Język ANSI C

część 9
tablice 2D

Jarosław Gramacki
Instytut Informatyki i Elektroniki

Przykłady podstawowych wyrażeń ze wskaźnikami

```
int Tab1D [100] = {1, 2, 3, 4};              // Tab1D[4]..Tab1D[99] == 0

int Tab2D [2][3] = {4, 5, 6, 10, 20};        // [ 4  5 6]
                                             // [10 20 0]
```

ta 2-ka jest zbędna

```
int Tab2D = {{4,  5, 6},           // dużo czytelniejszy sposób
             {10, 20}};

                                                 0              99
int Tab2D [100][100] = {{4,  5},   // wiersz 0   [4  5   0      ]
                        {10, 20, 30}}; // wiersz 1   [10 20 30      ]
                                       //            [              ]
                                       // wiersz 99  [        0     ]
```

bez tego wymiaru kompilator przyjmie tu 2

- ostatnia tablica to tablica 100. elementowa, której elementami są tablice stu elemetów (ta właściwość okażę się bardzo wygodna w języku C !!!)
- w języku C elementy tablicy umieszczane są w pamięci wierszami. W np. języku Fortran kolumnami

Tablice 2-wymiarowe jako argumenty funkcji

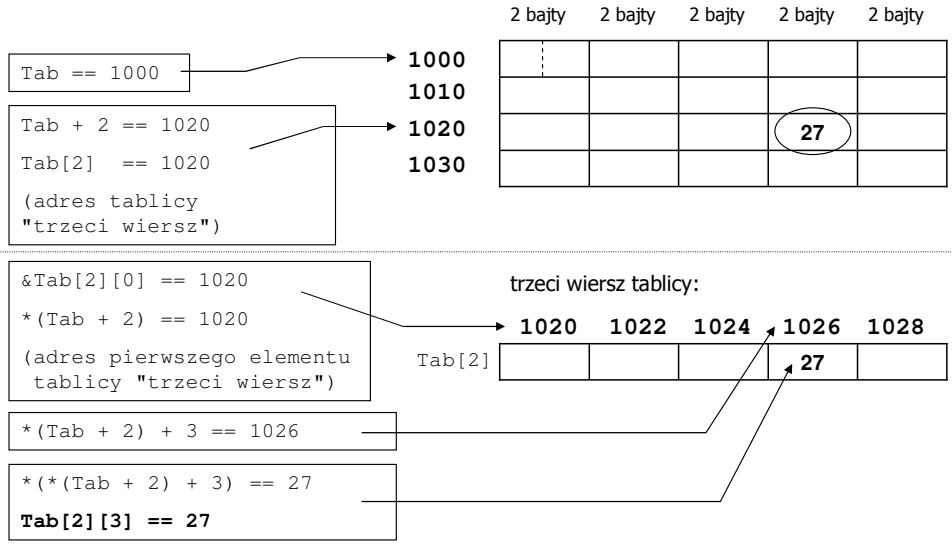
```
int MyTab2D[3][100];
void funkcja (int Tab2D [][]);     // deklaracja


void funkcja (int Tab2D [][100])   // definicja
{
...
MyTab2D [1][5] = 7;        // jak kompilator obliczy, do której komórki
                           // w pamięci się odwołać?
....
}
...

funkcja ( MyTab2D );  // wywołanie funkcji
```

bez tego (Linux):
error: invalid use of array with unspecified bounds

- do funkcji przekazujemy tablicę `int Tab2D[3][100]`
- podanie liczby kolumn jest konieczne w deklaracji i definicji funkcji. Liczbę wierszy można pominąć.
- element `[1][5]` to 105. element w "płaskiej" tablicy (licząc od zera): 1 * **100** + 5 = 105
- dla tablic więcej niż 2. wymiarowych w definicji funkcji należy podawać wszystkie oprócz ostatniego rozmiaru

Notacja wskaźnikowa dla tablic 2D

- dla uproszczenia rysunku zakładamy, że tablica zawiera 2-u bajtowe elementy typu `int` oraz stosujemy wymyślone adresy 4-ro cyfrowe
- deklaracja tablicy `int Tab[4][5]`



Notacja wskaźnikowa dla tablic 2D

– notacja wskaźnikowa dla tablic 2D powinna być używana tylko gdy to naprawdę konieczne (w praktyce dosyć rzadki przypadek)

– oba wyrażenia wskaźnikowe `Tab + 2` oraz `*(Tab + 2)` odnoszą się do tego samego adresu 1020. Po co więc to rozróżnienie ?

