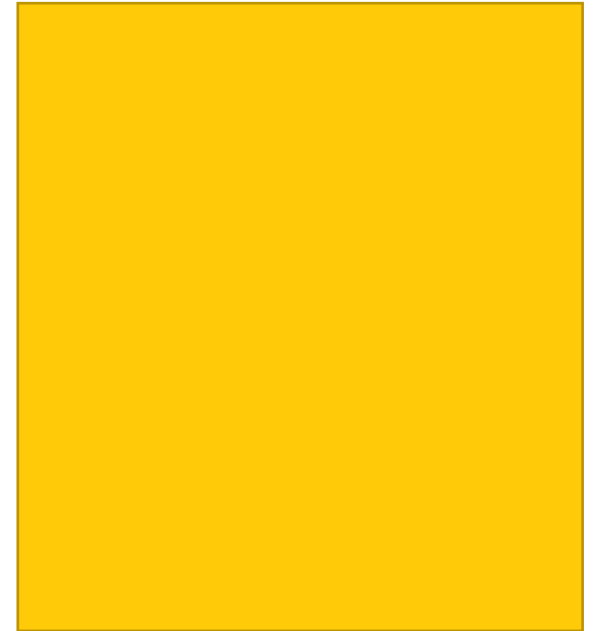


Podstawy programowania

# Laboratorium 4

Anna Prusinowska

# Tablice



# Po co są tablice?

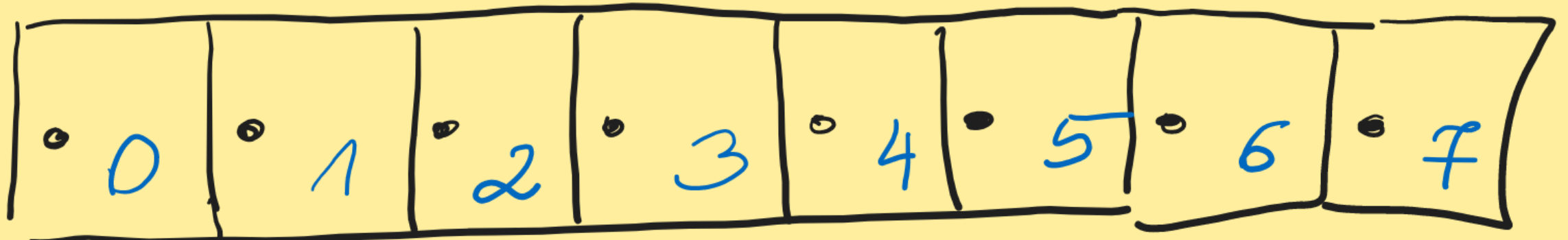
Cieężko wyobrazić sobie program księgowy, w którym można zarejestrować tylko jedną fakturę lub sklep internetowy, gdzie można mieć tylko jeden produkt w koszyku.

Aby uniknąć takiej sytuacji każdy program komputerowy operuje na zbiorach danych np. na zbiorze faktur albo zbiorze przedmiotów w koszyku.

## Przykład z życia wzięty

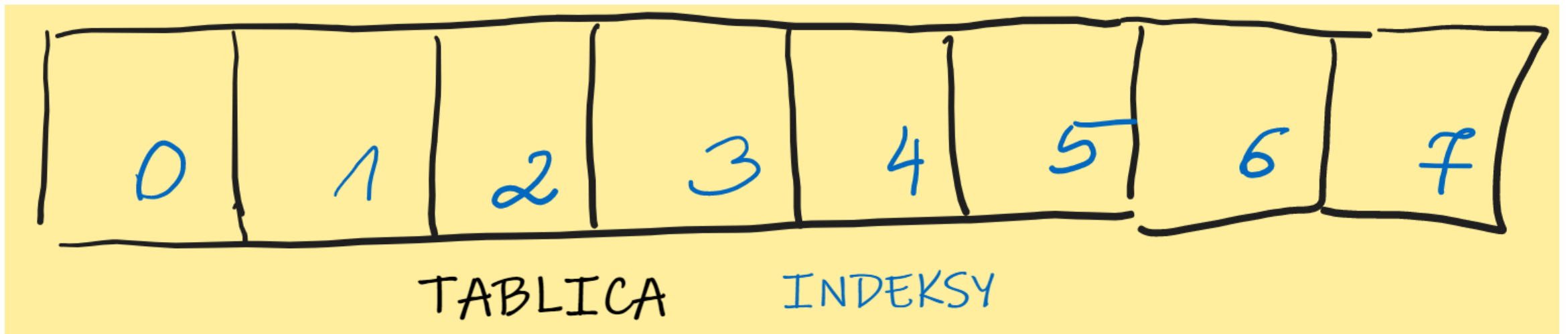
Wyobraź sobie, że chcesz uporządkować rzeczy w pokoju. Aby to zrobić, zamierzasz wykorzystać szafki – do każdej schowasz jakieś przedmioty. Ponieważ szafki są takie same, dlatego dla ułatwienia ponumerujesz je.

