Programowanie Niskopoziomowe

Wykład 4

- Tablice

Instytut Informatyki

dr inż. Marcin Radom

Tablice jednowymiarowe

- Podstawowa teoria i przykłady

Tablice

- **Struktura danych** określająca układ powiązań występujących w pewnym zestawie danych.

- **Tablica** – ciąg elementów tego samego rodzaju.

- W języku C występuje ścisła zależność pomiędzy wskaźnikami i tablicami. Każdą operację, którą można przeprowadzić za pomocą indeksowania tablicy można wykonać za pomocą wskaźników (wersja szybsza).

- **Deklaracja tablicy jednowymiarowej:**

```
typ   identyfikator    [ rozmiar ] ;
```

typ **: liczbowy, znakowy, wskaźnikowy lub typ struktury**

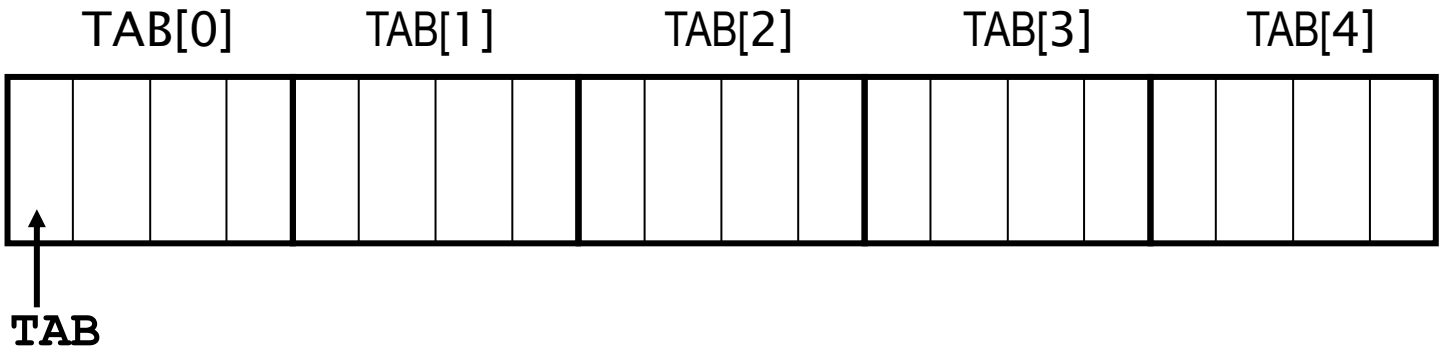
rozmiar **: wyrażenie stałe, o wartości większej od zera**

(w *gcc* wyrażenie całkowitoliczbowe)

Tablice

- **Przykład:**

```
int TAB[5];                    // 0, 1, 2, 3, 4
```



  - Definiuje tablicę *TAB* o rozmiarze 5, a więc blok pięciu kolejnych obiektów nazwanych *TAB[0], TAB[1], …, TAB[4]*
  - Zapis *TAB[i]* oznacza *i–ty* element tablicy *TAB*.
  - Nazwa tablicy *TAB* reprezentuje położenie jej elementu początkowego (nie jest jednak zmienną! – nie można przypisać innego adresu do *TAB*).

```
const int sumy = 50, iloczyny = 120;
float WYNIKI[ 2 * (sumy + iloczyny)]; // od wersji standardu C99
```

Tablice

- Odwołanie do tablicy następuje poprzez indeks – zawsze wartość OD ZERA do zadeklarowanej wielkości tablicy minus 1, np.:

```
int tablica[10];    // tablica 10-elementowa, numerowana od zera

int i;
i = TAB[3]; // TAB + 3 * sizeof int

WYNIKI[i + 2] = WYNIKI[i] + WYNIKI[i + 1];
```

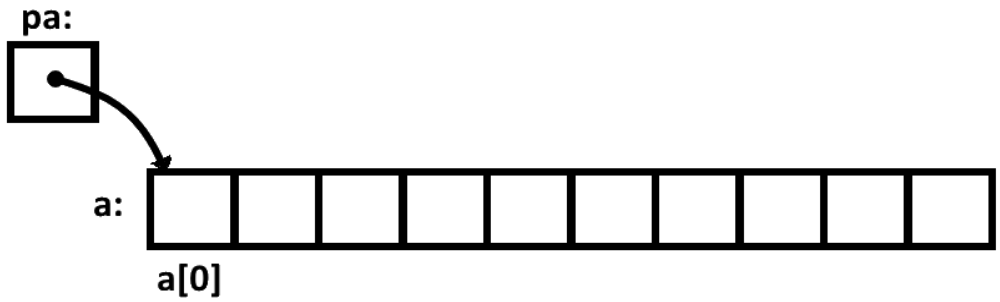
- W języku C nie jest sprawdzana zgodność indeksu z wymiarami tablicy! Często jest to przyczyną trudnych do wykrycia błędów. Np.:

```
double TAB_DANYCH [ 7 ] ;
for (int ix = 0; ix < 12; ++ix ) // wyjście poza zakres
     TAB_DANYCH [ ix ] = 48.74 ;
// program poprawnie się skompiluje, ale wystąpi błąd w trakcie
// wykonywania (wykonywanie programu zostanie przerwane przez
// system operacyjny)
```

Tablice – wskaźnik na element tablicy

- Przykładowo:

```
int a[10];
int *pa;                // wskaźnik do obiektów całkowitych
pa = &a[0]  // czytamy: wskaźnik pa zawiera (wskazuje) ADRES (&)
            // pierwszego elementu [0] tablicy a
```



Teraz poniższe przypisanie skopiuje zawartość *a[0]* do *x*:

```
int x = *pa;                // x ma taką WARTOŚĆ jak a[0], bo wskaźnik pa na to
                            // właśnie wskazuje; równoważne x = a[0]
```

- Jeśli wskaźnik *pa* (z powyższego przykładu) wskazuje na pierwszy element tablicy *a*, czyli *a[0]*, to z definicji ***(pa+1)*** wskazuje na element następny czyli *a[1]*.

Tablice – wskaźnik na element tablicy

- ***pa+i*** odnosi się do i-tego elementu po *pa* (w przykładzie do i-tego elementu tablicy *a*), ***pa-i*** do i-tego elementu przed *pa*.
- Jeśli chcemy odwołać się do WARTOŚCI elementu drugiego tablicy (o indeksie 1) to:

```
*(pa+1); // odnosi się do zawartości a[1] (w przykładzie)
```

- ***pa+i*** jest adresem *a[i]*, a ****(pa+i)*** jest zawartością *a[i]*
- Powyższe odnosi się do każdej tablicy niezależnie od rozmiaru lub typu elementów.
- **Wartością zmiennej lub wyrażenia typu tablicowego jest Z DEFINICJI adres pierwszego elementu tej tablicy**

```
int t[10];
int *wsk;
wsk = t;          // równoważne wsk = &t[0]; teraz wsk wskazuje na
                  // element t[0];
```

- Odwołanie do wartości ***t[i]*** jest równoważne odwołaniu ****(wsk+i)***. W języku C wyrażenie *t[i]* przy obliczaniu przekształcane jest bezpośrednio na **(t+i)*.