– różnica pomiędzy `Tab +2` oraz `*(Tab + 2)` jest w "jednostkach miary".

  – jeśli dodać 1 do `Tab + 2` otrzymamy `Tab + 3` czyli adres czwartego wiersza (dodanie 10 bajtów).

  – jeśli dodać 1 do `*(Tab + 2)` otrzymamy adres następnego elementu w wierszu (dodanie 2 bajtów)

```
Tab + 2 == 1020
Tab[2]  == 1020
(adres tablicy "trzeci wiersz")
```

```
&Tab[2][0] == 1020
*(Tab + 2) == 1020
(adres pierwszego elementu tablicy "trzeci wiersz")
```


Jedna uwaga

– do tablicy 2D można odwoływać się jak do tablicy 1D

– przykład czysto teoretyczny - nikt tak raczej nie robi !

porównaj "mapę pamięci" z sesji debuggera gdb na poprzednik wykładzie

```
int Tab2D[3][4];          // Tab2D jest typu int (*)[4]
int *ptr;

ptr = (int*) Tab2D;       // rzutowanie wskaźników konieczne !

for(i=0; i<3*4; i++)
  ptr[i] = 10;
```

Tablice wskaźników

– najczęściej używane jako tablice wskaźników do znaków

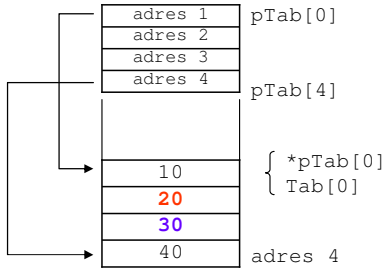
– poniżej ilustracja tablicy wskaźników do liczb całkowitych (przykład o charakterze trochę sztucznym)

```
void main(void) {
int Tab[4] = {10, 20, 30, 40};
int *pTab[4];                    // tablica czterech wskaźników do int

for(i=0; i<4; i++)
  pTab[i] = Tab + i;             // inicjacja tablicy wskaźników

for(i=0; i<4; i++)
  printf("%d ", *pTab[i]);       // drukuj liczby "przez pośrednika"

printf("\n%d %d ", *(pTab[1] + 1), *pTab[1] + 1);   // wyr. różnią się
}                                                   // tylko nawiasami
```

taka deklaracja pojawia się po raz pierwszy

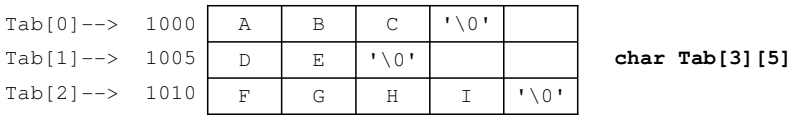
```
for(i=0; i<4; i++)
  printf("%lu ", pTab[i]);
```
gdy interesują nas adresy

```
Wynik:

10 20 30 40
30 21
```

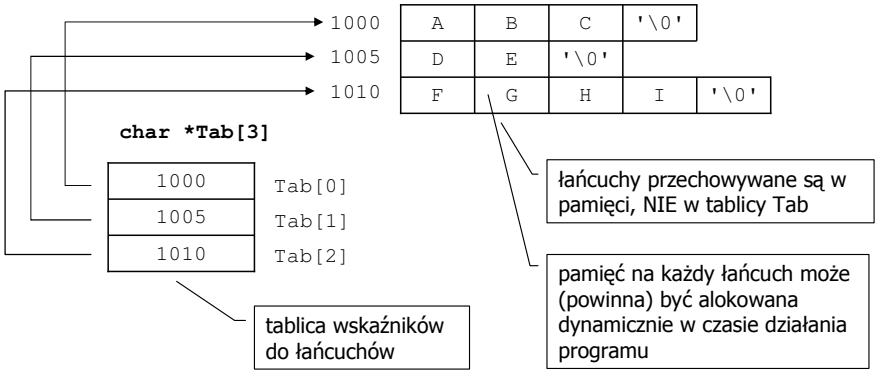


Tablice wskaźników do znaków

– stosujemy wymyślone adresy 4-ro cyfrowe



char Tab[3][5]



tablica wskaźników do łańcuchów

łańcuchy przechowywane są w pamięci, NIE w tablicy Tab

pamięć na każdy łańcuch może (powinna) być alokowana dynamicznie w czasie działania programu