Ponieważ jesteś informatykiem, numerację rozpocznieś od 0.



# Przykład z życia wzięty

Taki zbiór przedmiotów, uporządkowanych w szafkach, możemy porównać do **tablic**.

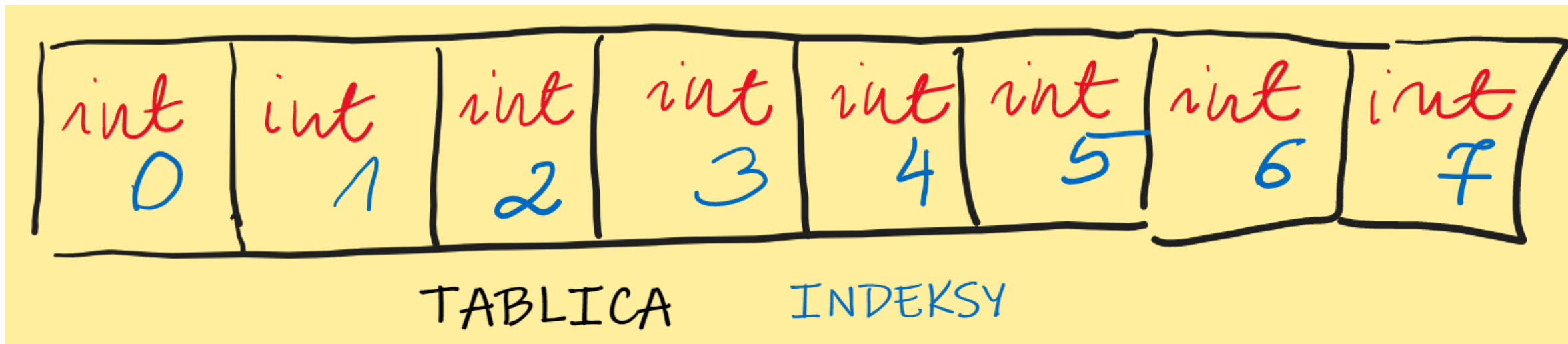


# Tablica - definicja

**Tablica** to zbiór (ciąg) elementów **tego samego typu**, które zajmują ciągły obszar w pamięci.

Tablica jest **typem pochodnym**. Oznacza to, że bierze się jakiś typ (np. int) i z jego elementów buduje się tablicę. Jest to wtedy tablica elementów typu int np.

**int nazwa\_tablicy[8]**



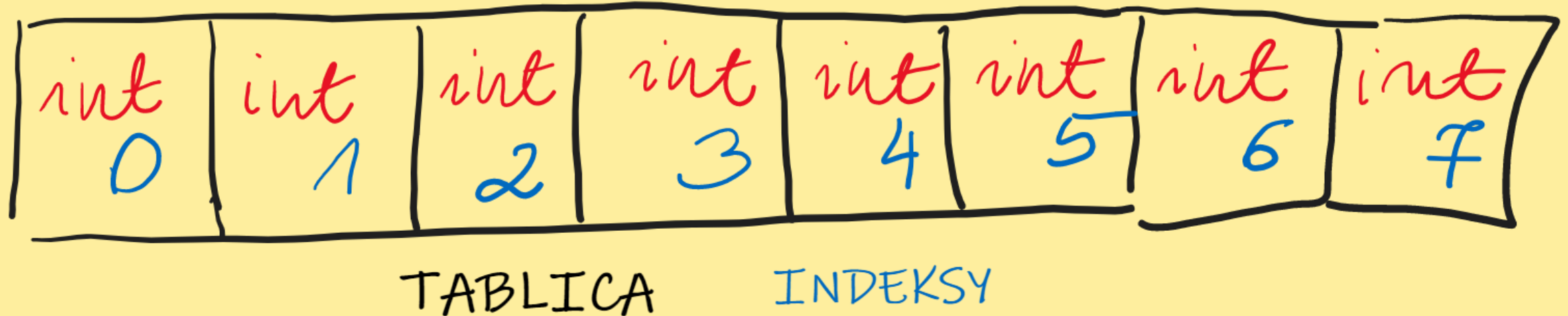
# Tablice – zapamiętaj!