Tablice

- **&t[i]** – adres i-tego elementu tablicy, to samo co: **wsk+i** - to też jest adres elementu *t[i]* w tym przykładzie
- Podsumowując: *tablica i jej indeks* to TO SAMO co *wskaźnik i przesunięcie* (względem tegoż wskaźnika)
- Między nazwą tablicy a wskaźnikiem jest istotna różnica. Wskaźnik to ZMIENNA, więc jak u innych zmiennych dozwolone są np.:

```
int jakas_tablica[100];
int *wsk = jakas_tablica;
wsk++;     // ok, teraz wskazujemy na 2-gi element tablicy
(*wsk)++;  // a teraz ten 2-gi element zwiększamy o 1
           // nawiasy są konieczne, bez nich efektem obliczenia byłoby
           // zwiększenie wskaźnika wsk
```

- Nazwa tablicy nie jest zmienną, więc to nie jest dozwolone:

```
jakas_tablica++;      // BŁĄD
jakas_tablica = wsk;  // BŁĄD
```

- Jednoargumentowe operatory ***** i **&** wiążą silniej niż operatory arytmetyczne, tj. mają wyższy priorytet, np.:

```
y = *wsk + 1;         // najpierw brana jest wartość obiektu, na który
                      // wskazuje wsk, dodawany jest do niego 1 i wynik przypisywany
                      // jest do y
```

Tablice

- Operacje określone przez jednoargumentowe operatory * i ++ wykonywane są **od prawej do lewej** (prawostronnie łączne o tym samym priorytecie), dlatego:

```
++*p;      // nie wymaga nawiasów i równoważne *p += 1 i *p = *p + 1
(*p)++     // równoważne powyższemu, ale tylko z nawiasami
```

- Wskaźniki są zwykłymi zmiennymi, więc mogą być używane bez adresowania pośredniego, np. jeśli *iq oraz ip* są wskaźnikami do obiektów typu *int*, to:

```
iq = ip;
```

Kopiuje zawartość *ip* do *iq* powodując, że *iq* będzie wskazywać na to samo, na co wskazuje *ip*.

Tablice

- **Przykład:**

Zapisy *TAB i TAB [0] są równoważne.

```
char bufor[8];
char *pp;
pp = bufor;                          /* przypisania
pp = & bufor[0];                        równoważne */
```

- *(TAB + 4) **i** TAB [4] **są równoważne**

```
const int ele = 25;
short    TS[ele];
int      TI[ele];
double   TD[ele];
for (int i = 0; i < ele; ++i ) {
     *(TS + i) = 1;
     *(TI + i) = 1;
     *(TD + i) = 1.0;
}
```

Tablice jednowymiarowe - inicjalizacja

- Tablicę można zainicjować poprzez umieszczenie po jej deklaracji znaku równości, a następnie (w nawiasach klamrowych) listę wartości początkowych (są to wyrażenia stałe) rozdzielonych przecinkami:

  ***typ* identyfikator[rozmiar] = {lista_ wartości};**

  Np.:

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

- W takim wypadku nie trzeba podawać ile elementów ma tablica ponieważ kompilator obliczy jej długość na podstawie liczby podanych wartości początkowych.

- Jeżeli liczba wartości początkowych jest mniejsza od podanego rozmiaru tablicy to brakujące elementy otrzymają wartość 0 (dla zmiennych statycznych, automatycznych i zewnętrznych).

- Podanie zbyt wielu wartości początkowych jest błędem.
- **Przykłady:**

```
long MM[3] = { 154835L, 337782L, 0L };
/* MM[0] == 154835,
   MM[1] == 33782,
   MM[2] == 0 */
```

Tablice jednowymiarowe - inicjalizacja

- **Przykłady:**

```
const float F1 = 3.5E7F, F2 = 33.E8F;
float DANE[ ] = { F1 + F2, F1 / (F1 - F2)};
/* tablica DANE ma 2 elementy */

double PQ[150] = { 1.5, 3.8, 3.8, 2.7 };
/* PQ[0] == 1.5, PQ[1] == 3.8, PQ[2] == 3.8,
   PQ[3] == 2.7, pozostałe  elementy == 0 */

double Empty[1200] = {0};              // zerowanie

char NN[5] = { "alfa" };  // NN[4] == 0 znak końca łańcucha znaków
char MM[4] = { "beta" };  // błąd, tablica zbyt mała
```

Tablice jednowymiarowe – wczytywanie elementów tablicy

- **Przykład:**

```
int TabLicz [ 125 ];
for (int i = 0; i < 125; ++i)
      scanf("%d", &TabLicz[ i ]);

double Rzeczywiste [ 12 ];
for (i = 0; i < 12; ++i)
      scanf("%lf", &Rzeczywiste[ i ]);
```

- **Odczytanie rozmiaru tablicy:**

```
long A[5];
int obszar, element, elementow;
obszar    = sizeof A;               // == 20 (5 elementów po 4 bajty każdy)
element   = sizeof A[0];            // == 4 (1 element zajmuje 4 bajty)
elementow = sizeof A / sizeof A[0]; // == 5 elementów (20 / 4)
```

Tablice jednowymiarowe

- Tablice znakowe mogą być inicjalizowane w szczególny sposób – można użyć stałej napisowej, np.:

```
char wzorzec[] = "tak";              // rozmiar tablicy 4 (trzy znaki wzorca
                                     // plus znacznik końca tekstu '\0')
```

Równoważny zapis to:

```
char wzorzec[] = {'t', 'a', 'k', '\0'};
```

- **Rozszerzenia gcc**
  - Od wersji standardu **C99** (dostępne też jako rozszerzenie standardu **C89**) możliwe jest podawanie elementów tablicy w dowolnej kolejności za pomocą indeksu (wartość stała):

```
[index] = wartosc_elementu
```

```
int T[5] = { 3, [2] = 5, [4] = 2 };
/* T[0] == 3, T[1] == 0, T[2] == 5, T[3] == 0, T[4] == 2 */
for (int i = 0; i < 5; ++i)
            printf("\t%d", T[i]);
//  Wypisze: 3  0  5  0  2
```

Tablice jednowymiarowe

- **Przykład** - opracować program, który wczytuje tablicę jednowymiarową A i wyprowadza najpierw jej elementy o indeksach parzystych a potem o nieparzystych po jednym w linii w postaci: *A[i] = wartość*

```
int main() {
      double Tab[100];                  // tablica liczb
      int n, i;                         // ilość liczb, licznik
      n = 101;                          // błędna początkowa wartość

      while (n < 1 || n > 100) {
           printf("Podaj ilosc liczb [1-100] : ");
                 scanf("%d", &n);

      }

      for (i = 0; i < n; ++i) {
           printf("Podaj wartosc A[%d] : ", i);
           scanf("%lf", &Tab[i]);
      }
```

Tablice jednowymiarowe

```
        // nagłówek parzysty
        printf("\nElementy o indeksach parzystych.\n");

        for (i = 0; i < n; i += 2)
             printf("A[%d] = %.3lf\n", i, Tab[i]); // wyprowadzanie

        // nagłówek nieparzysty
        printf("\nElementy o indeksach nieparzystych.\n");

        for (i = 1; i < n; i += 2)
             printf("A[%d] = %.3lf\n", i, Tab[i]);     // wyprowadzanie

        printf("\n");
        return 0;
}
```

Tablice jednowymiarowe – arytmetyka na adresach

- Najprostsze i najczęściej stosowane konstrukcje (gdzie *p* jest wskaźnikiem do pewnego elementu tablicy):
  - ***p++*** - wskazuje na następny element w tablicy
  - ***p += i*** – wskazuje na element oddalony o *i* pozycji od aktualnie wskazywanego

- Wskaźniki można porównywać ze stałą **zero** oraz przypisać wskaźnikowi zero.

