli N oznacza liczbę elementów tablicy, to ostatni jej element ma numer $N - 1$.

Deklaracja zmiennych tablicowych, parametryzacja rozmiaru

Liczba elementów tablicy (rozmiar) określa się zwykle używając stałych:

```
#define N 10
. . .
int tab[ N ];
```

C/C++

```
const int N = 10;
. . .
int tab[ N ];
```

C++

Parametryzacja liczby elementów tablicy pozwala na łatwiejszą modyfikację liczby przetwarzanych elementów.

Wykorzystanie modyfikatora *const*:

- Kwalifikator typu *const* może wystąpić z każdą specyfikacją typu. Zmienna z *const* powinna być zainicjowana ale potem nie może zmieniać wartości.
- Zmienna z kwalifikatorem *const* w języku C *nie jest traktowana jako wartość stała* i *nie może* być wykorzystywana do określania rozmiaru tablicy.
- Zmienna z kwalifikatorem *const* w języku C++ *może* być wykorzystywana do określania rozmiaru tablicy.

Deklaracja zmiennych tablicowych, modyfikator const

Dziesięcioelementowa tablica liczb całkowitych Różne warianty deklaracji	C	C++
`int tab[ 10 ];`	poprawne	poprawne

`#define N 10`
`. . .`
`int tab[ N ];`
	poprawne	poprawne

`const int N = 10;`
`. . .`
`int tab[ N ]`
	niepoprawne	poprawne

Preferowane postacie definicji zmiennych tablicowych:

C , ANSI89

```
#define MAKS_DL 80
char bufor[ MAKS_DL ];

#define LB_MIES 12
double dochody[ LB_MIES ];
```

C++

```
const int MAKS_DL = 80;
char bufor[ MAKS_DL ];

const int LB_MIES = 12;
double dochody[ LB_MIES ];
```

Dowoływanie się do elementów tablicy

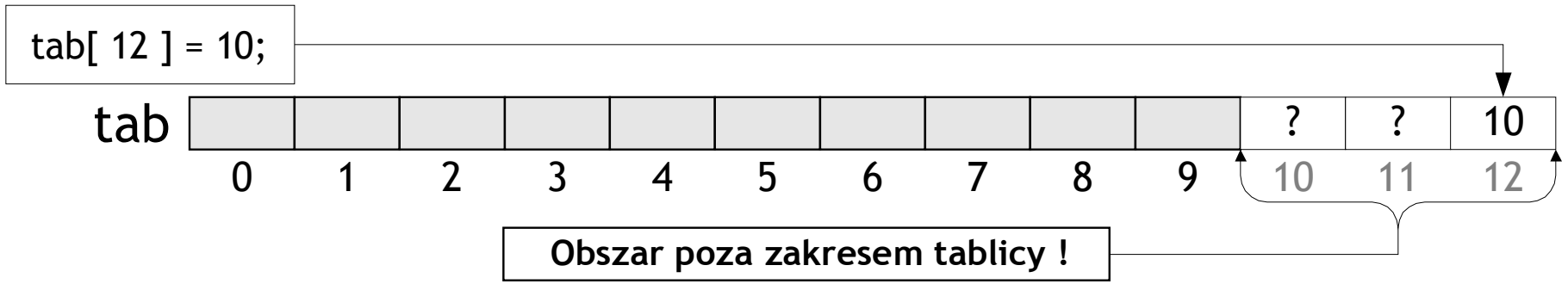
```
tab[ 0 ] = 1;
```

```
tab[ N - 1 ] = 5;
```

```
a = 2 * tab[ 3 ];
```

```
int i = 0, j = N – 1;
a = tab[ i ] + tab [ j ];
```

W języku C i C++ nie ma żadnych wbudowanych mechanizmów zabezpieczających przed odwoływaniem się do „elementów” leżących poza zakresem indeksowym tablic!



Tablice wolno inicjalizować na etapie deklaracji

- Jeżeli inicjalizowana tablica *nie posiada określonego rozmiaru*, zostanie on określony na podstawie *liczby elementów inicjalizujących*.
- Jeżeli liczba wartości początkowych jest *mniejsza od rozmiaru tablicy*, to elementy o brakujących wartościach początkowych otrzymują *wartość zero* (zmienne zewnętrzne, statyczne i automatyczne).
- Podanie *zbyt wielu* wartości początkowych jest *błędem*.
- W C89 i C++ nie ma sposobu na zainicjowanie *środkowego elementu* bez podania wszystkich wartości pośrednich (w C99 można).

```
int tab[ 10 ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

```
int dni_miesiecy[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

```
float przychody[ 12 ] = { 0, 0, 0 }; // Za mało wartości początkowych
```

Tablice wolno inicjalizować na etapie deklaracji

- Inicjalizacja wybranych elementów tablicy w C99:

```
int tab[ 10 ] = { [ 3 ] = 10, 20, 30, [ 4 ] = 50 } ;
```

```
00: 0
01: 0
02: 0
03: 10
04: 50
05: 30
06: 0
07: 0
08: 0
09: 0
```

Zainicjuj zgodnie z podanym indeksem.
Niezainicjowanym przypisz zero.
Zapamiętaj ostatnią inicjalizację.

- Jak wyznaczyć liczbę elementów tablicy gdy zdefiniowano ją bez rozmiaru?

```
int dni_miesiecy[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

```
int liczba_miesiecy = sizeof( dni_miesiecy ) / sizeof( int );
```

lub:

```
int liczba_miesiecy = sizeof( dni_miesiecy ) / sizeof( dni_miesiecy[ 0 ] );
```

Typowe operacje na tablicach

Przetwarzanie tablic realizowane jest zwykle z wykorzystaniem instrukcji iteracyjnych. Do przetwarzania tablic najczęściej wykorzystuje się iterację *for*.

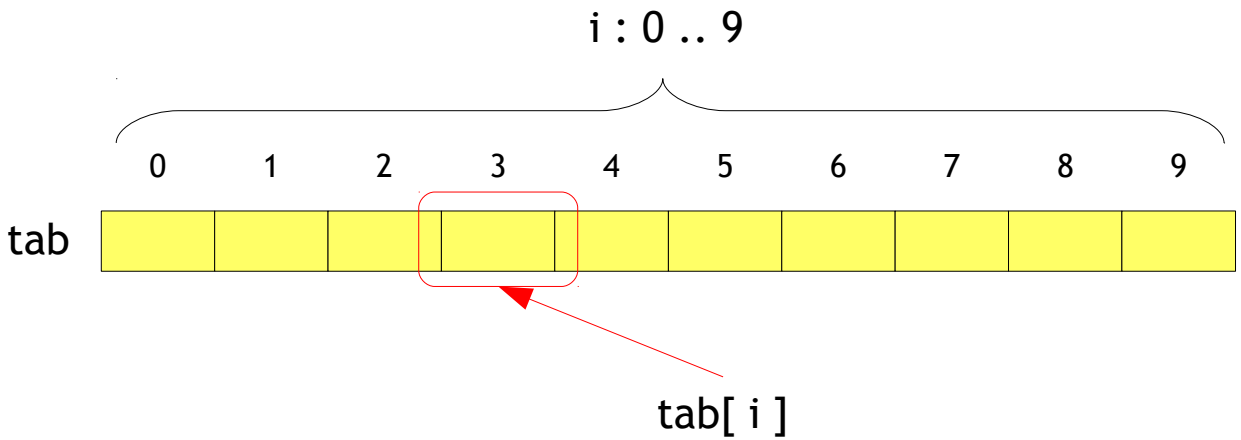
C , ANSI89

```
#define N 10
. . .
int tab[ N ];
int i;
```

C++

```
const int N = 10;
. . .
int tab[ N ];
int i;
```

Dla każdej wartości *i* z zakresu 0 .. *N - 1*
  Wykonaj przetwarzanie *i*-tego elementu tablicy *tab* : *tab*[*i*]



Ogólny schemat iteracyjnego przetwarzania tablic

```
for( i = 0; i < LB_ELEM_TABLICY; ++i )
  Jakaś operacja na elemencie i-tym tablicy t: t[ i ]
```

- Do iteracyjnego przetwarzania tablic potrzebna jest *zmienna indeksowa*. Jest to zmienna całkowita, nazwa jest dowolna, najczęściej stosuje się jednak zmienne o nazwach *i*, *j*, *k*, *l*, *m*, *n*, … .
- Najlepiej gdy rozmiar tablicy jest określony nazwaną stałą — wtedy łatwiej jest modyfikować rozmiar tablicy bez potencjalnie wielu zmian w programie.
- Zmienna indeksowa w C++ może być deklarowana w obrębie iteracji for, uwaga wg aktualnego standardu nie jest wtedy dostępna poza iteracją:

```
for( int i = 0; i < LB_ELEM_TABLICY; ++i )
  Jakaś operacja na elemencie i-tym tablicy t: t[ i ]

i = 0; // zmienna i nie jest dostępna w tym zasięgu!
```

Typowe operacje na tablicach

Ustawianie wartości wszystkich elementów tablicy, np. zerowanie:

```
for( i = 0; i < N; i++ )
  tab[ i ] = 0;
```

Wersja 1-sza

Operator postinkrementacji pozwala napisać to krócej:

```
for( i = 0; i < N; tab[ i++ ] = 0 )
  ;
```

Wersja 2-ga

Ogólnie, wypełnianie pewnym wzorcem *wzorzec*:

```
int wzorzec = -1;
. . .
for( i = 0; i < N; tab[ i++ ] = wzorzec )
  ;
```

Typowe operacje na tablicach, cd. ...

Gdy kolejność wykonywania operacji na tablicy nie ma znaczenia:

```
i = N;
while( --i >= 0 )
  tab[ i ] = 0;
```

lub z wykorzystaniem iteracji *for*:

```
for( i = N; --i >= 0 ; tab[ i ] = 0 )
   ;
```

Na marginesie, uwaga na takie konstrukcje:

```
x = tab[ ++i ] + tab[ i ];
```

```
tab[ ++i ] = a * ++i;
```

Typowe operacje na tablicach, cd. ...

Wczytywanie danych ze strumienia wejściowego programu do tablicy:

```
for( i = 0; i < N; i++ )
{
  cout << endl << '>';
  cin >> tab[ i ];
}
```

Wyprowadzanie danych z tablicy do strumienia wyjściowego programu:

```
for( i = 0; i < N; i++ )
  cout << endl << tab[ i ];
```

Wersja uproszczona:

```
for( i = 0; i < N; cout << endl << tab[ i++ ] )
  ;
```

Typowe operacje na tablicach, cd. ...

Sumowanie liczb zapisanych w tablicy:

```
int suma = 0;
. . .
for( i = 0; i < N; i++ )
  suma += tab[ i ];
```

Wersja uproszczona:

```
int suma;
. . .
for( i = 0, suma = 0; i < N; suma += tab[ i++ ] )
  ;
```

Przykładowe „dziwactwo”: wyznaczanie sumy co drugiego, dodatniego elementu tablicy, podzielnego przez 3:

```
int suma;
. . .
for( i = 0, suma = 0; i < N; i += 2 )
  if( tab[ i ] > 0  )
    if( tab[ i ] % 3 == 0 )
      suma += tab[ i ];
```

Kopiowanie zawartości tablic

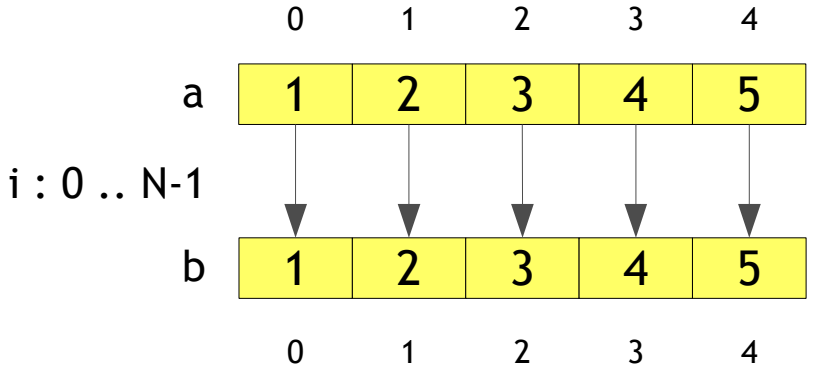
```
const int N = 5;

int a[ N ] = { 1, 2, 3, 4, 5 };
int b[ N ];
. . .
int i;
```

Tak w języku C/C++ nie wolno:

```
b = a;   // Nie wolno przypisywać do siebie tablic
```

Trzeba przepisać wartości każdego z elementów:

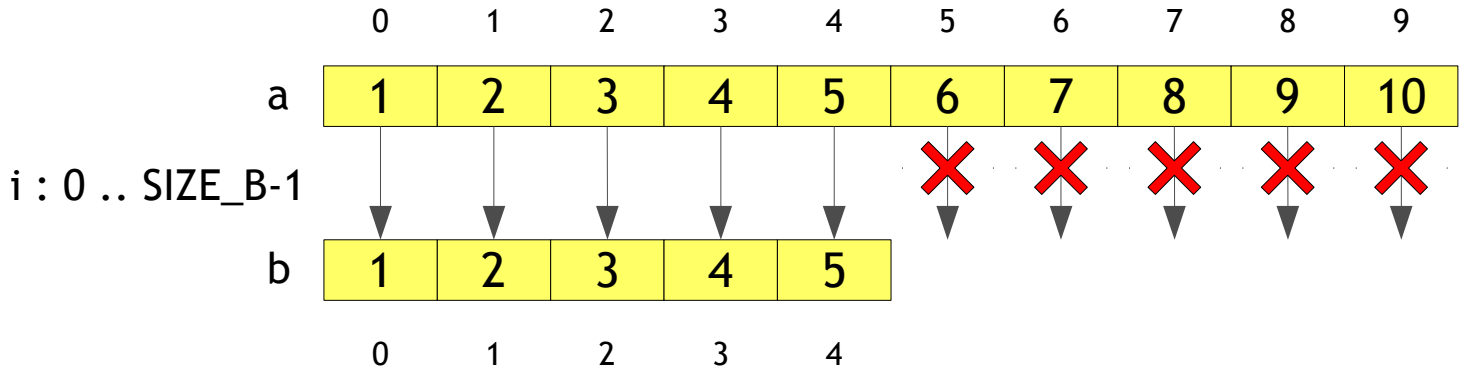
```
for( i = 0; i < N; i++ )
  b[ i ] = a[ i ];
```



Kopiowanie zawartości tablic

Uwaga na niejednakowe rozmiary tablic:

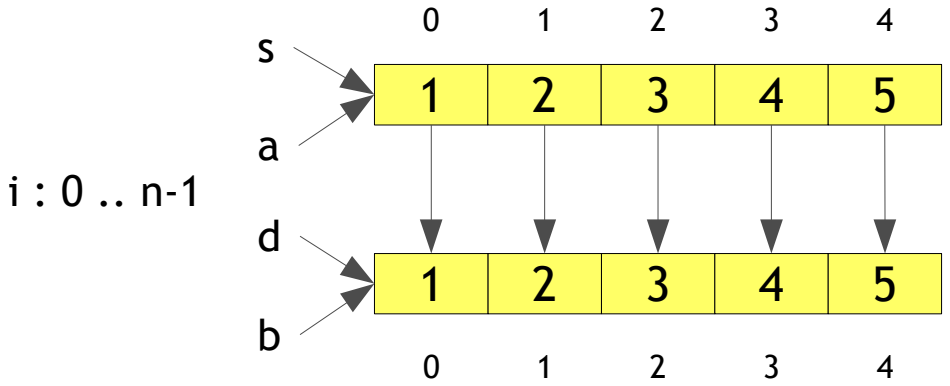
```
const int SIZE_A = 10;
const int SIZE_B = 5;

int a[ SIZE_A ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int b[ SIZE_B ];

for( i = 0; i < SIZE_B; i++ )
  b[ i ] = a[ i ];
```



Kopiowanie zawartości tablic, parametry tablicowe

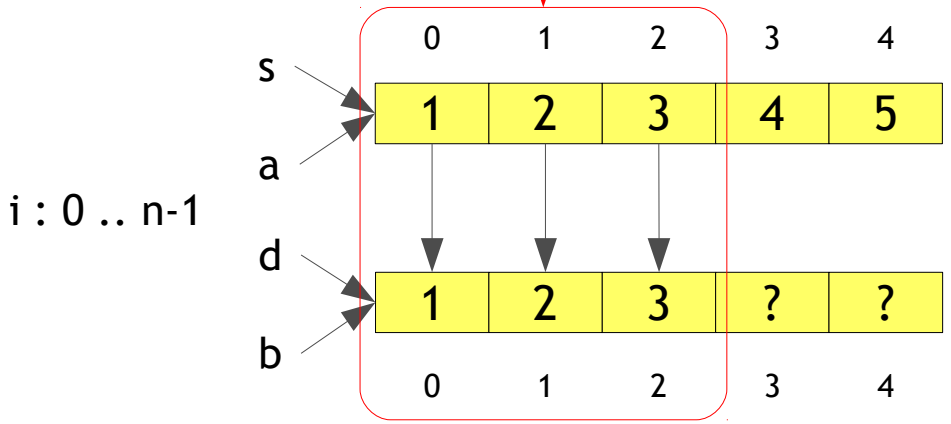
```
const int N = 5;

int a[ N ] = { 1, 2, 3, 4, 5 };
int b[ N ];
. . .
copy_int_table( b, a, 5 ); // Kopiuj z a do b 5-ęć elementów
```

```
// Przekopiuj z tablicy źródłowej s (ang. source) do tablicy docelowej
// d (ang. destination) nie więcej niż n elementów.
void copy_int_table( int d[], int s[], int n )
{
  int i = 0;
  for( i = 0; i < n; i++ )
    d[ i ] = s[ i ];
}
```



Uwaga, funkcja może operować tylko na części tablicy

```
const int N = 5;

int a[ N ] = { 1, 2, 3, 4, 5 };
int b[ N ];
. . .
copy_int_table( b, a, 3 ); // Kopiuj z a do b tylko 3 elementy
```



Kopiowanie zawartości tablic, parametry tablicowe

Krótsza wersja kopiowania tablic z wykorzystaniem iteracji *while*:

```
void copy_int_table( int d[], int s[], int n )
{
  while( --n >= 0 )
    d[ n ] = s[ n ];
}
```

- W języku C nazwy tablic są traktowane w specyficzny sposób. O tym już niedługo.
- Z tego powody wolno definiować tablicowe parametry formalne *bez rozmiaru*.
- Taki parametr przyjmuje do siebie tablicę o *dowolnym rozmiarze*.
- Tablice *pozornie* zachowują się tak, jakby były przekazywane *przez zmienną*.

Kopiowanie zawartości tablic, rozwiązania alteratywne

Ponieważ z definicji tablice to spójne obszary pamięci operacyjnej, można do ich kopiowania użyć funkcji *memmove* lub *memcpy* (nagłówek *mem.h*, zgodne z ANSI C):

```
memmove( b, a, N * sizeof( int ) );
```

Lub sprytniej:

```
memmove( b, a, N * sizeof( b[ 0 ] ) );
```

`memmove( dest, src, n );`

Kopiuje blok *n* bajtów z lokalizacji *src* do *dest*. Lokalizacje te mogą się nakładać.

Wykorzystanie funkcji typu *memmove* jest skuteczne dla tablic, których elementami są dane prostych typów wbudowanych w język (*char*, *int*, *float*, *double*). W przypadku tablic struktur i czy obiektów takie kopiowanie może być bardzo niebezpieczne.

Kopiowanie zawartości tablic, rozwiązania alteratywne

```
memcpy( b, a, N * sizeof( int ) );
```

Lub sprytniej:

```
memcpy( b, a, N * sizeof( b[ 0 ] ) );
```

`memcpy( dest, src, n );`

Kopiuje blok *n* bajtów z lokalizacji *src* do *dest*. Gdy lokalizacje te się nakładają, działanie funkcji jest niezdefiniowane.

Na marginesie, alternatywa dla iteracyjnego zerowania tablicy, Można do tego wykorzystać funkcję *memset* :