Tablice wskaźników do znaków

– deklaracja połączona z inicjalizacją tablicy wskaźników do znaków

```
#include <stdio.h> #include <stdlib.h>

char *month (int);

void main (void)
{
  int numer;

  printf("Podaj numer miesiaca: ");
  scanf("%d", &numer);
  printf("Miesiac nazywa sie: %s\n ", month( numer ) );
  getch();
}

char *month (int n)  //zwraca wskaznik do nazwy n-tego miesiaca
{
  char *name[]=      // w penym sensie "dynamiczna" alokacja pamięci
    {
      "Zla nazwa !!!",
      "Styczen", "Luty", "Marzec", "Kwiecien",
      "Maj", "Czerwiec", "Lipiec", "Sierpien", "Wrzesien",
      "Pazdziernik", "Listopad", "Grudzien"
    };
  return (n<1 || n>12) ? name[0] : name[n];
}
```


Tablice wskaźników do znaków

– deklaracja połączona z inicjalizacją tablicy wskaźników do znaków

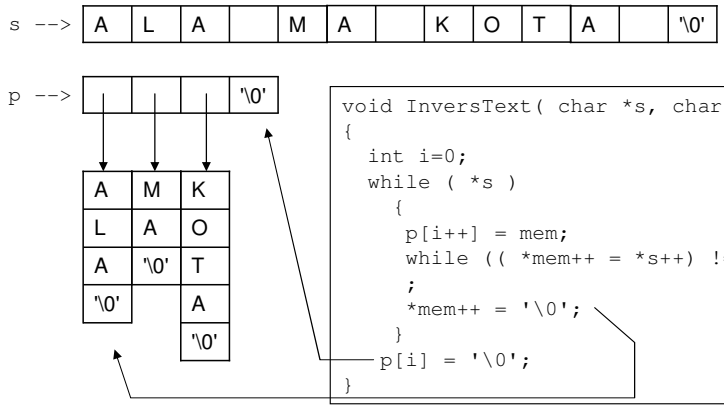
```
// Co DOKŁADNIE zostanie wydrukowane na ekranie ?
#include <stdio.h>

void main(void)
{
   char *list[5] =
     {"Jarek",
      "Piotr",
      "Wojciech",
      "Tomek",
      "Karolina" };

   for(i=0,j=4; i<5; i++,j--)
    {
      printf("%-8s ", *(list+i));
      printf("-%c -%d\n", list[i][j], j);
    }
}
```

notacja jak dla zwykłej tablicy 2D

```
// Odpowiedź:

Jarek    -k -4
Piotr    -t -3
Wojciech -j -2
Tomek    -o -1
Karolina -K -0
```

Tablice wskaźników do znaków

– odwróć kolejność słów (NIE liter !) w zdaniu



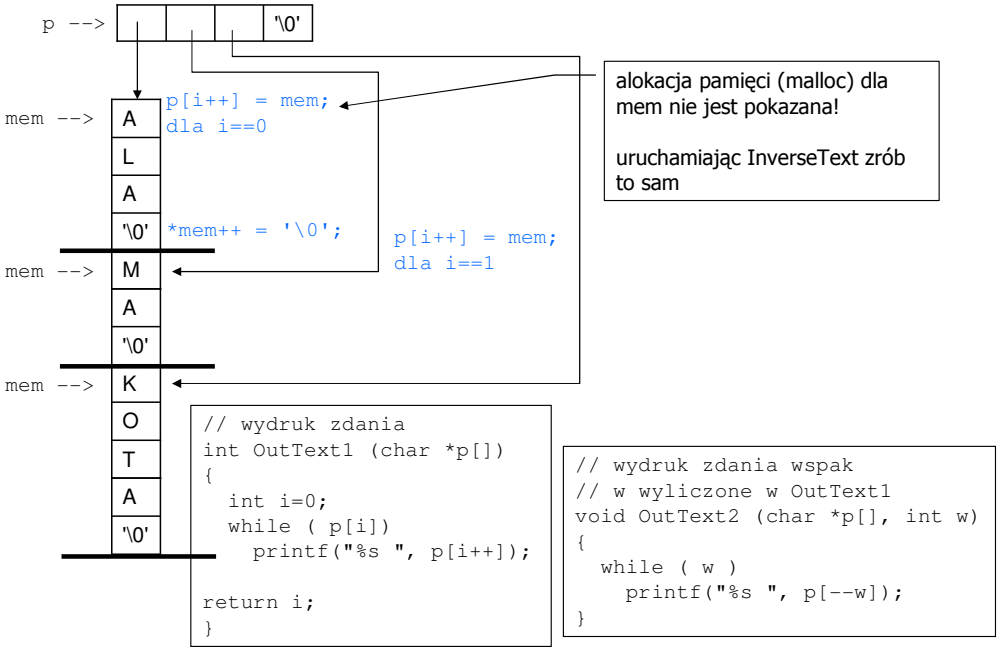
– rozdzielimy zdania na słowa i wskaźniki do słów zapiszemy w tablicy wskaźników do znaków