- Często zamiast zera używa się stałej symbolicznej **NULL** (zdefiniowana w nagłówku **<stdio.h>**), żeby podkreślić, że chodzi o specjalną wartość wskaźnikową.

```
if( p != 0 ) { … } równoważne if ( p != NULL ) { … }
```

- Przy pewnych ograniczeniach wskaźniki można ze sobą porównywać przy użyciu operatorów <, >, <=, ==, != oraz >=, tj.
  - jeżeli *p* i *q* wskazują na elementy pewnej tablicy to wyrażenie

    ```
    p > q
    ```

    jest prawdziwe tylko wtedy, gdy *p* wskazuje na dalszy element tablicy niż *q*.
- Dowolny wskaźnik zawsze można przyrównać do zera.

Tablice jednowymiarowe – arytmetyka na adresach

- Konstrukcja wskaźnik + wartość całkowita ***p + n*** oznacza adres n-tego elementu od miejsca wskazywanego przez *p* niezależnie od typu obiektów na które wskazuje *p*. Wartość przesunięcia *n* jest skalowalna do wielkości typu wynikającego z deklaracji *p* np. jeśli *p* wskazuje na obiekty typu double to czynnik skalujący wartość *n* to osiem.

- Po ludzku? Załóżmy, że *p* wskazuje na jakiś element tablicy typu int. Dajmy na to: pierwszy element (*p = &tablica[0]*)
  - Wtedy *p + 5* wskazuje na 5 elementów dalej, czyli 6-ty element tablicy (*tablica[5]*) Z TYM ŻE, polecenie *p + 5* przesunie adres w pamięci o 20 bajtów, ponieważ typ int jest kodowany na 4 bajtach (4*5 to 20, czyli p + 5 ustawi adres w pamięci 20 bajtów dalej).

- Odejmowanie wskaźników jest też poprawne: jeżeli *p* oraz *q* wskazują na elementy tej samej tablicy i *p < q*, to wtedy *q – p + 1* to liczba elementów POMIĘDZY *p* i *q* RAZEM z *q* (chodzi o to '+1').

- Przykład funkcji zliczającej znaki w łańcuchu znaków:

```
/* strlen: return length of string s */
int strlen(char *s) {
       char *p = s;            // wskaźnik p wskazuje na pierwszy znak tekstu
       while (*p != '\0') p++; // aż do napotkania znaku kończącego tekst
       return p - s;           // liczba znaków o jaką przesunięto p – długość tekstu
}
```

Tablice a przetwarzanie tekstów

- Funkcje biblioteki string.h

Tablice – przetwarzanie tekstów

- Stała napisowa np. **"jestem napisem"** jest tablicą znaków (w wewnętrznej reprezentacji jest zakończona znakiem '\0', aby programy wiedziały, gdzie ona się kończy. Np.:

```
char *Litery = "ABC";
```

- Stałe napisowe najczęściej występują jako argumenty funkcji, np.:

```
printf("jestem napisem");  // funkcja wypisze napis na ekranie,
// oczywiście bez znaku '\0', bo ten interesuje tylko program
```

- Gdy pojawia się w programie taki ciąg to dostęp do niego realizowany jest poprzez wskaźnik znakowy. Stała znakowa dostępna jest więc przez wskaźnik do swojego pierwszego znaku (**"jestem napisem"** jest chwilowym bytem, np. potrzebnym, aby *printf* coś wyświetliło, potem 'znika' z pamięci).

- Stałe znakowe nie muszą być argumentami funkcji, np.:

```
char *p_msg;
p_msg = "jestem napisem"; // p_msq przechowuje adres pierwszego znaku
```

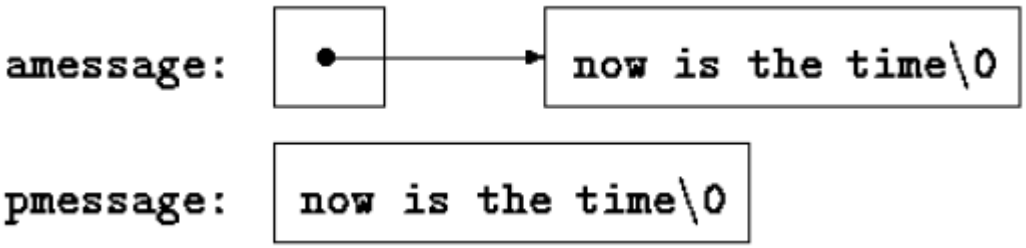
Tablice – przetwarzanie tekstów

- Zachodzi istotna różnica pomiędzy takimi definicjami:

```
/* Poszczególne znaki w tablicy można zamieniać, ale nazwa amessage zawsze będzie odwołaniem do tego samego miejsca w pamięci */
char amessage[] = "now is the time";    // Tablica
/* Wskaźnik, który został tak zainicjowany, wskazuje stałą napisową. Wskaźnik można zmieniać, żeby wskazywał na coś innego, ale próba zmiany w stałej napisowej ma skutek nieokreślony */
char *pmessage = "now is the time";     // Wskaźnik
```

- Znaki w tablicy *amessage* można zmieniać.

- Wskaźnik wskazuje na jeden konkretny napis. Można go potem wykorzystać do przechowywania adresu zupełnie innego napisu, ale nie można zmieniać poszczególnych literek 'oddzielnie' w oryginalnym napisie



Tablice – przetwarzanie tekstów, Przykład

- **Przykład:**

```
char *Napis = "Waniki.";          // tekst stały
*(Napis + 1) = 'y';               // błąd

char Tekst[16] = { "Pomołka." };
                                  // kopiowanie tekstu stałego do tablicy
Tekst[3] = 'y';                   // poprawnie
*(Tekst + 3) = 'y';               // poprawnie
```

Tablice – przetwarzanie tekstów, Przykład

- **Przykład**: Kopiowanie łańcuchów (tablic) znaków – czyli tekstów po prostu.

```
/* strcpy: copy t to s; array subscript version */
void strcpy(char *s, char *t){
      int i;
      i = 0;
      while ((s[i] = t[i]) != '\0')
           i++;
}
```

- Dla porównania, wersja robiąca to samo na wskaźnikach:

```
/* strcpy: copy t to s; pointer version */
void strcpy(char *s, char *t){
      int i;
      i = 0;
      while ((*s = *t) != '\0') {
           s++;  // przesuń wskaźnik s na następny znak
           t++;  // przesuń wskaźnik t na następny znak
      }          // krócej: while (*s++ = *t++) ;
}     // *s++ odczytaj wartość i przesuń wskaźnik
```

Tablice – przetwarzanie tekstów, Przykład

- Kopiowanie łańcuchów znaków: efektem instrukcji *s = t;* (gdzie *s* oraz *t* to tablice znaków) jest JEDNA tablica o dwóch nazwach -wskaźnikach: *s* oraz *t* (ponieważ do tego samego miejsca w pamięci może wskazywać dowolna liczba wskaźników).