- Rozmiar definiowanej tablicy musi być stałą całkowitą (większą od zera).
- Rozmiar tablicy musi być znany już w trakcie kompilacji – kompilator musi „wiedzieć”, ile miejsca ma zarezerwować na daną tablicę np.
  - `char znak[40]` // znak jest tablicą 40 elementów typu char;
  - `float liczba[3]` // liczba jest tablicą 3 elementów typu float;
  - `unsigned long kot[1222]` // kot jest tablicą 1222 elementów typu unsigned long.

# Tablice - zapamiętaj!

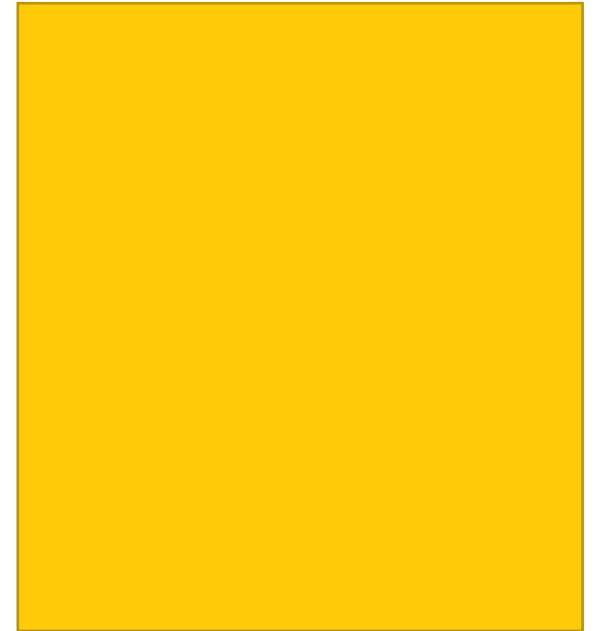
**Numeracja elementów tablicy zaczyna się od zera.**

Np. tablica **int tab[5]** to tablica 5 elementów typu int, a jej elementy to: **tab[0]**, **tab[1]**, **tab[2]**, **tab[3]**, **tab[4]**.





# Tworzenie tablic



# Tablice można tworzyć z:

- typów fundamentalnych (poza void);
- typów wyliczeniowych (enum);
- wskaźników;
- innych tablic;
- klas (obiektów, zdefiniowanych przez użytkownika);
- ze wskaźników do pokazywania na składniki klasy.

# Inicjalizacja tablic



# Inicjalizacja tablic

Tablicę można zainicjalizować w momencie definicji np.

```
int wiek[4] = {6, 77, 81, 2};
```

Wynikiem powyższej inicjalizacji będzie:

wiek[0] = 6



wiek[1] = 77

wiek[2] = 81

wiek[3] = 2

Element wiek[4], a także każdy inny, poza powyższymi (np. wiek[1000], wiek[1000000]) – nie istnieje!

# Inicjalizacja tablic – inne sposoby i przykłady

- `int B[4] = {13, 4};`  `B[0]=13, B[1]=4, B[2]=0, B[3]=0` – wszystkie pozostałe wartości poza podanymi explicite wynoszą 0;
- `int D[ ] = {0, 1, 2, 7, 4, 6};`  kompilator przelicza, ile liczb podano w klamrach i rezerwowana jest pamięć na tę liczbę elementów.

# Przykład 4a

Tablice

## Przykład 4a: Wypisz wszystkie wartości przechowywane w tablicy *imiona*.

```
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8      string imiona[6] = { "Helena", "Zbigniew", "Andrzej", "Marek", "Zofia", "Julia" };
9      for (int i = 0; i < 6; i++) {
10         cout << imiona[i] << endl;
11     }
12 }
```

Konsola debugowania programu Microsoft Visual Studio

Helena  
Zbigniew  
Andrzej  
Marek  
Zofia  
Julia

E:\PJ\_PRG\LABORATORIA\Lab 4\Lab4\_Przyklady\Lab4\_Przy  
.  
Naciśnij dowolny klawisz, aby zamknąć to okno...