– założenie: separatorem słów są spacje (nawet po ostatnim słowie w zdaniu)

– rozmiar `mem`: `strlen(s)`. Alokacja pamięci poza `InversText()`

Tablice wskaźników do znaków

– odwróć kolejność słów w zdaniu - szkic rozwiązania

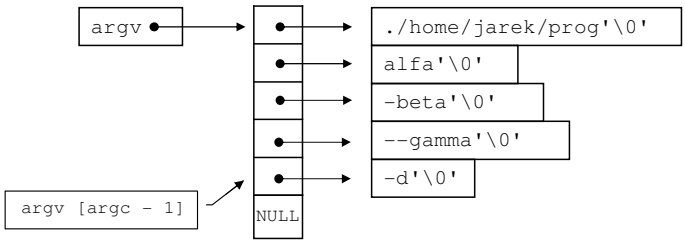


Argumenty wywołania programu

```
// prog.c
int main (int argc, char* argv[])
{
...
}
```

każdy z argumentów może nazywać się dowolnie, ale skoro wszyscy stosują takie identyfikatory ...

```
// wywołanie programu

jarek@mykonos:~$    ./home/jarek/prog   alfa      -beta     --gamma     -d
                          argv[0]       argv[1]   argv[2]   argv[3]   argv[4]

                    argc == 5
```

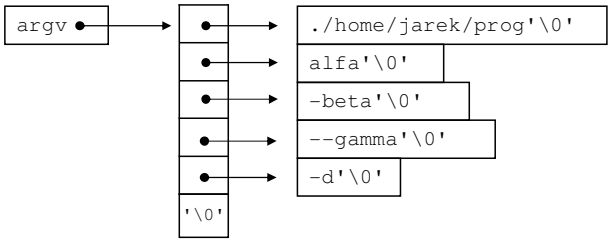


Argumenty wywołania programu

– kilka wyrażeń

`argv`	wsk. do początku tablicy tekstów argumentów (`argv[0]`)
`++argv`	teraz wskazujemy na argument `argv[1]` a nie na `argv[0]`
`*argv`	wsk. do argumentu (znaków tworzących argument)
`*++argv`	wsk. do argumentu (`++` jest tylko "narzędziem" do przesuwania się)
`(*++argv)[0]`	pierwszy znak tego argumentu. Nawiasy konieczne. Operator `[]` ma wyższy priorytet niż `*` oraz `++`
`*++argv[0]`	gdy wykonane pętli to: maszerujemy wzdłuż ciągu znaków argumentu (tu: ścieżki z programem)
`== *(++(argv[0]))`	

nawiasy wymuszone przez kompilator



Argumenty wywołania programu

– wygodne parsowanie opcji programu z użyciem biblioteki **getopt**

– http://www.gnu.org/software/libc/manual/html_node/Getopt.html

```
// Echo argumentów

int main (int argc, char* argv[]) {
 while (--argv > 0)
   printf("%s\n", *++argv);        // 1. przsuń się, 2. pobierz wartość
   printf("%s\n", *(argv++);       // razem z nazwą programu i ze scieżką
}

int main (int argc, char* argv[]) {
int i;

for(i=1; i < argc; i++)            // i=1 gdyż pomijamy nazwę programu
  printf("%s\n", argv[i]);         // wydruk "w pionie"

for(i=1; i < argc; i++)
  printf("%s%s", argv[i], (i < argc-1) ? "_" : ""); // wydruk "w poziomie"

getchar();
return 1; }
```

priorytet	łączność
[]	L
++ *	P

Notacja wskaźnikowa dla tablic 2D

– wydaje się, że lepiej nie nadużywać notacji wskaźnikowej (łatwo o pomyłkę) i stosować notację tablicową