- Porównywanie dwóch ciągów znaków – **strcmp** (*string compare*). Niekoniecznie tak jest ona zaimplementowana, ale działa tak samo jak:

```
/* strcmp: return <0 if s<t, 0 if s==t, >0 if s>t */
int strcmp(char *s, char *t) {
      int i;
      // wykonuje się tak długo aż nie trafi na pierwszą pozycję
      // na której tablice s i t się różnią
      for (i = 0; s[i] == t[i]; i++){
            if (s[i] == '\0')
                  return 0;
      }
      return s[i] – t[i];
}
```

Tablice – przetwarzanie tekstów, Przykład

- **Przykład:**

```
char Napis [ 16 ] = "Kopytko.";
int i = 0;
while( Napis[i] != 0 ){ // do końca tablicy
     // zamień wystąpienia liter 'o' na 'a' w tablicy Napis
     if ( Napis[i] == 'o') Napis[i] = 'a';
     ++i;
}
 // to samo, ale na wskaźnikach:
 char *ptr = Napis;
 while( *ptr != 0 ){     // do końca tablicy
     if ( *ptr == 'o')
          *ptr = 'a';
     ++ptr;
 }
```

Tablice – przetwarzanie tekstów, Przykład 1

- **Przykład:** Opracować prosty program szyfrujący, który w podanym tekście dokonuje następujących zmian:
  - zamiast liter a - z wpisuje literę następną (dla z -> a)
  - zamiast liter A - Z wpisuje literę poprzednią (dla A -> Z)
  - pozostałe znaki pozostawia bez zmian

  Tekst do przetwarzania może składać się z wielu wyrazów oddzielonych spacjami o łącznej długości jednej linii (do znaku NL).

  Program akceptuje 4 polecenia:
  - N | n - wczytaj nowy tekst,
  - K | k - koduj tekst,
  - D | d - dekoduj tekst,
  - Q | q- koniec programu.

```
int main(void){
	char Tekst[64];
	char* ptr;

	char opcja;
	bool dalej = true;
```

Tablice – przetwarzanie tekstów, Przykład 1 cd.

```
while (dalej){
     printf("Wprowadz opcje [N, K, D, Q] : ");
          fflush(stdin);
          scanf("%c", &opcja);

          switch(opcja & 0x5F){  // zamień na dużą literę np. 'n' na 'N'
               case 'N' :
                    printf("Wprowadz linie tekstu : \n");
                    fflush(stdin);
                    scanf("%63[ -~]", Tekst);  //&Tekst[0], ze spacjami
                    break;
               case 'K' :
                    printf("Tekst zakodowany : \n"); // koduj tekst
                    ptr = Tekst;// &Tekst[0]; pierwszy element tablicy
                    // aż nie trafi na znak '\0', końca tekstu
                    while(*ptr){
                         if (*ptr >= 'a' && *ptr < 'z') // oprócz litery z
                              printf("%c", *ptr + 1); // zwiększ kod
                                        // ASCII znaku o 1, czyli np. kod 'a'
                                        // to dziesiętnie 97, po dodaniu 1
                                        // wynosi 98 czyli oznacza 'b'
```

Tablice – przetwarzanie tekstów, Przykład 1 cd.

```
        else if (*ptr > 'A' && *ptr <= 'Z') // oprócz A
             printf("%c", *ptr - 1);       // poprzednia litera
        else if (*ptr == 'z')              // następna dla z to a
             printf("%c", 'a');
        else if (*ptr == 'A')              // poprzednia dla A to Z
             printf("%c", 'Z');
        else printf("%c", *ptr);           // tekst nie kodowany
             ++ptr; // przesunięcie wskaźnika na następną literę
}
printf("\n");
break;

case 'D':
     printf("Tekst zdekodowany : \n"); // dekoduj tekst
     ptr = Tekst;   // &Tekst[0]; pierwszy element tablicy
                    // aż nie trafi na znak '\0', końca tekstu
      while(*ptr){
          if (*ptr > 'a' && *ptr <= 'z') // oprócz a
               printf("%c", *ptr - 1);  // poprzednia litera
          else if (*ptr >= 'A' && *ptr < 'Z') // oprócz Z
               printf("%c", *ptr + 1);  // następna litera
```

Tablice – przetwarzanie tekstów, Przykład 1 cd.

```
                    else if (*ptr == 'a') // poprzednia dla a to z
                         printf("%c", 'z');
                    else if (*ptr == 'Z') // następna dla Z to A
                         printf("%c", 'A');
                    else
                          printf("%c", *ptr);
                               // tekst nie dekodowany
                    ++ptr;     // przesunięcie wskaźnika na następną
                               // literę w tekście
               }
               printf("\n");
               break;
          case 'Q':
               dalej = false; // koniec programu
               break;
          default:
               printf("Nie ma takiej opcji.\n"); // błędna opcja
     }
  }
  return 0;
}
```

Biblioteka string.h

- W nagłówku **<string.h>** zdefiniowane są następujące grupy funkcji operujących na łańcuchach znaków (tekstach):

```
typedef unsigned int size_t;
```

-
```
char* strcat( char* Destination, const char* Source );

      // Dodaje (konkatenuje) łańcuchy Source i Destination
      // (Source na koniec Destination). Wynik: wskaźnik
      // łańcucha, do którego dołączany jest tekst
```

-
```
char* strncat( char* Dest, const char* Source, size_t Count );

      // Analogicznie jak strcat lecz kopiuje nie więcej
      // niż Count znaków tekstu Source na koniec tekstu Dest
```

-
```
size_t strlen( const char* String );

      // Oblicza długość łańcucha String
```

Biblioteka string.h

- `char* strchr( const char* String, int C );`

```
// Wyszukuje pierwsze wystąpienie znaku C w łańcuchu
// String. Wynik: wskaźnik znalezionego znaku bądź NULL
// jeśli nie znaleziono
```

- `char* strrchr( const char* String, int C );`

```
// Analogicznie jak strstr lecz od końca (wskaźnik do
// ostatniego wystąpienia znaku C)
```

- `char* strpbrk( const char* String, const char* CharSet );`

```
// Wyszukuje pierwsze wystąpienie jednego ze znaków z
// łańcucha CharSet w łańcuchu String. Zwraca wskaźnik
// znalezionego znaku bądź NULL jeśli nie znaleziono
```

Biblioteka string.h

- `int strcmp( const char* String1, const char* String2 );`

```
// Porównanie dwóch łańcuchów. Wynik funkcji:
// < 0 - String1 mniejszy niż String2
// = 0 - String1 i String2 identyczne
//> 0 - String1 większy niż String2
```

- `int strncmp( const char* String1, const char* String2, size_t Count );`

```
// Analogicznie jak strcmp lecz porównuje nie więcej
// niż Count znaków zawartych w String1 i String2
```