# Tablice znakowe





# Tablice znakowe

- **Tablice znakowe** służą do przechowywania znaków (np. liter).
- Teksty w tablicach znakowych przechowuje się tak, że po ciągu liter (ich kodów liczbowych) następuje znak o kodzie **0 tj. null**. Oznacza on, gdzie kończy się ciąg liter.
- Ciąg liter zakończony znakiem null nazywa się **string** lub **C-string**.
- Niewymienione w tablicy elementy inicjuje się do końca tablicy zerami.

char word[137] = { "nej!" };

h 0	e 1	f 2	! 3	NULL 4	5	...	...	136	137
--------	--------	--------	--------	-----------	---	-----	-----	-----	-----

# TABLICA

# INDEKSY

# Tablice znakowe – przykład 2

```
char word[137] = { 'h', 'e', 'j', '!' };
```

odpowiada zapisowi

```
word[0] = 'h';
```

```
word[1] = 'e';
```

```
word[2] = 'j';
```

```
word[3] = '!';
```

- ponieważ nie było cudzysłowu, kompilator nie dokończył inicjowania znakiem **null**;
- wszystkie elementy tablicy, poczynając od word[4] do word[137] włącznie zostaną zainicjowane zerami;
- ponieważ **null** ma kod **0**, łańcuch w tablicy word zostanie poprawnie zakończony.

# Tablice znakowe – uwaga

```
char word[] = { 'h', 'e', 'j', '!' };
```

- jest to definicja tablicy znakowej o 4 elementach;
- znaku **null** tam nie będzie;
- tablica word nie przechowuje łańcucha znaków, ale pojedyncze znaki.

W przypadku definicji: 

```
char word[] = { "hej!" };
```

- zostanie zarezerwowana pamięć dla 5 elementów tablicy znakowej word;
- kolejne elementy tablicy przechowują następujące znaki: 'h', 'e', 'j', '!' i null.

# Przykład 4g

Tablice znakowe  
- rozmiary

## Przykład 4g: Program wyświetlający rozmiar zadanych tablic znakowych

```
4  #include <iostream>
5  #include <conio.h>
6  using namespace std;
7
8  int main() {
9
10     char nameFirst[] = { "Kasia" };
11     char nameSecond[] = {'K', 'a', 's', 'i', 'a'};
12
13     cout << "Rozmiar tablicy nameFirst = " << sizeof(nameFirst) << endl;
14     cout << "Rozmiar tablicy nameSecond = " << sizeof(nameSecond) << endl;
15
16     return 0;
17 }
```

```
Rozmiar tablicy nameFirst = 6
Rozmiar tablicy nameSecond = 5
```

# Wpisywanie tekstu do istniejących tablic – przykład 1

Przykład funkcji, kopiującej łańcuchy:

```
void copyString(char stringA[], char stringB[]) {  
    for (int i = 0; ; i++) {  
        stringA[i] = stringB[i];  
        if(stringA[i] == NULL) break;  
    }  
}
```

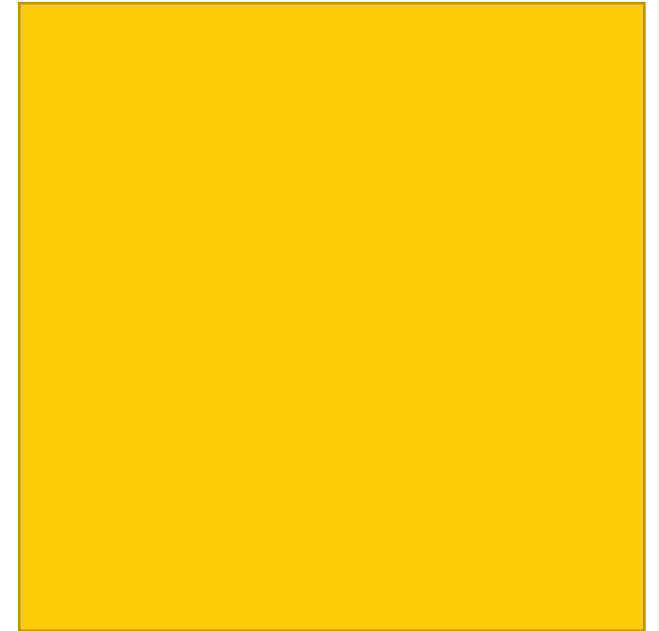
## Wpisywanie tekstu do istniejących tablic – przykład 2

Inny przykład funkcji, kopiującej łańcuchy:

```
void copyString(char stringA[], char stringB[]) {  
    int i = 0;  
    do {  
        stringA[i] = stringB[i];  
    }  
    while (stringA[i++] != NULL);  
}
```



# Tablice wielowymiarowe



# Tablice wielowymiarowe

- Tablice można tworzyć z różnych typów obiektów.
- Można je też tworzyć z innych tablic np.