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main (int argc, char* argv[], char* env[])
{
  int i, temp;

if (argc == 1) exit(1);

// Argumenty od tylu
temp = argc;              // kopia argc
while( temp )
  printf("Od tylu: %s\n", argv[--temp]);

// Argumenty od przodu
temp = argc;
for(i=0; temp--; i++)
  printf("%Od przodu 1: %s\n", argv[i]);

// Zmienne srodowiskowe (pierwsze 10)
for (i = 0; (env[i] != NULL && i<10) ; i++)
  printf("Env. %d-->%s\n", i, env[i]);
                                  c.d.n
```

dostęp do zmiennych środowiskowych

Notacja wskaźnikowa dla tablic 2D

```
c.d

// Pierwszy argument wywołania znak po znaku
printf("Pierwszy argument wywolania znak po znaku: ");
for(i=0; i<strlen(argv[1]); i++)
  printf("%c",argv[1][i]);
  printf("%\n");

// Drugi znak pierwszego argumentu
printf("Drugi znak trzeciego argumentu: %c\n",argv[2][1]);
printf("Drugi znak trzeciego argumentu: %c\n", *(*(argv+2)+1) );

// Argumenty od przodu inaczej, ale UWAGA !!!
// Gdyby użyć tej konstrukcji gdzieś w środku przykładu, to będzie źle, gdyż
// pamiętać trzeba, że *++argv przesuwa wskaźniki na trwałe.
// Gdy jesteśmy na końcu programu, to dalej argv nie jest już nam
// więcej potrzebne

temp = argc;
while( temp-- )
  printf("Od przodu 2a: %s\n", *argv++);   // bez ścieżki dostępu

getchar();
return 1;
}
```


Argumenty wywołania programu

```
// echo argumentów
#include <stdio.h>
void main (int argc, char *argv[]) {
  int i;
  for(i=0; i<argc; i++)
    printf("%s\n", argv[i]); }

// wywołanie 1:
jarek@mykonos:~$ ./prog  abc '*' xyz

// wynik:
./prog
abc
*
xyz

// wywołanie 2:
jarek@mykonos:~$ ./prog  abc * xyz

// wynik:
./prog
abc
Lab_zima2006
prog
argv.c
tab-na-plik.c
xyz
```

zawartość katalogu, z którego wywołano program !

Argumenty wywołania programu

```
./prog -p Piotrek                    wynik: HELLO_Piotrek
./prog -p -k Piotrek                 wynik: HELLO_Piotrek_BYE
./prog Piotrek                       wynik: Piotrek
./prog -pk Piotrek                   wynik: HELLO_Piotrek_BYE
./prog -p -k Piotrek Wojtek          wynik: info o błedzie wywołania
./prog -p -x Piotrek                 wynik: info o błedzie wywołania
```

```
int main (int argc, char *argv[]) {
  int i=0, j=0, err=0, count=0;
  char pocz[10]="", kon[10]="";
  char c, dest[30];

  for (i=1; i < argc; i++)
   if (argv[i][0] == '-') // parametry programu zaczynają się od '-'
    {
      count++;
      for (j=1; c = argv[i][j]; j++)
      switch(c)
      {
        case 'p': strcpy (pocz, "HELLO_");
        break;
        case 'k': strcpy (kon, "_BYE");
        break;
        default : { printf ("Nieznana opcja: %s\n" , argv[i]);
                    err = 1; }
        break;
      }
    } // if                c.d.n
```

i=1 bo: przeskocz nazwę programu


```
if ( err || argc > count+2 ) // nieznana opcja lub wiecej niz jeden argument
{
  printf ("Wywolanie programu: ");

  // wyluskanie nazwy programu z pelnej sciezki dostepu
  {
    for (i=0; argv[0][i] != '\0'; i++) // przeladaj znaki sciezki wywolania
      if (argv[0][i] == '\\')
         j = i+1;

    for (i=j; argv[0][i] != '\0'; i++) // idz po znakach nazwy programu
      printf ("%c", argv[0][i]);
  }
  printf (" [-p] [-k] [-pk] <wzorzec> \n");
  printf ("gdzie:\n   -p : dolaczyc domyslny tekst PRZED wzorcem\n");
  printf ("   -k : dolaczyc domyslny tekst ZA wzorcem\n");
  printf ("   wzorzec : dowolny tekst\n");
  }
else
  {
    strcpy (dest, pocz);
    strcat (dest, argv[argc-1]);
    strcat (dest, kon);
    printf ("Wzorzec: %s\n", argv[arg + 1]);
    printf ("Wyjscie: %s\n", dest);
  }
getchar();
return 1;
}
```

Tablice wskaźników do znaków

wersja 1