Biblioteka string.h

- ```
  char* strcpy( char* Destination, const char* Source );

  // Kopiuje łańcuch Source do Destination łącznie z '\0'
  // usuwając poprzednią wartość Destination.
  // Wynik: wskaźnik łańcucha, do którego
  // kopiowany jest tekst
  ```

- ```
  char* strncpy( char* Dest, const char* Source, size_t Count );

  // Analogicznie jak strcpy lecz kopiuje nie więcej
  // jak Count znaków z Source do Dest. Jeżeli Source ma
  // mniej niż Count znaków to dopełnia Dest znakami '\0'
  ```

Biblioteka string.h – Przykład

```
#include <stdio.h>      #include <stdlib.h>     #include <string.h>
int main(int argc, char *argv[]) {
      char Tab[128];
      int dlugosc = 0;
      char* ptr = Tab;
      char szukany;
      int licznik = 0;
      printf("Wprowadz tekst wielowyrazowy:\n");
      scanf("%127[ -~]", Tab);      // łącznie za spacjami, do 127 znaków
      printf("Wprowadz szukany znak: ");
      fflush(stdin);                //czyszczenie bufora danych!
      scanf("%c", &szukany);

      dlugosc = strlen(Tab);
      while( ptr != NULL ){
            // zwraca wskaźnik do pierwszego wystąpienia szukanego znaku
            ptr = strchr(ptr, szukany);
            if( ptr != NULL) {
                  ++licznik;        // ilość wystąpień znaku w tekście w p.p. wyszukiwanie rozpocznie się
                                    // od ostatnio znalezionego znaku i pętla będzie działała w nieskończoność
                  ++ptr;
            }
      }
      printf("Znak %c zostal znaleziony %d razy.\n", szukany, licznik);
      system("PAUSE");
      return 0;
}
```

Dynamiczny przydział pamięci

- Tablice wskaźników,
  wielowymiarowe postrzępione, inne

Dynamiczny przydział pamięci C++

- Do przydzielania nowego bloku pamięci służy operator **new** (C++), np.:

```
int *wsk;
wsk = new int;          // powoduje utworzenie nowego obiektu typu int
                        // nie ma on nazwy, ale jego adres jest
                        // przekazywany wskaźnikowi wsk
```

- Natomiast do zwolnienia dynamicznie przydzielonej pamięci służy operator **delete** (C++).

```
delete wsk;             // powoduje likwidację powyższego obiektu (o ile
                        // wskaźnik wsk nadal na niego pokazuje)
```

- Utworzenie tablicy:

```
float *tab;
tab = new float[15];    // powoduje utworzenie piętnastoelementowej
                        // tablicy typu float. Tablica nie ma
                        // nazwy, ale wskaźnik jest informowany o
                        // jej adresie
delete [] tab;          // powoduje skasowanie tablicy
```

Dynamiczny przydział pamięci

- Przykłady:

```
float *taba  =  new float [ 150 ];
delete [ ] taba;

int  rozmiar;
scanf( "%d" , &rozmiar );
long *Tablica  =  new long [ rozmiar ];

double *TD1 = new double[150];
double x = TD1[53];                    // poprawnie

double *TD2 = new double [20][30];     // błąd
double *TD2 = new double [600];        // ok
double y = TD2[3][21];                 // błąd, TD2 jest jednowymiarowa
double z = TD2[3 * 30 + 21];           // ok
```

Tablice wskaźników

- Wskaźniki są zmiennymi, więc można z nich budować tablice tak jak z innych zmiennych (czyli też umieszczać je we wspólnej tablicy).

- Jest to wtedy tablica wskaźników, czyli tablica adresów pewnych istotnych w punktu widzenia programu zmiennych.

- **Przykład:**
  - Mamy wiele wierszy tekstu o różnej długości i chcemy je posortować (okrojona wersja programu ***sort*** w systemie Unix).
  - Przykładowe rozwiązanie: umieścić je wszystkie w jednej długiej tablicy znaków, jeden tekst za drugim (czyli np. 3 łańcuchy znaków `"ala" "ma" "kota"` umieścimy w jednym: `"ala\0ma\0kota"`) – niewygodne (przestawianie fragmentów tekstu wiąże się z rozsuwaniem i zsuwaniem tablicy).
  - Inne rozwiązanie:
    - Wykorzystanie tablicy wskaźników.
    - Wiersze tekstu do posortowania umieścimy jeden za drugim w dużej tablicy znakowej i każdy wiersz będzie dostępny za pomocą wskaźnika do pierwszego znaku.
    - Te wskaźniki umieścimy w innej tablicy.

Tablice wskaźników

- Wiersze porównać można za pomocą bibliotecznej funkcji ***strcmp*** (przekazując do niej wskaźniki na dwa porównywane łańcuchy znaków/wiersze tekstu).
- Gdy trzeba zamienić ze sobą miejscami dwa wiersze tekstu wystarczy zamienić miejscami ich wskaźniki w tablicy wskaźników. Nie trzeba zamieniać miejscami samych tekstów.
- Sortowanie odbywa się więc na tablicy wskaźników (Zamieniamy miejscami jeden wskaźnik z drugim, zależnie od tego, który wskazuje na tekst zaczynający się na a, który na b, c itd.).

- Najwyższy możliwy poziom abstrakcji pseudo-kodu programu:
  - przeczytaj wszystkie wiersze z wejścia
  - uporządkuj je
  - wypisz wiersze w odpowiedniej, posortowanej kolejności

```c
#include <stdio.h>
#include <string.h>
#define MAXLINES 5000 /* maksymalna liczba linii do posortowania */
#define MAXLEN 1000 /* max length of any input line */

//PROTOTYPY FUNKCJI:
char *lineptr[MAXLINES]; /* tablica wskaznikow do linijek tekstu */
int readlines(char *lineptr[], int nlines);
void writelines(char *lineptr[], int nlines);
void qsort(char *lineptr[], int left, int right);
char line[MAXLEN]; /* aktualna linia tekstu */
char longest[MAXLEN];
int getline(char line[], int maxline);

int max; /* maksymalna dlugosc tablicy znakow do danego momentu */

/* sort input lines */
main(){
 int nlines; /* liczba linii do posortowania */
 if ((nlines = readlines(lineptr, MAXLINES)) >= 0){
  qsort(lineptr, 0, nlines-1);
  writelines(lineptr, nlines);
  return 0;
 } else {
  printf("error: input too big to sort\n");
  return 1;
 }
}

/* readlines: wczytaj linie tekstu */
int readlines(char *lineptr[], int maxlines){
 int len, nlines;
 char *p, line[MAXLEN];
 nlines = 0;
 while ((len = getline(line, MAXLEN)) > 0){
  // jeśli nlines >= maxlines lub nie udało się przydzielić pamięci
  if (nlines >= maxlines || (p = malloc(len)) == NULL){
   return -1;
  } else {
   line[len-1] = '\0'; /* delete newline */
   strcpy(p, line);
   lineptr[nlines++] = p;
  }
 }
 return nlines; /* liczba wierszy tekstu */
}

/* writelines: wypisz linie */
void writelines(char *lineptr[], int nlines){
 int i;
 for (i = 0; i < nlines; i++) printf("%s\n", lineptr[i]);
} /* lineptr[i] to wskaźnik do pierwszego znaku i-tego wiersza tekstu*/
```

```c
/* getline: pobierz linijke tekstu z klawiatury */
int getline(char s[],int lim){
 int c, i;
 for (i=0; i < lim-1 && (c=getchar())!=EOF && c!='\n'; ++i) {
 /* dopóki i < lim - 1 i nie wpisano znaku \n lub koniec bufora
    wczytuje kolejne znaki budujące pojedynczy wiersz tekstu do
    posortowania */
  s[i] = c;
 }
 if (c == '\n'){
  s[i] = c;
  ++i;
 }
 s[i] = '\0';
 return i;
}

/* qsort: sortuj v[left]...v[right] w rosnacym porzadku */
void qsort(char *v[], int left, int right){
 int i, last;
 void swap(char *v[], int i, int j);
 if (left >= right) /* do nothing if array contains */
  return; /* fewer than two elements */
 // piwot przeniesiony na początek przedziału, indeks left
 swap(v, left, (left + right)/2); // piwot to element środkowy
 last = left; // indeks wskazujący miejsce zamiany
for (i = left+1; i <= right; i++){
  if (strcmp(v[i], v[left]) < 0) /* porównanie tekstów */
   swap(v, ++last, i); /* zamiana tekstów w tablicy wskaźników */
 }
 swap(v, left, last); // przeniesienie piwotu na właściwe miejsce
 qsort(v, left, last-1);
 qsort(v, last+1, right);
}

/* swap: zamien v[i] z v[j] */
void swap(char *v[], int i, int j){
 char *temp;
 temp = v[i];
 v[i] = v[j];
 v[j] = temp;
}
```