```
memset( b, 0, N * sizeof( b[ 0 ] ) );
```

`memset( s, c, n );`

Wypełnia *n* pierwszych bajtów obszaru *s* bajtem o wartości *c*.

Problem do rozwiązania — statystyka czasów przejazdu slalomu

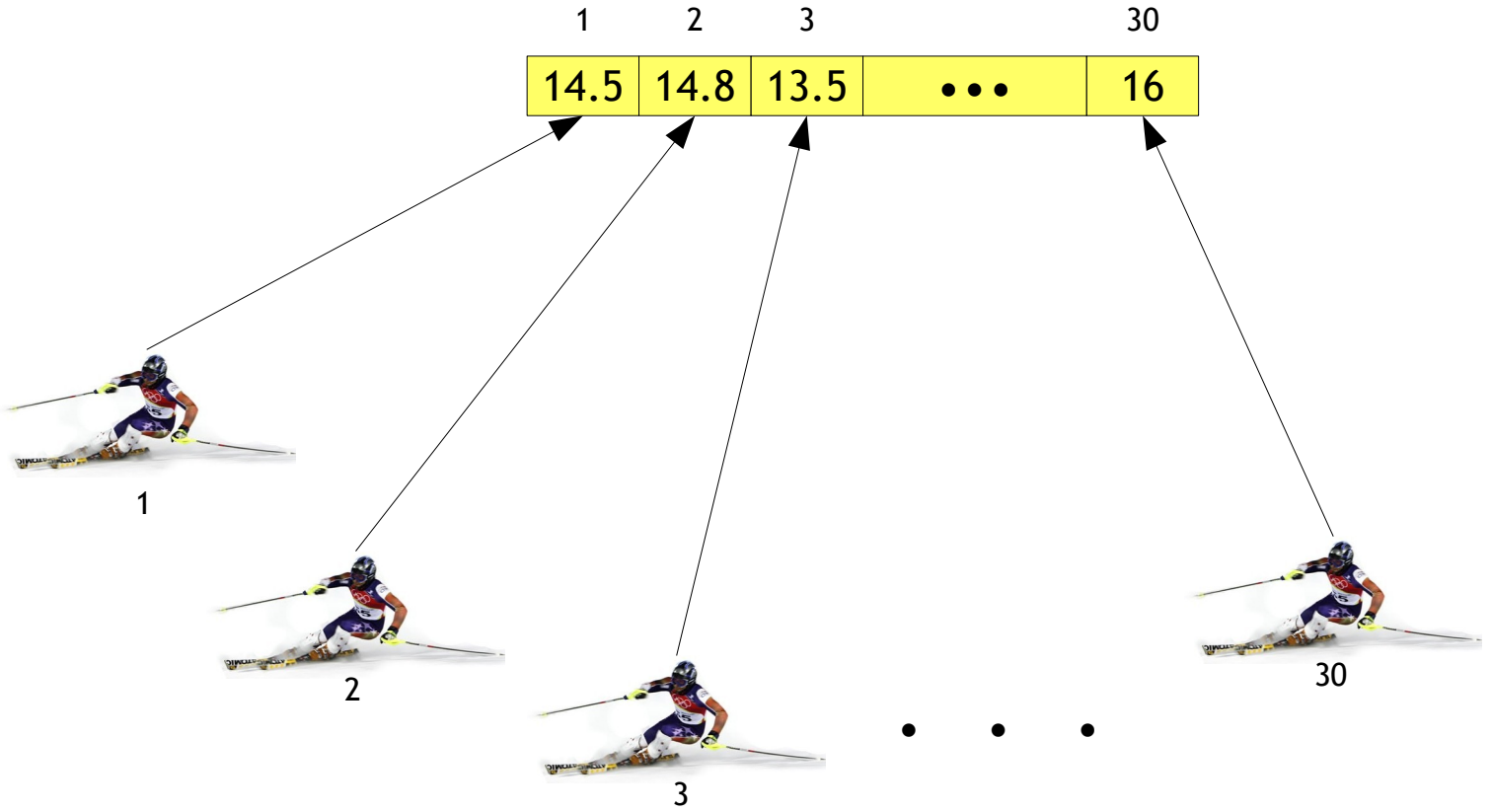
Napisać program rejestrujący czasy przejazdu zawodników startujących w slalomie.

Wymagania dla programu:

- Liczba zawodników — zwykle 30.
- Wpisywanie danych na bieżąco, wyniki po wpisaniu wszystkich czasów.
- Wyświetlenie czasów *uporządkowanych rosnąco* — od najlepszego do najgorszego.
- Wyznaczenie i wyświetlenie statystyki czasów: czasu *średniego*, wartości *środkowej* (mediany), *wariancji* i *odchylenia standardowego*.

Analiza problemu — co mamy na wejściu?

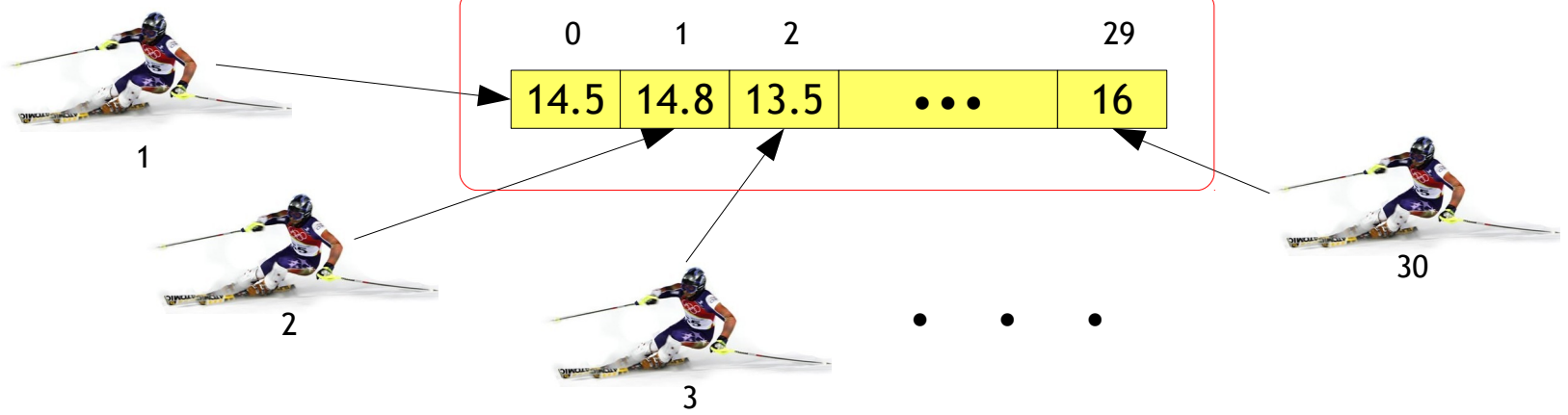
Na wejściu mamy 30-ci liczb rzeczywistych — każdy z nich to czas przejazdu odpowiedniego zawodnika.



Analiza problemu — jak zapamiętać dane wejściowe?

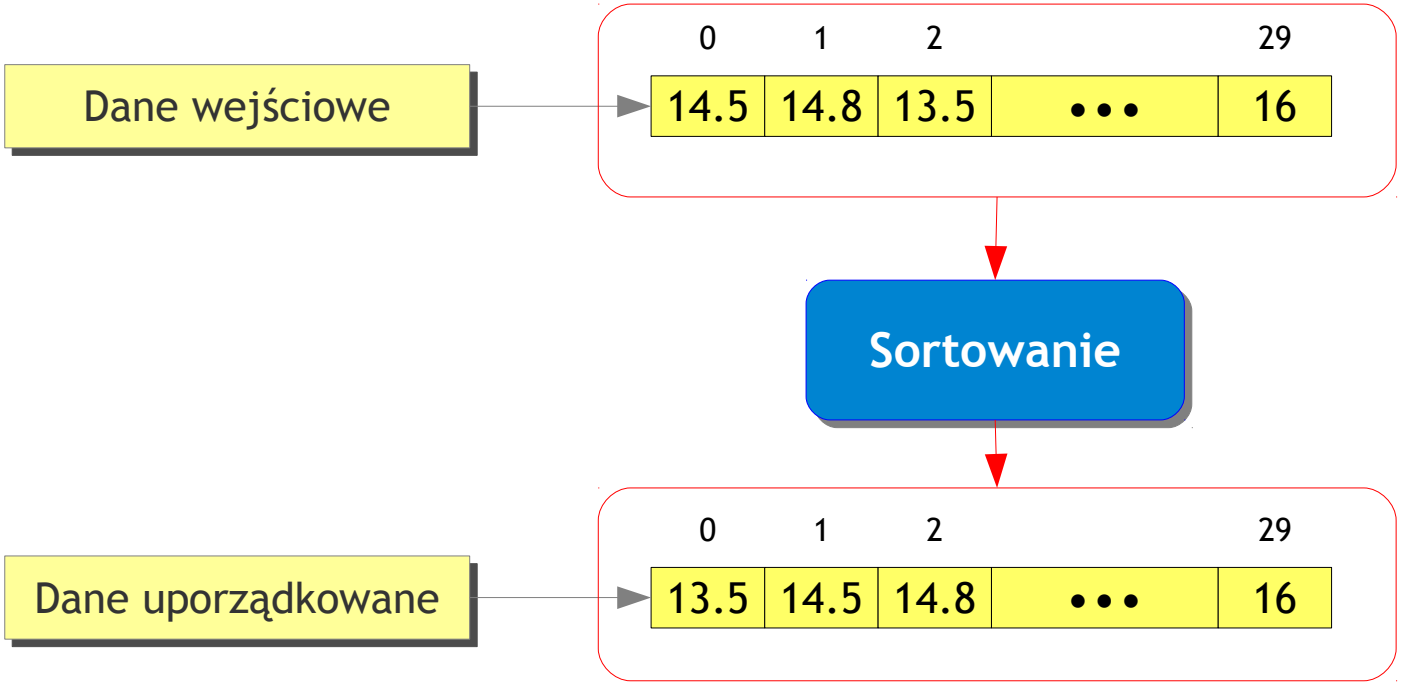
Czasy przejazdów 30 zawodników startujących w slalomie można zapamiętać w 30-to elementowej tablicy.

```
// Liczba zawodnikow
const int LB_ZAWOD = 30;

// Tablica do przechowywania czasow
double czasy[ LB_ZAWOD ];
```



Analiza problemu — czasy należy uporządkować narastająco

Wystarczy wykorzystać dowolny z algorytmów sortowania — dla 30 elementów może to być jakikolwiek z prostych algorytmów sortowania.



Uwaga! Aby można było posortować dane, trzeba je po wczytaniu zmagazynować w pamięci operacyjne — najlepiej w *tablicy*. Algorytmów sortowania tablic jest wiele.

Analiza problemu — statystyka czasów, średnia i mediana

Czasy tworzą ciąg 30-stu danych liczbowych. Załóżmy, że tych liczb jest n.

Możemy wtedy powiedzieć, że mamy n danych liczbowych $a_1, a_2, a_3, \ldots, a_n$.

Średnia arytmetyczna $a_{śr}$ to:

$$a_{śr} = \frac{a_1 + a_2 + a_3 + \ldots + a_n}{n}$$

Mediana uporządkowanego rosnąco ciągu n liczb $a_1 \leq a_2 \leq a_3 \leq \ldots \leq a_n$:

- to środkowy wyraz ciągu (dla n nieparzystego),
- to średnia arytmetyczna środkowych wyrazów ciągu (dla n parzystego).

Analiza problemu — statystyka czasów, wariancja i odchylenie

Wariancja n danych liczbowych $a_1, a_2, a_3, \ldots, a_n$ o średniej arytmetycznej $a_{śr}$ to:

$$\sigma^2 = \frac{(a_1 - a_{śr})^2 + (a_2 - a_{śr})^2 + (a_3 - a_{śr})^2 + \ldots (a_n - a_{śr})^2}{n}$$

Odchylenie standardowe σ to pierwiastek kwadratowy z *wariancji*:

$$\sigma = \sqrt{\sigma^2}$$

Analiza problemu — statystyka czasów, po co to wszystko?

- *Wartość średniej arytmetycznej* nie zawsze dobrze opisuje zbiór danych — dane *odstające* mocno zaburzają wartość średniej.
- *Mediana* może lepiej przybliżać informację o typowych czasach przejazdów uzyskiwanych przez *większość zawodników*.

Przykład:

Czasy przejazdów: 1 2 2 3 3 3 3 4 30 40 $a_{śr} = 9.1$ $mediana = 3$

Analiza problemu — statystyka czasów, po co to wszystko?

- *Wariancja* i *odchylenie standardowe* powie jak czasy przejazdów są rozrzucone wokół czasu średniego.
- *Mniejsza wartość* odchylenia standardowego oznacza, że *więcej czasów* przejazdów jest *blisko średniej*.
- *Większa wartość* odchylenia standardowego oznacza, że rozkład czasów jest bardziej równomierny — jest więcej czasów bliskich wartości skrajnych.

Przykład:

Czasy przejazdów: 1 2 2 3 3 3 3 4 4 5 $a_{śr} = 3$ $\sigma \approx 1.1$

Czasy przejazdów: 1 1 1 2 3 3 4 5 5 5 $a_{śr} = 3$ $\sigma \approx 1.6$

Czas na program, wersja najprostsza

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

int main()
{
  const int LB_ZAWOD = 30;        // Liczba zawodnikow
  double czasy[ LB_ZAWOD ];       // Tablica do przechowywania czasow

  double suma, srednia, mediana; // Zmienne dla statystyki czasow
  double wariancja, odchylenie;

  int    nr_min, roboczy;         // Zmienne dla algorytmu sortowania
  double min;

  int    i;                       // Zmienna robocza iteracji

  // Wprowadzenie wartosci poszczegolnych czasow i zapisanie w tablicy
  cout << "Podaj czasy przejazdow kolejnych zawodnikow:" << endl;
  for( i = 0; i < LB_ZAWOD; i++ )
  {
    cout << i + 1 << ": ";
    cin >> czasy[ i ];
  }
```