**int table [3] [4];**

Tablica **table** składa się z **3 wierszy** i **4 kolumn**:

table [0] [0]	table [0] [1]	table [0] [2]	table [0] [3]
table [1] [0]	table [1] [1]	table [1] [2]	table [1] [3]
table [2] [0]	table [2] [1]	table [2] [2]	table [2] [3]

# Tablice wielowymiarowe w pamięci komputera

Elementy tablicy wielowymiarowej umieszczane są kolejno w pamięci komputera tak, że najszybciej zmienia się najbardziej skrajny prawy indeks. Tzn., że **tablica przechowywana jest „rzędami”**.

Przykład: `int Z [2] [4] = { 10, 20, 30, 40, 50, 60, 70, 80 };`

spowoduje przypisanie wartości poszczególnym elementom tablicy

`Z [0] [0] = 10;    Z [0] [1] = 20;    Z [0] [2] = 30;    Z [0] [3] = 40;`

`Z [1] [0] = 50;    Z [1] [1] = 60;    Z [1] [2] = 70;    Z [1] [3] = 80;`

# Sposoby na przekazywanie argumentów do funkcji

Tablice i inne  
jako  
argumenty  
funkcji

# Przekazywanie argumentów do funkcji

Do funkcji można przekazać argumenty.

Tak jak w matematyce, gdzie  $y = f(x)$ , tak w programowaniu przykładowa funkcja może wyglądać następująco:

**int funkcja (int x)**

gdzie *int x* jest argumentem funkcji.

# Sposoby przekazywania argumentów do funkcji

1. Przekazywanie argumentów przez wartość.
2. Przekazywanie argumentów przez wskaźnik.
3. Przekazywanie argumentów przez referencję.
4. Przekazywanie tablic jako argument funkcji.

# Przekazywanie argumentów przez wartość

- Funkcja zawiera argumenty wraz z ich typami, oddzielone przecinkami.
- Zmienna przekazana do funkcji jest jej kopią.
- Ponieważ w funkcji wszelkie operacje wykonują się na kopii zmiennych, dlatego widoczne są tylko w bloku funkcji.
- Aby można było widzieć zmiany poza ciałem funkcji, trzeba przypisać do odpowiednich zmiennych poza funkcją wartość zwracaną przez funkcję (słowo kluczowe *return*).
- Jedna funkcja może zwrócić tylko jedną wartość, w związku z czym modyfikacja może dotyczyć tylko jednej zmiennej spoza funkcji.

# Przykład 4b

Funkcje –  
przekazywanie  
argumentów  
przez wartość



## Przykład 4b: Program czterokrotnie zwiększający wartość początkową

```
3  #include <iostream>
4  using namespace std;
5
6  int increase(int number)
7  {
8      // zmienna 'number' jest kopią zmiennej 'length'
9      number = number * 4;
10     return number;
11 }
12
13 int main()
14 {
15     cout << "Program zwiększający czterokrotnie liczbę początkową." << endl << endl;
16     int length = 250;
17     cout << "int length = " << length << endl;
18     // przypisujemy nową wartość dla zmiennej
19     length = increase(length);
20     cout << "result = " << length << endl;
21     return 0;
22 }
```

AP

```
Program zwiększający czterokrotnie liczbę początkową.

int length = 250
result = 1000

E:\PJ_PRG\LABORATORIA\Lab 4\Lab4_Przykłady\Lab4_Przykład4b
.
Naciśnij dowolny klawisz, aby zamknąć to okno...
```


# Wskaźniki



# Wskaźniki

- **Wskaźnik** jest typem zmiennej (**zmienną wskaźnikową**).
- Do wskaźnika można przypisać dowolny adres pamięci, na którą ten wskaźnik będzie wskazywał.
- Przekazując wskaźnik do funkcji, zostaje on przekazany przez wartość (czyli skopiowany), jednak prawdziwą wartością wskaźnika jest **adres pamięci**.
- Próbując wyświetlić wskaźnik, wyświetlony zostanie adres pamięci.
- Aby dostać się do prawdziwej wartości wskaźnika, trzeba posłużyć się **operatorem wyłuskania** \* (**gwiazdka**) – operatorem pobrania wartości wskaźnika.