Tablice wielowymiarowe

- W języku C występują prostokątne tablice wielowymiarowe, ale są rzadziej stosowane niż tablice wskaźników.
- Tablice wielowymiarowe są to tablice, których elementami są inne tablice.
- Przykład tablicy dwuwymiarowej (w języku C tablica dwuwymiarowa jest w rzeczywistości tablicą jednowymiarową, w której każdy z elementów jest tablicą):

```
int tab[4][2];          // tab jest tablicą 4 elementów, z których każdy
                        // jest dwuelementową tablicą typu int. Inaczej 4
                        // wiersze po 2 elementy
```

  - Poszczególne elementy tak zdefiniowanej tablicy:

tab[0][0] tab[0][1] tab[1][0] tab[1][1] tab[2][0] tab[2][1] tab[3][0] tab[3][1]

- Uwaga - błędny zapis `tab[i, j]` kompilator zinterpretuje jako `tab[j]`, ponieważ operator wyliczenia ',' jest lewostronnie łączny.
- Inicjalizacja tablicy dwuwymiarowej – listą wartości początkowych umieszczonych w nawiasach klamrowych (każdy wiersz odpowiednią podlistą):

```
int tab[4][2] = {{0, 1}, {2, 3}, {4, 5}, {6, 7}};
int tab[4][2] = {0, 1, 2, 3, 4, 5, 6, 7}; // niezalecane
```

Tablice wielowymiarowe

- **Przykłady:**

```
float MACIERZ[10][20];
/* 10 wierszy, 20 kolumn */
MACIERZ[nw][nk] // nw – liczba wierszy, nk – liczba kolumn


const int kwa = 30;
long KWADRAT[kwa][kwa];

for (int i = 0; i < kwa; i ++)
     for (int j = 0; j < kwa; j ++)
    KWADRAT [ i ] [ j ] = i == j ? 0 : 1;
// główna przekątna zostanie wyzerowana, pozostałe elementy == 1
```

Tablice wielowymiarowe

- **Przykłady:**

```
int BB[2][3] = { { 1, 2, 3} , {4, 5, 6} };
/*
 BB[0][0] == 1, BB[0][1] == 2, BB[0][2] == 3
 BB[1][0] == 4, BB[1][1] == 5, BB[1][2] == 6
*/

float CC[3][2] = { { 8.5 } , { 3.2 } };
/*    CC[0][0] == 8.5, CC[0][1] == 0
      CC[1][0] == 3.2, CC[1][1] == 0
      CC[2][0] == 0  , CC[2][1] == 0   */

long LL[2][3];
sizeof LL;                    // (2 * 3) * 4 bajty = 24
sizeof LL[0];                 // 3 (wiersz ma 3 elementy) * 4 bajty = 12
sizeof LL[0][0];              // 4 bajty
```

Tablice wielowymiarowe

- **Wczytywanie macierzy wierszami:**

```
const int Wie = 10, Kol = 5;
int MM [ Wie ][ Kol ];

for (int i = 0; i < Wie; ++i)
     for (int j = 0; j < Kol; ++j)
               scanf("%d", & MM[ i ][ j ];

// to samo co powyżej, ale z wykorzystaniem wskaźników
int ile = Wie * Kol, k;
int *p = &MM[0][0];

k = 0;
// tablica dwuwymiarowa to w rzeczywistości tablica jednowymiarowa
while (k++ < ile)
     scanf("%d", p++);
```

Tablice wielowymiarowe

- **Przykład tablicy trójwymiarowej:**

```
float TTT[5][10][20];
// tablica trójwymiarowa składająca się z 5-ciu macierzy
// o 10 wierszach i 20 kolumnach każda

for (int mac = 0; mac <  5; mac ++)
     for (int wie = 0; wie < 10; wie ++)
               for (int kol = 0; kol < 20; kol ++)
               TTT[mac][wie][kol] = 1.0F;
```

Tablice wielowymiarowe – Przykład 1

```
int main(){
     double A[100], B[100], C[100];
     int n = 102;
     int i;

     while (n > 100 || n < 1){
           printf("Podaj dlugosc tablicy [1-100]: ");
           scanf("%d", &n);
     }

     printf("Wprowadz %d elementow tablicy A (liczby rzeczywiste) : ", n);

     for (i = 0; i < n; ++i)
           scanf("%lf", &A[i]);

     printf("Wprowadz %d elementow tablicy B (liczby rzeczywiste) : ", n);

     for (i = 0; i < n; ++i)
           scanf("%lf", &B[i]);
```

Tablice wielowymiarowe – Przykład 1

```
    for (i = 0; i < n; ++i)
        if (A[i] > B[i])
                C[i] = 2 * A[i] + B[i] + 1;
        else
                C[i] = A[i] - B[i] - 1;

    for (i = 0; i < n; ++i)
        printf("C[%d] = %8.2lf\n", i, C[i]);

    return 0;

}
```

Tablice wielowymiarowe – Przykład 2

- **Przykład** - Oblicz sumę elementów dolnego trójkąta macierzy kwadratowej i górnego trójkąta macierzy kwadratowej (elementy głównej przekątnej nie są sumowane)

```
int main( ){
      int Rozmiar = 200;
      float Macierz[100][100];
      int Wiersz, Kolumna;
      float SumaGorna = 0.0f, SumaDolna = 0.0f;

      while (Rozmiar < 1 || Rozmiar > 100){
      printf("\nPodaj rozmiar macierzy : ");
                scanf("%d",&Rozmiar);
      }

      printf("\nWprowadz wartosci elementow wierszami : \n");

      for (Wiersz = 0; Wiersz < Rozmiar; ++Wiersz)
                for (Kolumna = 0; Kolumna < Rozmiar; ++Kolumna)
                          scanf("%f", &Macierz[Wiersz][Kolumna]);
```