1/4

Czas na program, wersja najprostsza

```
// Wyznaczenie wartosci sredniej wprowadzonych czasow
for( i = 0, suma = 0; i < LB_ZAWOD; suma += czasy[ i++ ] )
  ;

// Sortowanie przez proste wybieranie
for( roboczy = 0; roboczy < LB_ZAWOD - 1; roboczy++ )
{
  // Ustalamy robocze minimum
  min = czasy[ roboczy ];
  nr_min = roboczy;

  // Czy wsrod kolejnych elementow jest mniejszy?
  for( i = roboczy + 1; i < LB_ZAWOD; i++ )
    if( czasy[ i ] < min )
    {
      min = czasy[ i ]; // Zapamietaj mniejszy
      nr_min = i;       // Zapamietaj nr mniejszego
    }

  // Zamien mniejszy z roboczym
  czasy[ nr_min ] = czasy[ roboczy ];
  czasy[ roboczy ] = min;
}
```

2/4

Czas na program, wersja najprostsza

```
// Wyznaczenie sredniej i mediany
srednia = suma / LB_ZAWOD;
if( LB_ZAWOD % 2 != 0 )
  mediana = czasy[ LB_ZAWOD / 2 ]; // Liczba czasów jest nieparzysta
else
  mediana = ( czasy[ LB_ZAWOD / 2 - 1 ] + czasy[ LB_ZAWOD / 2 ] ) / 2;

// Wyznaczanie wariancji i odchylenia standartowego. Ponownie
// wykorzystujemy zmienna suma -- teraz do wyznaczenia sumy kwadratów
// roznic od wartosci sredniej
for( i = 0, suma = 0; i < LB_ZAWOD; i++ )
  suma += pow( czasy[ i ] - srednia, 2 ); // Podnies do drugiej potegi

wariancja = suma / LB_ZAWOD;
odchylenie = sqrt( wariancja );

// Wyprowadzenie zawartosci tablicy do strumienia wyjsciowego
cout << "\nCzasy uporzadkowane rosnaco:";
for( i = 0; i < LB_ZAWOD; i++ )
  cout << endl << czasy[ i ];
```

3/4

Czas na program, wersja najprostsza

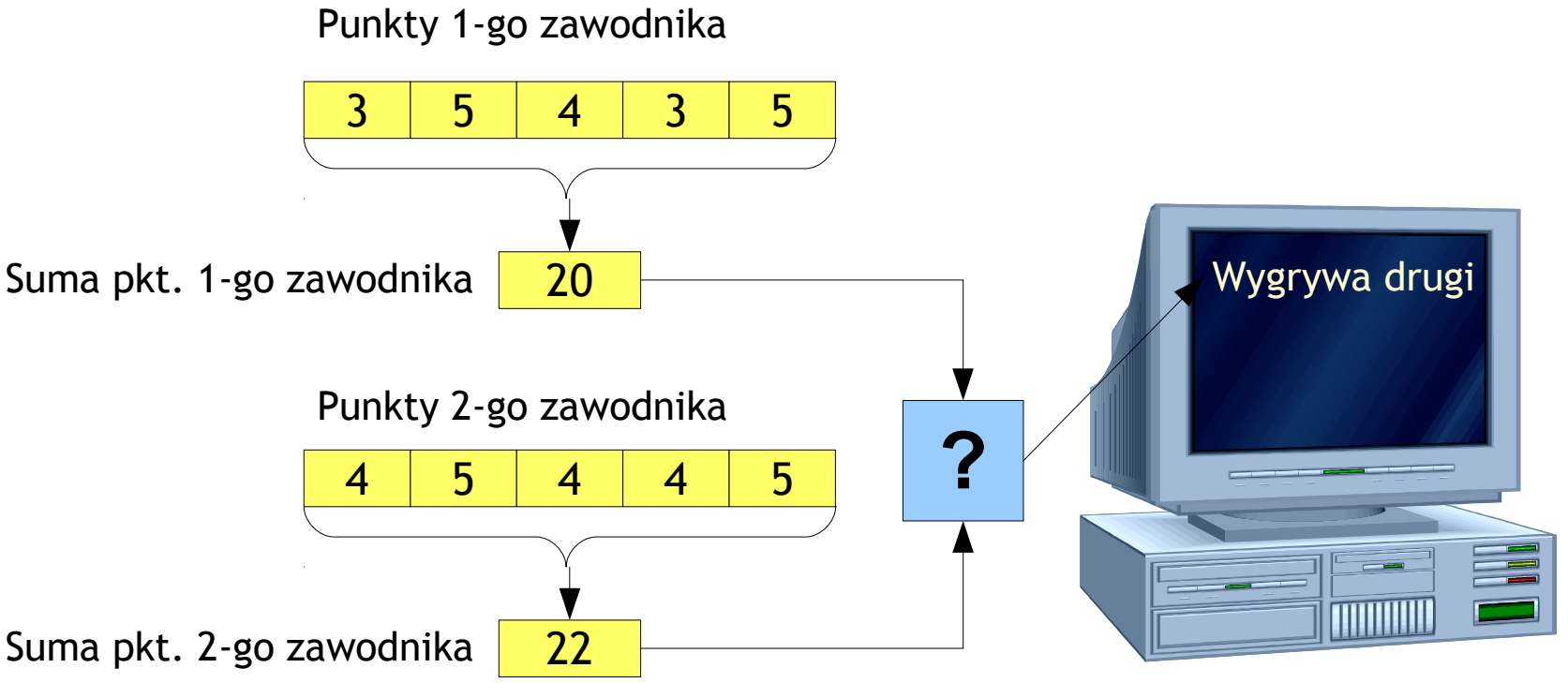
```
  cout << fixed;  // Notacja z kropka dziesietna
  cout.precision( 2 ); // Dwa miejsca po przecinku
  cout << "\nStatystyka czasow przejazdow";
  cout << "\n   Srednia: " << srednia;
  cout << "\n  Srodkowa: " << mediana;
  cout << "\n Wariancja: " << wariancja;
  cout << "\nOdchylenie: " << odchylenie;

  cout << endl << "Nacisnij Enter by zakonczyc program";
  cin.ignore();
  cin.get();
  return EXIT_SUCCESS;
}
```

4/4

Wykorzystanie tablic, przykład 2

W pewnym pięcioboju zwodnicy rywalizują ze sobą parami. Punkty zdobyte w czasie każdej z pięciu konkurencji są sumowane. Zwycięża ten zawodnik, który zgromadził większą liczbę punktów.

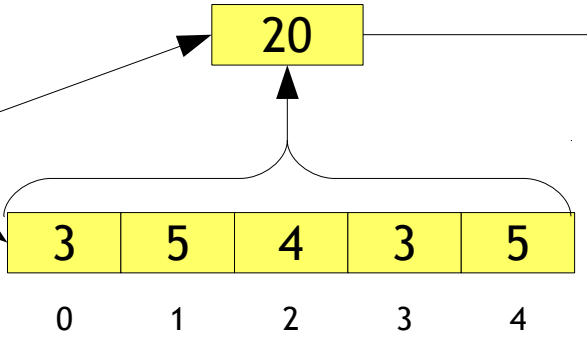
Należy napisać program, pozwalający na wczytanie punktów zgromadzonych przez każdego zawodnika, w każdej z pięciu konkurencji. Zadaniem programu jest wyznaczenie sumy punktów i wytypowanie zwycięzcy.



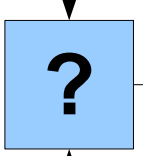
Tablica jak magazyn dla punktów każdej z konkurencji

```
const int MAKS = 5;
int punkty1[ MAKS ];
int suma1;
```

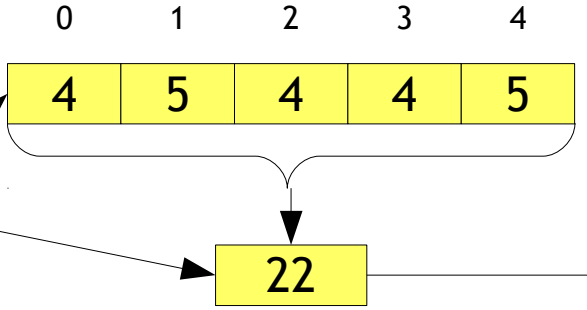
Suma pkt. 1-go zawodnika



Punkty 1-go zawodnika



Punkty 2-go zawodnika




```
const int MAKS = 5;
int punkty2[ MAKS ];
int suma2;
```

Suma pkt. 2-go zawodnika

Wykorzystanie tablic, przykładowe rozwiązanie, wersja 1-sza

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

int main()
{
  const int MAKS = 5;
  int punkty1[ MAKS ];
  int punkty2[ MAKS ];
  int i, suma1, suma2;

  cout << "Podaj punkty pierwszego zawodnika:" << endl;
  for( i = 0; i < MAKS; i++ )
  {
    cout << ">";
    cin >> punkty1[ i ];
  }

  cout << "Podaj punkty drugiego zawodnika:" << endl;
  for( i = 0; i < MAKS; i++ )
  {
    cout << ">";
    cin >> punkty2[ i ];
  }
```

1/2

Wykorzystanie tablic, przykładowe rozwiązanie, wersja 1-sza, cd. ...

```
  for( i = 0, suma1 = 0; i < MAKS; suma1 += punkty1[ i++ ] )
    ;

  for( i = 0, suma2 = 0; i < MAKS; suma2 += punkty2[ i++ ] )
    ;

  cout << endl << "Suma punktow pierwszego zawodnika: " << suma1;
  cout << endl << "Suma punktow drugiego   zawodnika: " << suma2;

  if( suma1 > suma2 )
    cout << endl << "Wygral pierwszy zawodnik";
  else
    if( suma1 < suma2 )
      cout << endl << "Wygral drugi zawodnik";
    else
      cout << endl << "Remis";

  cout << endl << "Nacisnij Enter by zakonczyc program";
  cin.ignore();
  cin.get();
  return EXIT_SUCCESS;
}
```

2/2

Wykorzystanie tablic, przykładowe rozwiązanie, wersja 1-sza, cd. ...