# Wskaźniki

- Stosowanie wskaźników bardzo przyspiesza operacje na tablicach.
- Treścią wskaźnika jest informacja o tym, gdzie wskazywany obiekt się znajduje.
- np. `int *wsk`  wskaźnik nazywa się „wsk” (bez gwiazdki). Gwiazdka mówi, że coś o nazwie „wsk” jest wskaźnikiem.

# Wskaźniki – dynamiczne alokowanie pamięci

Wskaźniki – 4 ważne zastosowania:

1. Umożliwiają dynamicznie alokowanie pamięci RAM.
2. Zwiększają szybkość działania programu.
3. Pozwalają na przekazywanie podprogramom do pracy oryginalnych zmiennych jako argumentów (w tym również całych tablic).
4. Pozwalają na współpracę z urządzeniami zewnętrznymi.

# Przykład 4h

Wskaźniki

**Przykład 4h:** Program prezentujący odnośnienie się do obiektu poprzez nazwę oraz wskaźnik, który na ten obiekt pokaże

```

3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8      int zmienna = 6, drugi = 3; // zdefiniowanie 2 obiektów typu int
9      int* wskaznik; // zdefiniowanie wskaźnika
10
11     wskaznik = &zmienna; // wskaźnik pokazuje na adres obiektu zmienna;
12                        // adres jest podstawiony do wskaźnika operatorem przypisania =
13
14     // prosty wypis na ekran
15     cout << "zmienna = " << zmienna << endl;
16     cout << " -> a odczytana przez wskaznik = " << *wskaznik << endl;
17
18     zmienna = 12; // przypisanie nowej wartości do zmiennej
19                // wskaźnik cały czas pokazuje na ten obiekt i zauważa zmianę
20     cout << "zmienna = " << zmienna << endl;
21     cout << " -> a odczytana przez wskaznik = " << *wskaznik << endl;
22
23     *wskaznik = 100; // wskaźnik pokazywał na zmienną, więc do obiektu zmienna zostaje wpisana liczba 100
24                    // ! do obiektu można wpisać coś na różne sposoby:
25                    // 1) albo używając jego nazwy (zmienna)
26                    // 2) albo używając wskaźnika, który na ten obiekt pokazuje (*wskaznik)
27     cout << "zmienna = " << zmienna << endl;
28     cout << " -> a odczytana przez wskaznik = " << *wskaznik << endl;
29
30     wskaznik = &drugi; // wskaźnik nie pokazuje raz na zawsze na ten sam obiekt
31                        // można go łatwo przestawić, aby pokazywał na inny (drugi)
32                        // liczba 100 jest treścią zmiennej,
33                        // a wyrażenie *wskaznik odnosi się tutaj do obiektu drugi
34     cout << "zmienna = " << zmienna << endl;
35     cout << " -> a odczytana przez wskaznik = " << *wskaznik << endl;
36     return 0;
37 }

```

```

zmienna = 6
-> a odczytana przez wskaznik = 6
zmienna = 12
-> a odczytana przez wskaznik = 12
zmienna = 100
-> a odczytana przez wskaznik = 100
zmienna = 100
-> a odczytana przez wskaznik = 3

```



# Przekazywanie argumentów przez wskaźnik

- Umożliwia ono modyfikację więcej niż jednej wartości więcej niż jednej zmiennej, występującej poza funkcją (bez użycia *return*).
- Wewnątrz funkcji nie są tworzone kopie zmiennych spoza funkcji (argumenty wskaźnikowe wskazują na zmienne spoza funkcji).
- Wskaźniki przekazywane są przez wartość. Wskaźnik jest typem zmiennej (zmienna wskaźnikowa). Mimo, że zostaje skopiowany, to wartość wyłuskana ze wskaźnika nie jest kopią.