Tablice wielowymiarowe – Przykład 2

```
    for (Wiersz = 0; Wiersz < Rozmiar; ++Wiersz)
          for (Kolumna = 0; Kolumna < Rozmiar; ++Kolumna)
                    if (Kolumna > Wiersz)
                              SumaGorna += Macierz[Wiersz][Kolumna];
                    else if (Kolumna < Wiersz)
                              SumaDolna += Macierz[Wiersz][Kolumna];

          printf("\nSuma Gorna = %10.2f\nSuma Dolna = %10.2f\n",
                  SumaGorna, SumaDolna);
    return 0;

}
```

Np.: * górny trójkąt macierzy, + dolny trójkąt macierzy, - przekątna

```
  0 1 2 3 4 5 6
0 - * * * * * *
1 + - * * * * *
2 + + - * * * *
3 + + + - * * *
4 + + + + - * *
5 + + + + + - *
6 + + + + + + -
```

Tablice wielowymiarowe – Przykład 3

- **Mnożenie macierzy C = A * B**

```
const int Aw = 2;// liczba wierszy A
const int Ak = 3;// liczba kolumn A
const int Bw = 3;// liczba wierszy B
const int Bk = 2;// liczba kolumn B

int main (){
     int i, j, m;
     double s = 0.0;
     double A[Aw][Ak], B[Bw][Bk], C[Aw][Bk];
     // liczba elementów w wierszu pierwszej macierzy musi być równa liczbie
     // elementów w kolumnie drugiej macierzy
     if (Ak != Bw){
          printf("Nieprawidlowe rozmiary.\n");
          return;
     } else {
          for (i = 0 ; i < Aw ; ++i){ // wczytaj elementy tablicy A
                    for (j = 0 ; j < Ak ; ++j){
                              printf("A[%d][%d] = ", i, j);
                              scanf("%lf", &A[i][j]);
                    }
     }
```

Tablice wielowymiarowe – Przykład 3

```
printf("\n");

for (i = 0 ; i < Bw ; ++i){ // wczytaj elementy tablicy B
     for (j=0 ; j < Bk ; ++j){
               printf("B[%d][%d] = ", i, j);
               scanf("%lf", &B[i][j]);
     }
}

// mnożenie macierzy A * B. Wynik zapisywany jest w tablicy C
for (i = 0 ; i < Aw ; ++i)
     for(j = 0 ; j < Bk ; ++j){
               s = 0.0;
               for (m = 0 ; m < Ak ; ++m){ // wiersz z A razy kolumna z B
                         s += (A[i][m] * B[m][j]);
               }
               C[i][j] = s; // s - suma dla wiersz razy kolumna
}
```

Tablice wielowymiarowe – Przykład 3

```
    printf("\nMacierz A\n\n"); // wypisz elementy tablicy A
    for (i = 0 ; i < Aw ; ++i){
          printf("\n");
          for (j = 0 ; j < Ak ; ++j) printf("%5.2lf\t",  A[i][j]);
    }

    printf("\n\nMacierz B\n\n"); // wypisz elementy tablicy B
    for (i = 0 ; i < Bw ; ++i){
          printf("\n");
          for (j = 0 ; j < Bk ; ++j) printf("%5.2lf\t", B[i][j]);
    }

    printf("\n\nMacierz C\n\n"); // wypisz elementy tablicy C
    for (i = 0 ; i < Aw ; ++i){
          printf("\n");
          for (j = 0 ; j < Bk ; ++j) printf("%5.2lf\t", C[i][j]);
    }
    printf("\n\n");
 }
 return 0;
}
```

Tablice wielowymiarowe – Przykład 4

- Program symuluje działanie bufora cyklicznego do przechowywania liczb *int* za pomocą tablicy. Długość tablicy powinna wynosić 2**N (2^N), aby łatwo było sprawdzać przekroczenie dopuszczalnej wartości indeksu.

```
int main (){
     const int Dlugosc = 4;// 2 ** N
     const int Maska = Dlugosc - 1;

     int Bufor [ 4 ] = {0}; // wypełnij tablicę zerami
                       // indeksy tablicy Bufor :
     int Glowa = 0;     // pierwszy pusty (miejsce na nowy element)
     int Ogon  = 0;     // pierwszy do odczytu (element do odczytania)
     int Pom;

     char Polecenie;
     int JeszczeRaz = 1;

     while (JeszczeRaz){
          printf("\nWybierz polecenie [R, W, Q] : ");
          fflush(stdin);
          scanf("%c", &Polecenie);
```

Tablice wielowymiarowe – Przykład 4

```
switch(Polecenie & 0x5F){
    case 'R' : if (Glowa == Ogon)
                        printf("\nBufor pusty"); // warunek pustego bufora
               else {
                        // odczyt wg. ogona i zwiększenie go o 1
                        printf("\n>%4d",Bufor[Ogon++]);
                        Ogon &= Maska; // ewentualne zapętlenie ogona
                                 // dla Ogon > 3 przypisze do Ogon wartość 0
                                 // Ogon = 4 to      00000100
                                 // Maska = 3 to     00000011
                                 // Ogon = Ogon & Maska to 0
               }
               break;
    case 'W' : Pom = Glowa; // kopia głowy
               Pom++; // zwiększenie kopii
               Pom &= Maska; // ew. zapętlenie
               // przy próbie zapisania do bufora więcej niż czterech liczb
               // instrukcja ta spowoduje przypisanie do Pom wartości równej 0
               // (dla Pom <= 3 wartość bez zmian, dla Pom > 3 wartość 0)
```

Tablice wielowymiarowe – Przykład 4

```
                    if (Pom == Ogon) printf("\nBufor pelny.");
                    else {
                              printf("\nWpisz element : ");
                              // zapisanie w miejsce głowa
                              scanf("%d", &Bufor[Glowa]);
                              Glowa = Pom; // aktualizacja głowy
                    }
                    break;
          case 'Q' : JeszczeRaz = 0;
                    break;
          default : printf("\nZle polecenie [R, W, Q]\n");
        } //end switch
    } //end while

    printf("\n\n");
    return 0;
}
          // np. jeśli opcją w dodamy kolejne elementy np. 23, 34, 12,
          // to opcją r odczytamy (i jednocześnie usuniemy z bufora) elementy
          // w następującej kolejności: 23, 34, 12
```

Tablice dwuwymiarowa – tablica wierszy

- Po następujących definicjach:

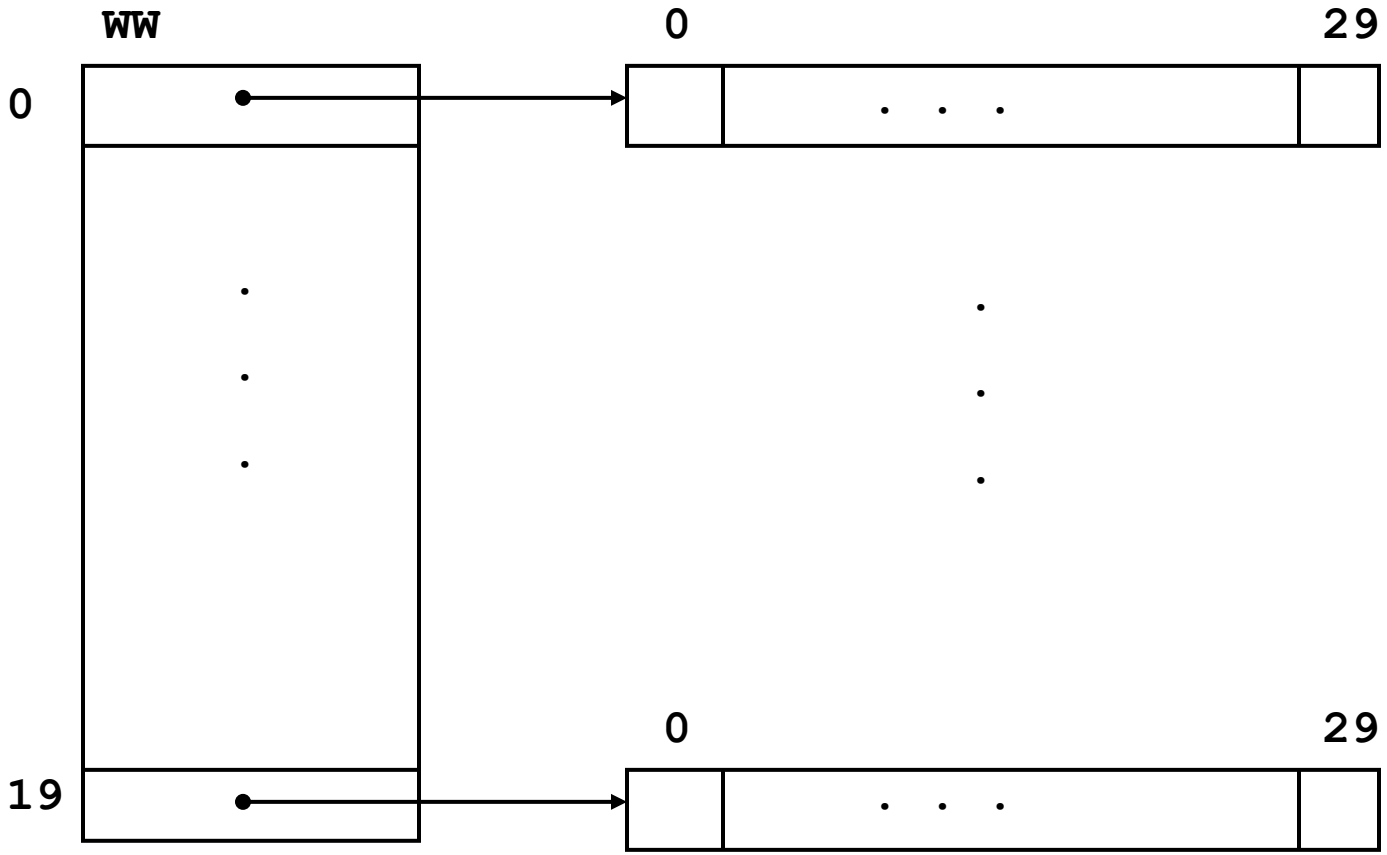
```
int a[10][20];   /* prawdziwa tablica dwuwymiarowa. Zawiera 200
                 miejsc o rozmiarze odpowiednim dla typu int. Do
                 znalezienia elementu tablicy a[wiersz][kolumna]
                 stosuje się tradycyjny przepis na wyliczenie
                 indeksu dla tablic prostokątnych:
                 20 * wiersz + kolumna */
int *b[10];      // przydziela 10 miejsc pamięci na wskaźniki i nie inicjuje ich
```

- **Przykład** tablicy wskaźników:

```
int** WW = new int*[20];          // przydziela 20 miejsc pamięci na wskaźniki
for (int i = 0; i < 20; ++i)
          WW[i] = new int[30];    // przydziela pamięć na 30 elementów typu
                                  // int. Każdy wskaźnik spod WW[i] będzie
                                  // wskazywał na tablicę 30 elementową

WW[4][5] = 17;
int x = WW[4][5];     // poprawne
```

Tablice dwuwymiarowa – tablica wierszy



Tablice dwuwymiarowe nierównomierne

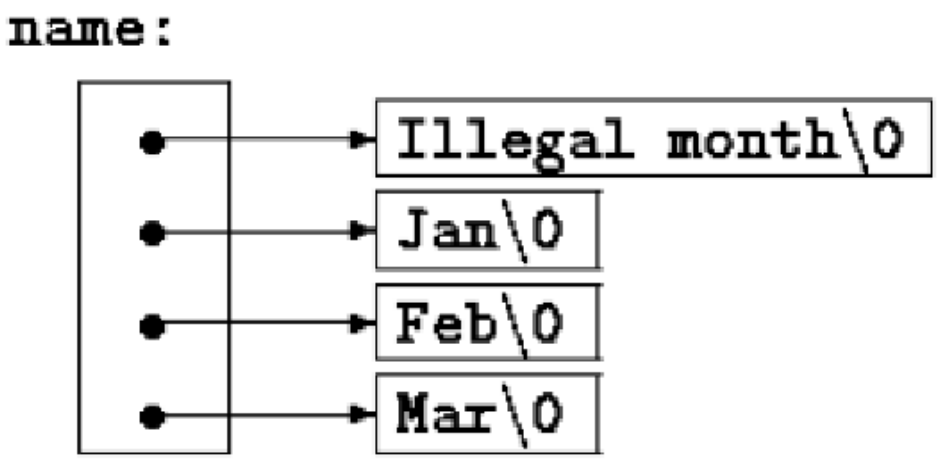
- Przewagą tablicy wskaźników jest to, że wiersze tablicy mogą mieć różne długości.
- Najczęściej wykorzystuje się je do zapamiętywania tekstów o różnych długościach.
- **Przykład:**

```
int DL [ ] = {5, 15000, 7, 2000, 3};
int N = sizeof(DL) / sizeof(DL[0]);    // licba elementów tablicy DL
int** WW = new int*[N];                // tablica wskaźników o długości N
for (int i = 0; i < N; ++i)
      WW[i] = new int[DL[i]];          // przydziela pamięć na kolejne wiersze
                                       // o długości kolejno: 5, 15000, 7, 2000, 3

WW[1][12547] = 17;                     // ten wiersz ma długość 15000, więc ok
int x = WW[1][12547];
```

Tablice dwuwymiarowe nierównomierne

```
char *name[] = { "Illegal month", "Jan", "Feb", "Mar" };
```




```
char aname[][15] = { "Illegal month", "Jan", "Feb", "Mar" };
```

aname:

Illegal month\0 Jan\0	Feb\0	Mar\0
0 15	30	45

Pytania?