```
. . .
for( i = 0, suma1 = 0; i < MAKS; suma1 += punkty1[ i++ ] )
  ;

for( i = 0, suma2 = 0; i < MAKS; suma2 += punkty2[ i++ ] )
  ;

cout << endl << "Suma punktow pierwszego zawodnika: " << suma1;
cout << endl << "Suma punktow drugiego   zawodnika: " << suma2;
. . .
```

Sumowanie, zamiast dwóch iteracji jedna:

```
. . .
for( i = 0, suma1 = 0, suma2 = 0; i < MAKS; i++ )
{
  suma1 += punkty1[ i ];
  suma2 += punkty2[ i ];
}

cout << endl << "Suma punktow pierwszego zawodnika: " << suma1;
cout << endl << "Suma punktow drugiego   zawodnika: " << suma2;
. . .
```

Wykorzystanie tablic, przykładowe rozwiązanie, wersja 2-ga

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

void czytajPunkty( int punkty[], int ile );
int  sumujPunkty( int punkty[], int ile );
void pokazKtoWygral( int pkt1, int pkt2 );

int main()
{
  const int MAKS = 5;
  int punkty1[ MAKS ];
  int punkty2[ MAKS ];
  int suma1, suma2;

  cout << "Podaj punkty pierwszego zawodnika:" << endl;
  czytajPunkty( punkty1, MAKS );

  cout << "Podaj punkty drugiego zawodnika:" << endl;
  czytajPunkty( punkty2, MAKS );

  suma1 = sumujPunkty( punkty1, MAKS );
  suma2 = sumujPunkty( punkty2, MAKS );

  pokazKtoWygral( suma1, suma2 );
```

1/3

Wykorzystanie tablic, przykładowe rozwiązanie, wersja 2-ga

```
  cout << endl << "Nacisnij Enter by zakonczyc program";
  cin.ignore();
  cin.get();
  return EXIT_SUCCESS;
}

void czytajPunkty( int punkty[], int ile )
{
  for( int i = 0; i < ile; i++ )
  {
    cout << ">";
    cin >> punkty[ i ];
  }
}

int sumujPunkty( int punkty[], int ile )
{
  int suma = 0;
  for( int i = 0; i < ile; suma += punkty[ i++ ] )
    ;
  return suma;
}
```

2/3

Wykorzystanie tablic, przykładowe rozwiązanie, wersja 2-ga

```
void pokazKtoWygral( int pkt1, int pkt2 )
{
  cout << endl << "Suma punktow pierwszego zawodnika: " << pkt1;
  cout << endl << "Suma punktow drugiego   zawodnika: " << pkt2;
  if( pkt1 > pkt2 )
    cout << endl << "Wygral pierwszy zawodnik";
  else
    if( pkt1 < pkt2 )
      cout << endl << "Wygral drugi zawodnik";
    else
      cout << endl << "Remis";
}
```

3/3

Idziemy o krok dalej. A co, gdy zawodników jest np. 10-ciu?

```
const int LB_KONK = 5;
```

```
int punkty1[ LB_KONK ];
int suma1;
```

```
int punkty2[ LB_KONK ];
int suma2;
```

```
int punkty3[ LB_KONK ];
int suma3;
```

.
.
.

```
int punkty10[ LB_KONK ];
int suma10;
```

Punkty 1-go zawodnika

3	5	4	3	5
0	1	2	3	4

Punkty 2-go zawodnika

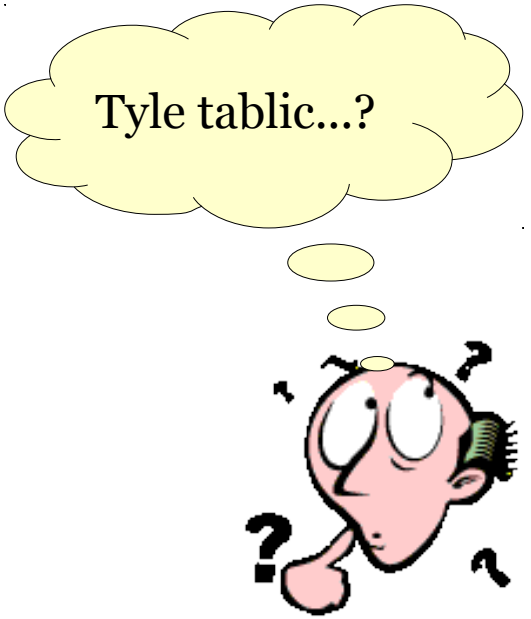
4	3	4	3	5
0	1	2	3	4

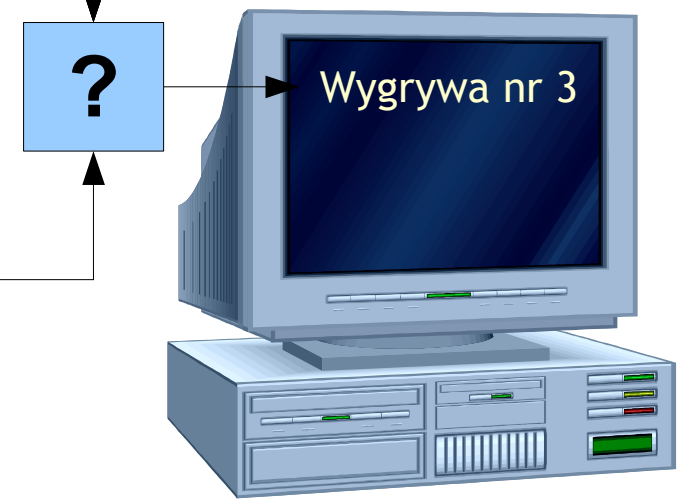
Punkty 3-go zawodnika

6	5	5	6	5
0	1	2	3	4

Punkty 10-go zawodnika

1	4	4	3	2
0	1	2	3	4





Idziemy o krok dalej. A co, gdy zawodników jest np. 10-ciu?

Punkty 1-go zawodnika

3	5	4	3	5
0	1	2	3	4

Punkty 2-go zawodnika

4	3	4	3	5
0	1	2	3	4

Punkty 3-go zawodnika

6	5	5	6	5
0	1	2	3	4

.
.
.

Punkty 10-go zawodnika

1	4	4	3	2
0	1	2	3	4

	0	1	2	3	4
0	3	5	4	3	5
1	4	3	4	3	5
2	6	5	5	6	5
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
9	1	4	4	3	2

Koncepcja tablicy dwuwymiarowej

Stałe określające liczbę zawodników oraz konkurencji:

```
const int LB_ZAWOD = 10;  // Liczba zawodnikow
const int LB_KONK  = 5;   // Liczba konkurencji
```

Tablica dwuwymiarowa to inaczej tablica, której elementami są tablice:

```
int punkty[ LB_ZAWOD ][ LB_KONK ];
```

		0	1	2	3	4
Punkty 1-go zawodnika	0	3	5	4	3	5
Punkty 2-go zawodnika	1	4	3	4	3	5
Punkty 3-go zawodnika	2	6	5	5	6	5
.	.					
.	.	.	.	.	.	.
.	.					
Punkty 10-go zawodnika	9	1	4	4	3	2

Tablicy dwuwymiarowej odpowiada matematyczne pojęcie *macierzy*. Tablicy jednowymiarowej odpowiada *wektor*.

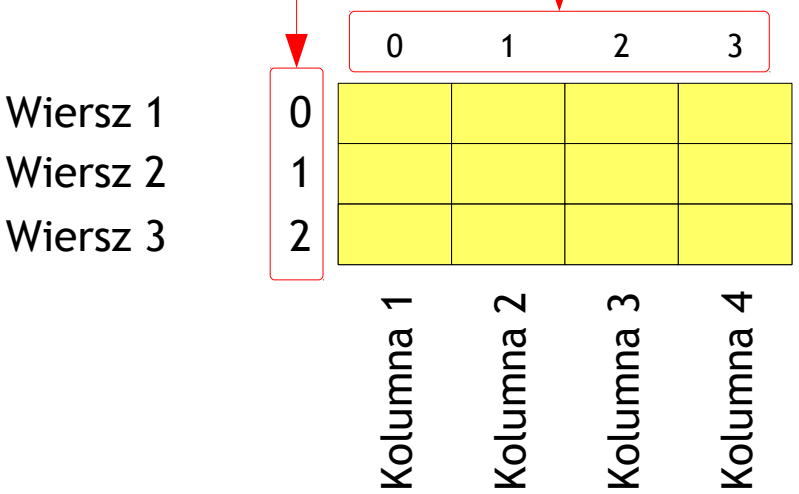
Prosta tablica dwuwymiarowa — deklaracja

Stałe określające liczbę wierszy i kolumn:

```
const int LB_W = 3;  // Liczba wierszy
const int LB_K = 4;  // Liczba liczba kolumn
```

Tablica dwuwymiarowa o rozmiarze 3x4 (LB_WxLB_K):

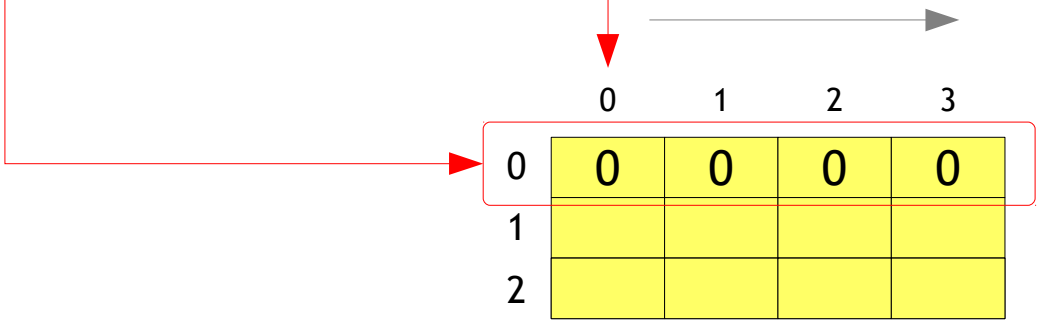
```
int tablica[ LB_W ][ LB_K ];
```



Prosta tablica dwuwymiarowa — przykłady operacji

Wstawianie liczby 0 do wszystkich elementów pierwszego wiersza:

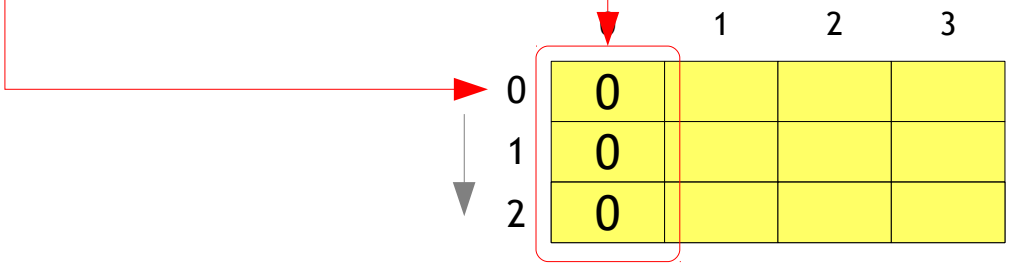
```
const int LB_W = 3;  // Liczba wierszy
const int LB_K = 4;  // Liczba liczba kolumn

int tablica[ LB_W ][ LB_K ];

int w, k; // Roboczy numer wiersza i kolumny

for( k = 0; k < LB_K; k++ )
  tablica[ 0 ][ k ] = 0;
```



Prosta tablica dwuwymiarowa — przykłady operacji

Wstawianie liczby 0 do wszystkich elementów pierwszej kolumny:

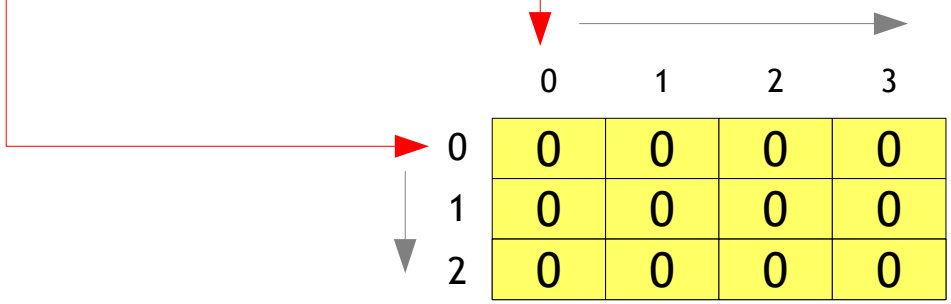
```
const int LB_W = 3;  // Liczba wierszy
const int LB_K = 4;  // Liczba liczba kolumn

int tablica[ LB_W ][ LB_K ];

int w, k; // Roboczy numer wiersza i kolumny

for( w = 0; w < LB_W; w++ )
  tablica[ w ][ 0 ] = 0;
```