# Przykład 4c

Funkcje –  
przekazywanie  
argumentów  
przez wskaźnik

## Przykład 4c: Program modyfikujący wartości trzech zmiennych

Uwagi:

- Aby pobrać adres dowolnej zmiennej wystarczy napisać: **&nazwa\_zmiennej**.
- Aby utworzyć zmienną wskaźnikową, to po typie zmiennej dopisujemy **\*** (**gwiazdkę**).

```

1  ~
2  #include <iostream>
3  using namespace std;
4
5
6  void increaseLot(int* length, int* height, int* scales)
7  {
8      // zmienna '*length', '*height' i '*scales' nie są kopiami - operowanie na nich zmienia ich wartość w "całym" programie
9      // funkcja nie zwraca nic (ponieważ nie miałyby to sensu)
10     *length = *length * 2;
11     *height = *height * 3;
12     *scales = *scales * 4;
13 }
14
15 int main()
16 {
17     // zmienne
18     int length = 10;
19     int height = 100;
20     int scales = 1000;
21
22     cout << "Wartosci przed modyfikacja: " << endl;
23     cout << "length = " << length << endl;
24     cout << "heingt = " << height << endl;
25     cout << "scales = " << scales << endl << endl;
26
27     // wskaźniki do zmiennych
28     int* indicator_length = &length;
29     int* indicator_height = &height;
30     int* indicator_scales = &scales;
31
32     // wywołanie funkcji
33     increaseLot(indicator_length, indicator_height, indicator_scales);
34
35     // wyświetlenie nowych wartości
36     cout << "Wartosci po modyfikacji: " << endl;
37     cout << "length = " << length << endl;
38     cout << "heingt = " << height << endl;
39     cout << "scales = " << scales << endl << endl;
40     return 0;
41 }

```

Wartosci przed modyfikacja:

```

length = 10
heingt = 100
scales = 1000

```

Wartosci po modyfikacji:

```

length = 20
heingt = 300
scales = 4000

```

# Referencja

- **Referencja** to bezpośredni adres pamięci zmiennej. Nie można jej zmienić, skasować, uszkodzić.
- **Adres** to nie tylko numer komórki pamięci. To dodatkowa informacja, jakiego typu obiekt znajduje się pod danym numerem.
- Nadpisanie referencji powoduje natychmiastową utratę danych (w odróżnieniu do wskaźnika, którego usunięcie nie spowoduje utraty zmiennej, na jaką wskazywał).

# Przekazywanie argumentów przez referencję

- Jest ono łatwiejsze, niż korzystanie ze wskaźników.
- Zmienne wewnątrz funkcji nie są kopią, co oznacza, że operując na zmiennych referencyjnych operujemy też na zmiennych oryginalnych (modyfikujemy je).
- W funkcji tworzona jest dowolna liczba argumentów w raz z typami.
- Nazwy argumentów **poprzedzone są &**.

# Przykład 4d

Funkcje –  
przekazywanie  
argumentów  
przez  
referencję

## Przykład 4d: Program modyfikujący wartość zmiennej (wykorzystanie referencji)

```
4  #include <iostream>
5  using namespace std;
6
7  void change(int& number)
8  {
9      // modyfikując referencję modyfikujemy też zmienną oryginalną value
10     number = 111222333;
11 }
12
13 int main()
14 {
15     int value = 0;
16     cout << "value = " << value << endl;
17
18     // wywołanie funkcji (referencja zmiennej 'value')
19     change(value);
20
21     // wyświetlenie nowej wartości
22     cout << "changed value = " << value << endl;
23     return 0;
24 }
```

```
value = 0
changed value = 111222333
```



# Przekazywanie tablicy jako argumentu funkcji

- Tablice przesyła się do funkcji, podając funkcji tylko adres początku tej tablicy.
- **Nazwa tablicy bez podania jej indeksu** jest **wskaźnikiem na pierwszy element tablicy (adresem jej zerowego elementu)**.
- W przypadku tablic dwuwymiarowych (lub większych) w argumencie funkcji należy podać wymiar tablicy (inaczej program nie skompiluje się, ponieważ kompilator nie będzie w stanie określić wielkości poszczególnych jej wymiarów).

# Przesyłanie tablicy wielowymiarowej do funkcji

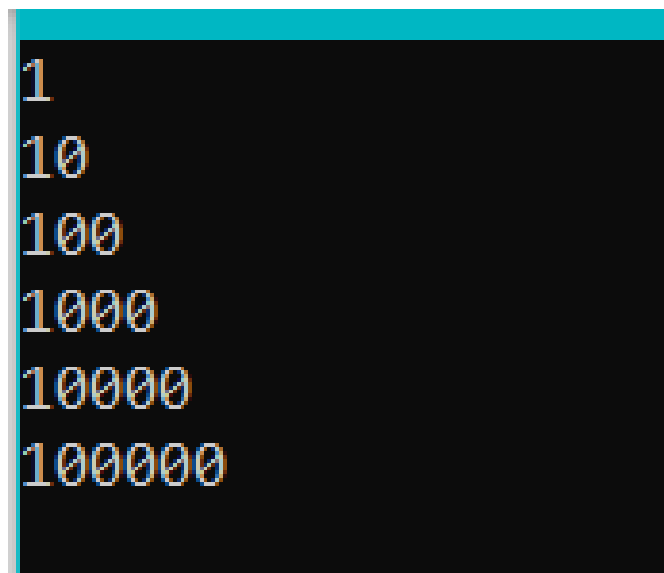
- Do funkcji przesyłany jest tylko adres początku tablicy.
- Powinien być znany typ elementów tej tablicy.
- Aby funkcja mogła łatwo obliczyć sobie, gdzie w pamięci znajduje się określony element, musi znać liczbę kolumn tej tablicy np. `void funkcjaA(int Z[ ][2])` lub `funkcjaB(int Z[5][2])`.