Prosta tablica dwuwymiarowa — przykłady operacji

Wstawianie liczby 0 do wszystkich elementów tablicy (wypełnianie wierszami):

```
const int LB_W = 3;  // Liczba wierszy
const int LB_K = 4;  // Liczba liczba kolumn

int tablica[ LB_W ][ LB_K ];

int w, k; // Roboczy numer wiersza i kolumny

for( w = 0; w < LB_W; w++ )
  for( k = 0; k < LB_K; k++ )
    tablica[ w ][ k ] = 0;
```

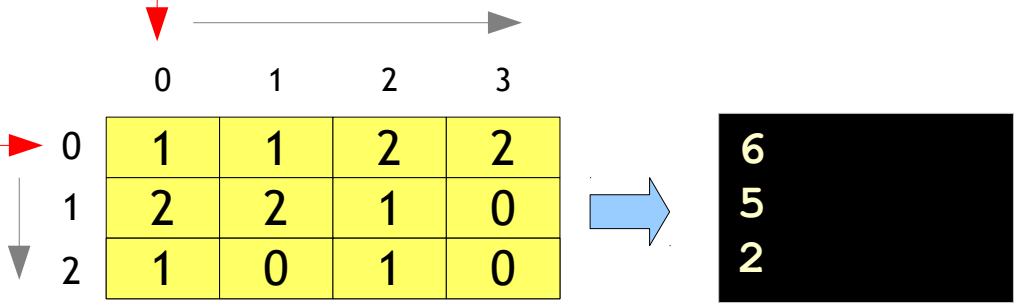


Przy każdym wykonaniu iteracji wewnętrznej, nr wiersza *w* jest ustalony. Zmienia się dopiero po wypełnieniu wartością 0 całego wiersza.

Prosta tablica dwuwymiarowa — przykłady operacji

Sumowanie wartości elementów w każdym wierszu

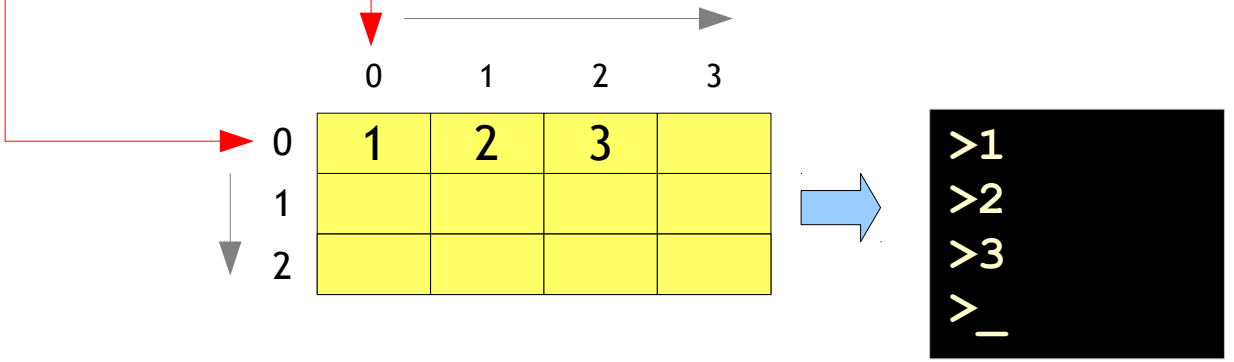
```
const int LB_W = 3;  // Liczba wierszy
const int LB_K = 4;  // Liczba liczba kolumn

int tablica[ LB_W ][ LB_K ];

int w, k; // Roboczy numer wiersza i kolumny

for( w = 0; w < LB_W; w++ )
{
  int suma = 0;
  for( k = 0; k < LB_K; k++ )
    suma += tablica[ w ][ k ];
  cout << endl << suma;
}
```



Prosta tablica dwuwymiarowa — przykłady operacji

Wprowadzanie danych do tablicy z **cin** wierszami.

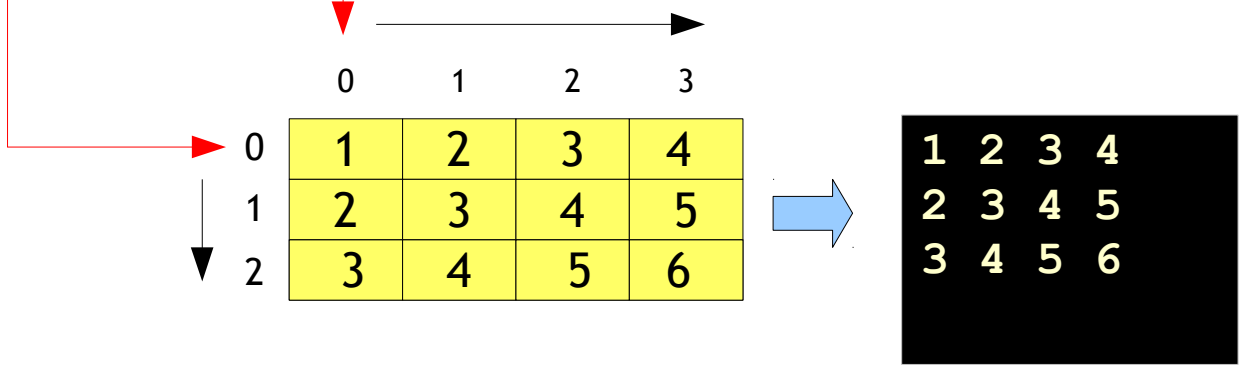
```
const int LB_W = 3;  // Liczba wierszy
const int LB_K = 4;  // Liczba liczba kolumn

int tablica[ LB_W ][ LB_K ];

int w, k; // Roboczy numer wiersza i kolumny

for( w = 0; w < LB_W; w++ )
{
  for( k = 0; k < LB_K; k++ )
  {
    cout << endl << '>';
    cin >> tablica[ w ][ k ];
  }
}
```



Prosta tablica dwuwymiarowa — przykłady operacji

Wyprowadzanie danych z tablicy do **cout** wierszami.

```
const int LB_W = 3;  // Liczba wierszy
const int LB_K = 4;  // Liczba liczba kolumn

int tablica[ LB_W ][ LB_K ];

int w, k; // Roboczy numer wiersza i kolumny

for( w = 0; w < LB_W; w++ )
{
  cout << endl;
  for( k = 0; k < LB_K; k++ )
    cout << tablica[ w ][ k ] << ' ';
}
```



Wracamy do programu z wynikami 10-ciu zawodników

Stałe określające liczbę zawodników oraz konkurencji, zmienne programu:

```
const int LB_ZAWOD = 10; // Liczba zawodnikow
const int LB_KONK  = 5;  // Liczba konkurencji

int punkty[ LB_ZAWOD ][ LB_KONK ]; // Tablica punktów

int nr_zawod, nr_konk;   // Zmienne robocze iteracji
int maks, nr_maks;       // Minimum i nr minimum
```

		0	1	2	3	4
Punkty 1-go zawodnika	0	3	5	4	3	5
Punkty 2-go zawodnika	1	4	3	4	3	5
Punkty 3-go zawodnika	2	6	5	5	6	5
.	.					
.	.	.	.	.	.	.
.	.					
Punkty 10-go zawodnika	9	1	4	4	3	2

Tablica wyników a suma punktów zawodnika

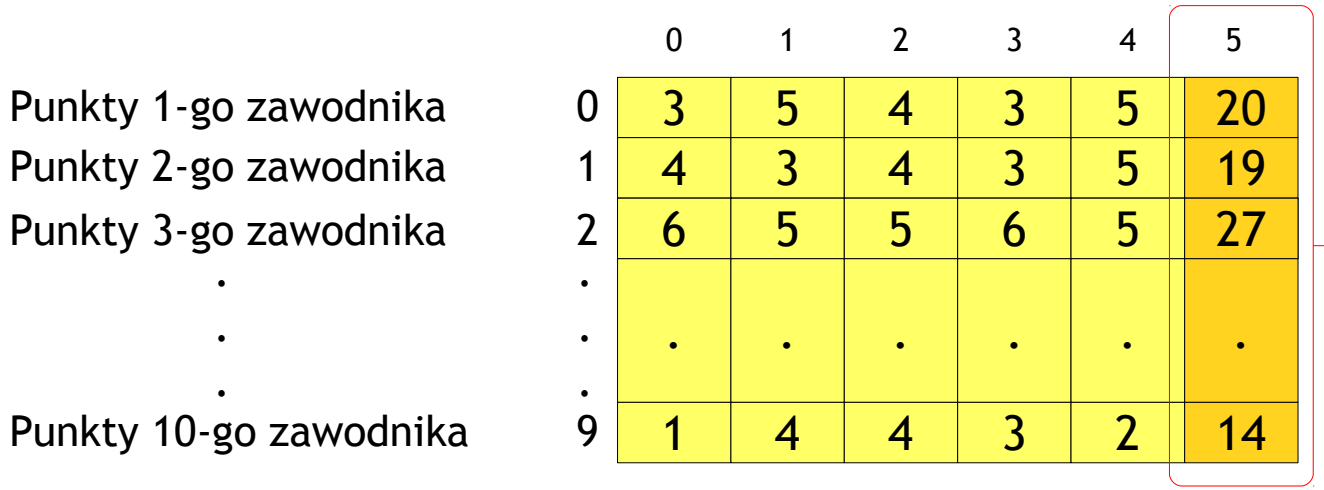
Będziemy wyznaczać sumy punktów każdego z zawodników. Sum zatem będzie tyle, ilu zawodników, gdzie je zapamiętać?

		0	1	2	3	4	
Punkty 1-go zawodnika	0	3	5	4	3	5	20
Punkty 2-go zawodnika	1	4	3	4	3	5	19
Punkty 3-go zawodnika	2	6	5	5	6	5	27
.	.						
.	.	.	.	.	.	.	.
.	.						
Punkty 10-go zawodnika	9	1	4	4	3	2	14

Potrzebna kolejna tablica?

Tablica wyników a suma punktów zawodnika

Wprowadzamy dodatkową kolumnę w tablicy, będzie ona przechowywać sumę punktów każdego zawodnika:



		0	1	2	3	4	5
Punkty 1-go zawodnika	0	3	5	4	3	5	20
Punkty 2-go zawodnika	1	4	3	4	3	5	19
Punkty 3-go zawodnika	2	6	5	5	6	5	27
.	.						
.	.	.	.	.	.	.	.
.	.						
Punkty 10-go zawodnika	9	1	4	4	3	2	14

```
const int LB_ZAWOD = 10;  // Liczba zawodnikow
const int LB_KONK  = 5;   // Liczba konkurencji
const int SUMA = LB_KONK; // Indeks elemementu wiersza na sume punktow

int punkty[ LB_ZAWOD ][ LB_KONK + 1 ]; // Tablica punktów


int nr_zawod, nr_konk;   // Zmienne robocze iteracji
int maks, nr_maks;       // Minimum i nr minimum
```

Odczyt danych

Wczytanie punktów kolejnych zawodników

```
// Czytaj punkty kolejnych zawodnikow
for( nr_zawod = 0; nr_zawod < LB_ZAWOD; ++nr_zawod )
{
  cout << "Podaj punkty zawodnika nr: " << nr_zawod + 1 << endl;
  for( nr_konk = 0; nr_konk < LB_KONK; ++nr_konk )
  {
    cout << '>';
    cin >> punkty[ nr_zawod ][ nr_konk ];
  }
}
```

Sumowanie

Sumowanie punktów kolejnych zawodników

```
// Sumuj punkty kolejnych zawodnikow
for( nr_zawod = 0; nr_zawod < LB_ZAWOD; ++nr_zawod )
{
  punkty[ nr_zawod ][ SUMA ] = 0;
  for( nr_konk = 0; nr_konk < LB_KONK; ++nr_konk )
    punkty[ nr_zawod ][ SUMA ] += punkty[ nr_zawod ][ nr_konk ];
}
```

Wyprowadzanie sum

Wyprowadzenie sum punktów kolejnych zawodników

```
// Wyprowadz sumy punktów kolejnych zawodników
cout << endl << "Sumy punktow zawodnikow";
for( nr_zawod = 0; nr_zawod < LB_ZAWOD; ++nr_zawod )
{
  cout << endl << nr_zawod + 1 << ": ";
  cout << punkty[ nr_zawod ][ SUMA ];
}
```

Znajdowanie zwycięzcy

Wyznaczanie maksimum wśród sum punktów

```
// Wyznacz nalepszy wynik -- maksymalna sume
maks = punkty[ nr_maks = 0 ][ SUMA ];
for( nr_zawod = 1; nr_zawod < LB_ZAWOD; ++nr_zawod )
  if( punkty[ nr_zawod ][ SUMA ] > maks )
  {
    nr_maks = nr_zawod;
    maks = punkty[ nr_zawod ][ SUMA ];
  }

cout << endl << "Wygrywa zawodnik nr : " << nr_maks + 1;
cout << endl << "Jego suma punktów to: " << maks;
```

Uwaga na mały trik:

```
nr_maks = 0;
maks = punkty[ nr_maks ][ SUMA ];
```

```
maks = punkty[ nr_maks = 0 ][ SUMA ];
```

Kompletny program

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

int main()
{
  const int LB_ZAWOD = 3;   // Liczba zawodnikow
  const int LB_KONK  = 5;   // Liczba konkurencji
  const int SUMA = LB_KONK; // Indeks elem. z suma punktow

  // Dwuwymiarowa tablica punktów wszystkich zawodnikow
  int punkty[ LB_ZAWOD ][ LB_KONK + 1 ];

  int nr_zawod, nr_konk; // Zmienne robocze iteracji
  int maks, nr_maks;     // Minimum i nr minimum
```

1/3

Kompletny program

```
// Czytaj i sumuj punkty kolejnych zawodnikow
for( nr_zawod = 0; nr_zawod < LB_ZAWOD; ++nr_zawod )
{
  punkty[ nr_zawod ][ SUMA ] = 0;
  cout << "Podaj punkty zawodnika nr: " << nr_zawod + 1 << endl;
  for( nr_konk = 0; nr_konk < LB_KONK; ++nr_konk )
  {
    cout << '>';
    cin >> punkty[ nr_zawod ][ nr_konk ];

    punkty[ nr_zawod ][ SUMA ] += punkty[ nr_zawod ][ nr_konk ];
  }// for
}// for

// Wyprowadz sumy punktów kolejnych zawodników
cout << endl << "Sumy punktow zawodnikow";
for( nr_zawod = 0; nr_zawod < LB_ZAWOD; ++nr_zawod )
{
  cout << endl << nr_zawod + 1 << ": ";
  cout << punkty[ nr_zawod ][ SUMA ];
}
```

2/3

Kompletny program

```
  // Wyznacz nalepszy wynik -- maksymalna sume
  maks = punkty[ nr_maks = 0 ][ SUMA ];
  for( nr_zawod = 1; nr_zawod < LB_ZAWOD; ++nr_zawod )
    if( punkty[ nr_zawod ][ SUMA ] > maks )
    {
      nr_maks = nr_zawod;
      maks = punkty[ nr_zawod ][ SUMA ];
    }

  cout << endl << "Wygrywa zawodnik nr : " << nr_maks + 1;
  cout << endl << "Jego suma punktów to: " << maks;
  cout << endl << "Nacisnij Enter by zakonczyc program";

  cin.ignore();
  cin.get();
  return EXIT_SUCCESS;
}
```

3/3

Kolejny program — wytypuj szóstkę do „dużego lotka”

Zadaniem programu jest wylosowanie sześciu liczb, pozwalających wypełnić pojedynczy typowanie tzw. „dużego lotka”.

Losujemy liczby z przedziału 1 .. 49, w wylosowanej szóstce nie może być powtórzeń.

```
Losuje szostke liczb z 49-ciu.
Wylosowana szostka:
4 9 17 31 37 43
Nacisnij Enter by zakonczyc program...
```

Wytypuj szóstkę do „dużego lotka” — analiza + pseudokod

```
Zdefiniuj zmienną całkowitą  — licznik losowanych liczb: wylosowanych
Zdefiniuj zmienną całkowitą  — pamięta aktualnie wylosowaną liczbę: liczba

Zainicjuj generator liczb pseudolosowych
Wyzeruj zmienna wylosowanych

Wykonuj
   Wylosuj wartość z przedzialu 1 .. 49 i wstaw do zmiennej liczba
   Wyprowadź zawartość zmiennej liczba do strumienia wyjściowego
   Zwiększ zmienną wylosowanych o 1
Dopóki zmienna Wylosowanych ma wartość mniejszą od 6
```

To ma szansę zadziałać, ale nigdy nie wiadomo, czy losując kolejną liczbę, nie otrzymamy czasem którejś z już wcześniej wylosowanych..., ale umówmy się, że ten problem rozwiążemy później.

Wytypuj szóstkę do „dużego lotka” — koślawa, pierwsza wersja

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

int main()
{
  const int ZAKRES = 49;  // Zakres losowanych liczb

  int liczba;       // Losowana liczba, kandydat do szostki
  int wylosowanych; // Licznik losowanych liczb

  // Inicjalizacja gneratora liczb pseudolosowych
  srand( ( unsigned )time( 0 ) );

  cout << "\nLosuje szostke liczb z 49-ciu.";
  cout << "\nWylosowana szostka:\n";
```

Wytypuj szóstkę do „dużego lotka” — koślawa, pierwsza wersja

```
  // Zerowanie licznika wylosowanych liczb
  wylosowanych = 0;
  do
  {
    // Losowanie liczby z zakresu 1..49
    liczba = rand() % ZAKRES + 1;

    // Wyprowadzenie wylosowanej liczby
    cout << liczba << ' ';
    ++wylosowanych;
  }
  while( wylosowanych < 6 );

  cout << "\n\nNacisnij Enter by zakonczyc program";
  cin.get();
  return EXIT_SUCCESS;
}
```

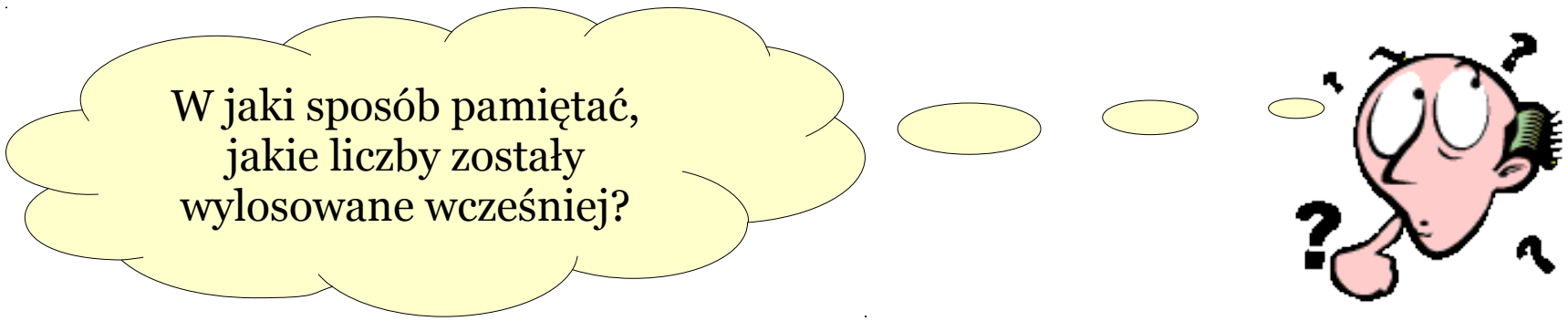
```
Wylosowana szostka:
35 42 9 9 20 32
Nacisnij Enter by zakonczyc program...
```

Wytypuj szóstkę do „dużego lotka” — jak wyeliminować duplikaty?

Zdefiniuj zmienną całkowitą — licznik losowanych liczb: wylosowanych
Zdefiniuj zmienną całkowitą — pamięta aktualnie wylosowaną liczbę: liczba

Zainicjuj generator liczb pseudolosowych
Wyzeruj zmienna wylosowanych

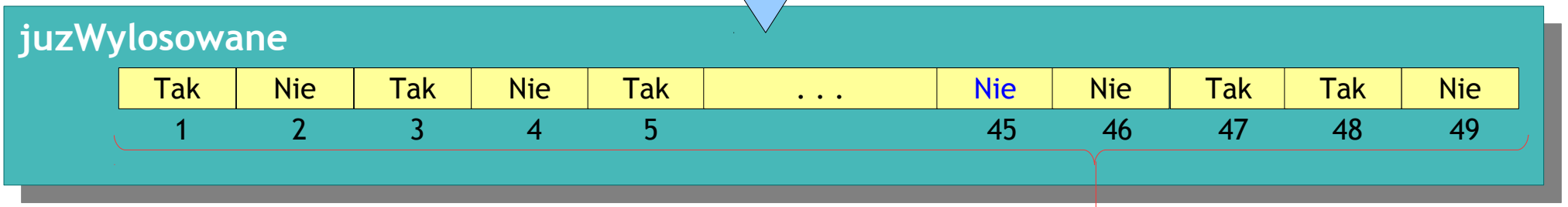
Wykonuj
  Wylosuj wartość z przedzialu 1 .. 49 i wstaw do zmiennej liczba
  Jeżeli wartość zmiennej Liczba *nie była* jeszcze wylosowana To
    Wyprowadź zawartość zmiennej liczba do strumienia wyjściowego
    Zwiększ zmienną wylosowanych o 1
Dopóki zmienna Wylosowanych ma wartość mniejszą od 6



Wytypuj szóstkę do „dużego lotka” — rejestr wylosowanych liczb

Każdy element rejestru odpowiada liczbie — kandydatce do szóstki. Jeżeli na danej pozycji w rejestrze ustawiona jest wartość *Tak*, to dana liczba została już wylosowana.

Jeżeli na zadanej pozycji jest wartość *Nie*, taka liczba jeszcze nie była wylosowana.



Załóżmy, że wylosowano liczbę 45.