# Przykład 4e

Funkcje –  
przekazywanie  
tablicy  
jednowymiaro  
wej jako  
argumentu

## Przykład 4e: Program wyświetlający tablicę jednowymiarową

```
3  #include <iostream>
4  using namespace std;
5
6  // przekazujemy przez wskaźnik
7  void display(int table[])
8  {
9      for (int t = 0; t < 6; t++)
10         cout << table[t] << endl;
11 }
12
13 int main()
14 {
15     int sampleArray[6] = { 1,10,100,1000,10000,100000 };
16
17     // nazwa tablicy to wskaźnik na tablica[0]
18     display(sampleArray);
19
20     return 0;
21 }
```



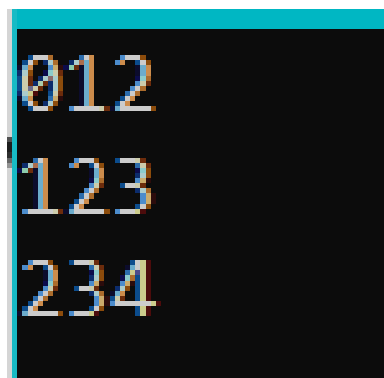
```
1
10
100
1000
10000
100000
```

# Przykład 4f

Funkcje –  
przekazywanie  
tablicy  
wielowymiarowej  
jako argumentu

## Przykład 4f: Program wyświetlający tablicę dwuwymiarową

```
3  #include <iostream>
4  using namespace std;
5
6  void display(int table[3][3])
7  {
8      for (int i = 0; i < 3; i++) {
9          for (int j = 0; j < 3; j++) {
10             cout << table[i][j];
11         }
12         cout << endl;
13     }
14 }
15
16 int main()
17 {
18     int sampleArray[3][3];
19
20     for (int i = 0; i < 3; i++)
21         for (int j = 0; j < 3; j++)
22             sampleArray[i][j] = i + j;
23
24     display(sampleArray);
25     return 0;
26 }
```



0	1	2
1	2	3
2	3	4

# Zadania

*Do  
samodzielnego  
wykonania*

## Zadanie 4.1

Znajdź największy element danej tablicy liczb całkowitych.  
Liczbę elementów tablicy i ich wartości pobierz od użytkownika.

(1 p.)



## Zadanie 4.2

Napisz program, który wskazuje najczęściej występujący element w tablicy liczb całkowitych. Liczbę elementów tablicy i ich wartości pobierz od użytkownika.

(2 p.)

## Zadanie 4.3

Napisz program, który w zadeklarowanej tablicy dwuwymiarowej 10x10 umieszcza:

- w pierwszej kolumnie liczby od 0 do 9;
- w drugiej sumę tych liczb (np. 0+0, 1+1 itd..);

(1 p.)

### **\* Część bonusowa:**

- w trzeciej – kwadraty tych liczb;
- w czwartej – sumę numeru wiersza i numeru kolumny;
- w piątej – wynik różnicy numeru wiersza i numeru kolumny;
- w pozostałych kolumnach 0 (interpretacja graficzna tablicy poniżej).

(\*2 p.)

# Wskazówki do zad. 4.3

## 1. **<conio.h>**

Conio oznacza Console-Input-Output. conio.h to niestandardowy plik nagłówkowy używany m. in. w C++. Ten plik zawiera funkcje wejścia-wyjścia konsoli, używane przez kompilatory.

## 2. **<iomanip>**

Manipulatory IO; nagłówek zapewniający manipulatory parametryczne. Szerzej – dokumentacja:

<https://docs.microsoft.com/pl-pl/cpp/standard-library/iomanip?view=msvc-160>