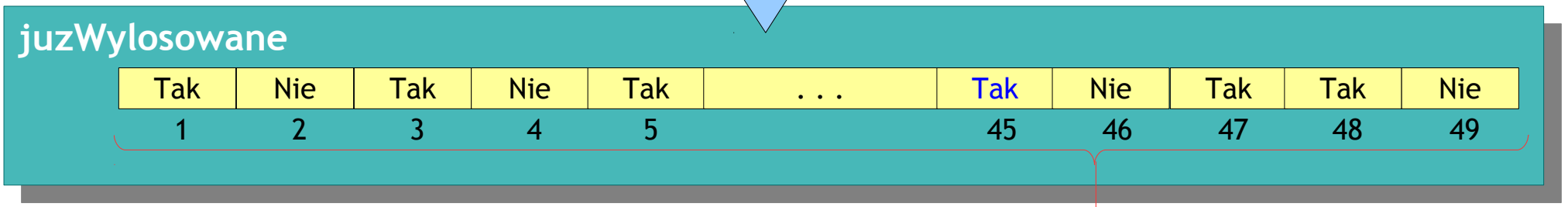
Na pozycji tej liczbie odpowiadającej jest *Nie*, zatem takiej liczby jeszcze nie wylosowano.

Liczba 45 może wejść do losowanej szóstki.

Wytypuj szóstkę do „dużego lotka” — rejestr wylosowanych liczb

Każdy element rejestru odpowiada liczbie — kandydatce do szóstki. Jeżeli na danej pozycji w rejestrze ustawiona jest wartość *Tak*, to dana liczba została już wylosowana.

Jeżeli na zadanej pozycji jest wartość *Nie*, taka liczba jeszcze nie była wylosowana.

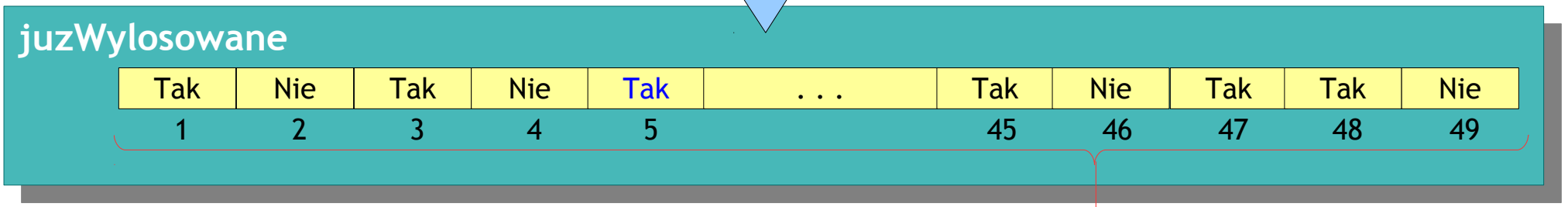


Zaznaczamy, że liczba 45 została wylosowana wpisując do rejestru *Tak*, na pozycji odpowiadającej tej liczbie.

Wytypuj szóstkę do „dużego lotka” — rejestr wylosowanych liczb

Każdy element rejestru odpowiada liczbie — kandydatce do szóstki. Jeżeli na danej pozycji w rejestrze ustawiona jest wartość *Tak*, to dana liczba została już wylosowana.

Jeżeli na zadanej pozycji jest wartość *Nie*, taka liczba jeszcze nie była wylosowana.



Załóżmy, że wylosowano liczbę 5.

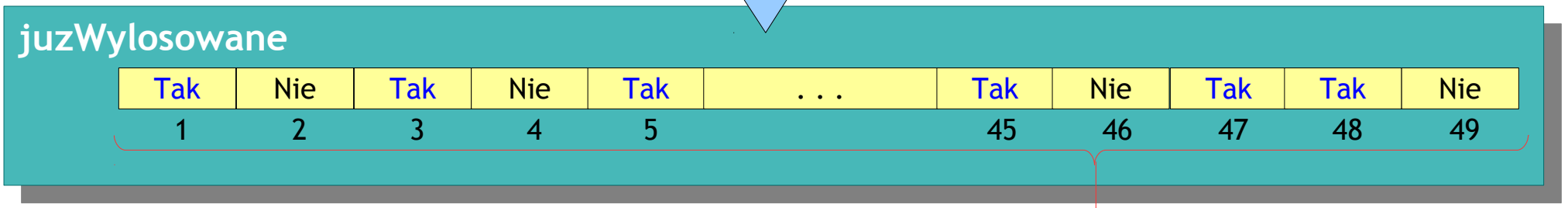
Na pozycji tej liczbie odpowiadającej jest *Tak*, zatem tę liczbę już wylosowano.

Liczba 5 zostaje odrzucona jako dublet.

Wytypuj szóstkę do „dużego lotka” — rejestr wylosowanych liczb

Każdy element rejestru odpowiada liczbie — kandydatce do szóstki. Jeżeli na danej pozycji w rejestrze ustawiona jest wartość *Tak*, to dana liczba została już wylosowana.

Jeżeli na zadanej pozycji jest wartość *Nie*, taka liczba jeszcze nie była wylosowana.

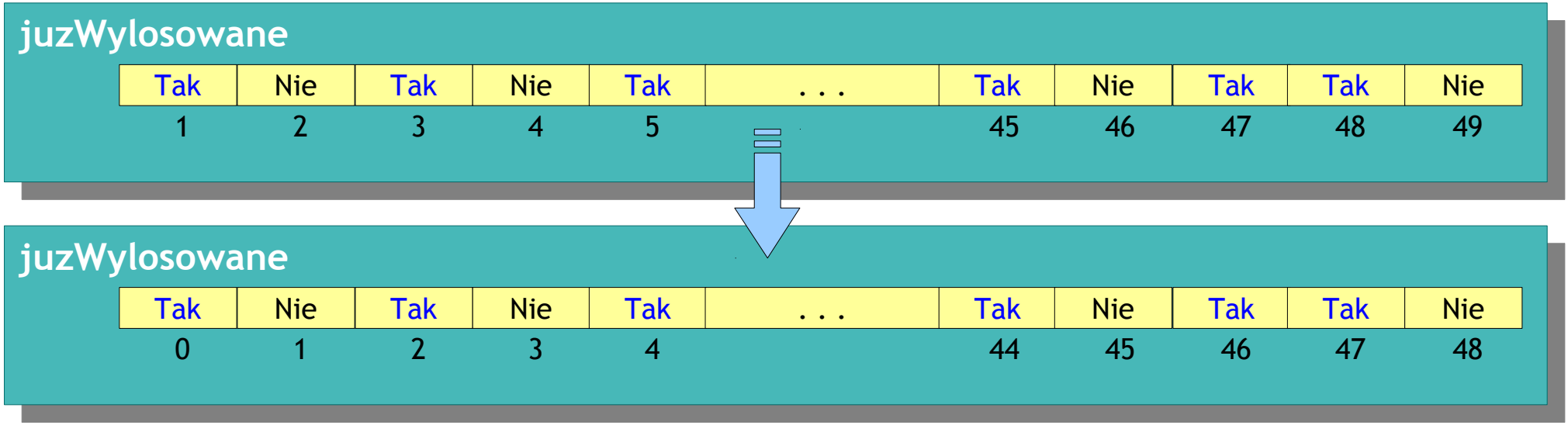


Z rejestru wylosowanych liczb można „wyczytać”, które z nich zostały wylosowane:

1 3 5 45 47 48

Rejestr wylosowanych liczb a język C++

Indeksy tablic w C/C++ liczone są od zera:



Jak zapisać *Tak*/*Nie* w C++?

```
const int ZAKRES = 49;          // Zakres losowanych liczb
bool  juzWylosowane[ ZAKRES ]; // Rejestr wylosowanych liczb
```

Operacje na rejestrze wylosowanych liczb a język C++

```
const int ZAKRES = 49;          // Zakres losowanych liczb
bool  juzWylosowane[ ZAKRES ]; // Rejestr wylosowanych liczb
int   liczba;                   // Losowana liczba, kandydat do szostki
```

Zerowanie rejestru

```
for( liczba = 0; liczba < ZAKRES; ++liczba )
  juzWylosowane[ liczba ] = false;
```

Czy wartość w zmiennej *liczba* była wylosowana?

```
if( juzWylosowane[ liczba ] )
{
  Tu coś, gdy liczba już była wylosowana
}
```

Czy wartość w zmiennej *liczba* nie została jeszcze wylosowana?

```
if( ! juzWylosowane[ liczba ] )
{
  Tu coś, gdy liczba nie została jeszcze wylosowana
}
```

Zaznaczenie w rejestrze, że wartość w zmiennej *liczba* została wylosowana:

```
juzWylosowane[ liczba ] = true;
```

Operacje na rejestrze wylosowanych liczb a język C++

Zaznaczenie w rejestrze, że wartość w zmiennej *liczba* została wylosowana:

```
wylosowanych = 0;
do
{
  // Losowanie liczby z zakresu 0..48
  liczba = rand() % ZAKRES;

  if( ! juzWylosowane[ liczba ] )
  {
    // Zapisanie w rejestrze, ze lb. zostala wylosowana
    juzWylosowane[ liczba ] = true;

    // Zwiekszenie liczby wylosowanych liczb
    ++wylosowanych;
  }
}
while( wylosowanych < 6 );
```

Wyprowadzenie wylosowanych liczb do strumienia wyjściowego:

```
for( liczba = 0; liczba < ZAKRES; ++liczba )
  if( juzWylosowane[ liczba ] )
    cout << liczba + 1 << ' ';
```

Wytypuj szóstkę do „dużego lotka” — pełna wersja programu

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

int main()
{
  const int ZAKRES = 49;          // Zakres losowanych liczb

  bool  juzWylosowane[ ZAKRES ]; // Rejestr wylosowanych liczb

  int liczba;       // Losowana liczba, kandydat do szostki
  int wylosowanych; // Licznik losowanych liczb

  // Inicjalizacja gneratora liczb pseudolosowych
  srand( ( unsigned )time( 0 ) );

  cout << "\nLosuje szostke liczb z 49-ciu.";
  cout << "\nWylosowana szostka:\n";

  // Zerowanie licznika wylosowanych liczb
  for( liczba = 0; liczba < ZAKRES; ++liczba )
    juzWylosowane[ liczba ] = false;
```

Uwaga! Wewnątrz programu losowane będą liczby z zakresu 0..48. Przekształcenie na zakres 1..49 nastąpi przy wyprowadzaniu liczb.

Wytypuj szóstkę do „dużego lotka” — pełna wersja programu

```
  wylosowanych = 0;
  do
  {
    // Losowanie liczby z zakresu 0..48
    liczba = rand() % ZAKRES;
    if( ! juzWylosowane[ liczba ] )
    {
      // Zapisanie w rejestrze, ze lb. zostala wylosowana
      juzWylosowane[ liczba ] = true;
      // Zwiekszenie liczby wylosowanych liczb
      ++wylosowanych;
    }
  }
  while( wylosowanych < 6 );

  // Wyprowadzenie wylosowanej szostki do strumienia wyjsciowego
  for( liczba = 0; liczba < ZAKRES; ++liczba )
    if( juzWylosowane[ liczba ] )
      cout << liczba + 1 << ' ';

  cout << "\n\nNacisnij Enter by zakonczyc program";
  cin.get();
  return EXIT_SUCCESS;
}
```

Wewnątrz programu losowane są liczby z zakresu 0..48.

Dostosowanie do zakresu 1..49.

To jeszcze nie koniec z tablicami, one będą często wracały, za chwilę powrócą w postaci tablic znaków

Pytania? Polemiki?
Teraz, albo:
roman.siminski@us.edu.pl