```
#include <stdio.h>
#include <string.h>       // for strcmp()
#define MAX 5             // number of strings
#define LEN 40            // maximum length of string

int main(void)    {
   int i;
   int enter = 0;           // flag: set to 1 if entry OK
   char name[40];           // storage for name typed by user
   char list[MAX][LEN] =    // array of strings
                  {  "Katrina",
                     "Nigel",
                     "Alistair",
                     "Francesca",
                     "Gustav"     };
   printf("\nEnter your name: ");
   gets(name);
   for (i=0; i<MAX; i++)
      if( strcmp( &list[i][0], name) == 0 )// if match
         {
         enter = 1;  break;                // set flag, leave loop
         }
   if ( enter == 1 )                       // if flag set
      printf("Found ...");
   else                                    // otherwise
      printf("Not Found ... ");
}
```

```
printf ("%d", sizeof(list));
200
```

lub:

```
if( strcmp( list[i], name) == 0 )
```


Tablice wskaźników do znaków

wersja 2

```
#include <stdio.h>
#include <string.h>       // for strcmp()
#define MAX 5             // number of strings
#define LEN 40            // maximum length of string

int main(void)    {
   int i;
   int enter = 0;           // flag: set to 1 if entry OK
   char name[40];           // storage for name typed by user
   char *list[MAX] =        // array of pointers to strings
                  {  "Katrina",
                     "Nigel",
                     "Alistair",
                     "Francesca",
                     "Gustav"     };
   printf("\nEnter your name: ");
   gets(name);
   for (i=0; i<MAX; i++)
      if( strcmp( list[i], name) == 0 )// if match
         {
         enter = 1;                    // set flag
         break;                        // leave loop
         }
   if ( enter == 1 )                   // if flag set
      printf("Found ...");
   else                                // otherwise
      printf("Not Found ... "); }
```

```
printf ("%d", sizeof(list));
20 !!! (5 x 4)
```

Tablice wskaźników do znaków

– "dziwny" problem z wczytywaniem wierszy

```
...
char buf[80];
char *lines[100];
int i;

for(i = 0; i < 100; i++)
{
  char *p = fgets(buf, 80, fp);
  if(p == NULL) break;
  lines[i] = p;
}
...
```

– na końcu wszystkie wiersze zawierają kopię ostatniego !!!

– Wyjaśnienie:
Pamięć przydzielono tylko na jeden wiersz. Przy każdym wywołaniu `fgets()` poprzedni wiersz jest nadpisywany.
fgets() NIE przydziela pamięci. W każdym obiegu pętli, zawsze zwraca ten sam wskaźnik, który dostała jako pierwszy argument (tu: adres tablicy `buf`)

– Usunięcie błędu:
Przydzielać pamięć dla każdego wiersza pobieranego z pliku.

Tablice wskaźników do znaków

– wprowadzanie danych do tablicy wskaźników do znaków

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_DLUG_STR   127
#define MAX_LICZBA_STR 100
//
// 1. Odczytuje ciagi znakow

// 2. Odczytane ciagi umieszcza w pamieci w  przydzielanych
//    dynamicznie obszarach

// 3. Adresy wszystkich ciagow przechowywane sa w tablicy wskaznikow

// 4. Maksymalna liczba ciagow ustalona jest Z GÓRY
//

// c.d na następnej stronie
```

– wprowadzanie danych do tablicy wskaźników do znaków

```
int main (void) {
  char str[MAX_DLUG_STR+1],        // tablica (bufor) do odczytu wiersza
       *lines [MAX_LICZBA_STR],    // tablica wskaźników do ciągów
       *wsk;                       // wskaźnik bieżącego ciągu
  int count = 0, i;

for (i=0; i<MAX_LICZBA_STR; i++)   // inicjacja tablicy wskaźników
      lines[i] = NULL;

while (count < MAX_LICZBA_STR){
      printf("Podaj ciag nr %d: ", count+1);
      gets(str);                   // odczyt wierszy i przydział pamięci
      if ( strlen(str) )           // w zależności od długości wiersza
        {
        if((wsk = (char*)malloc( strlen(str) + 1) ) != NULL)
          strcpy (lines[count++] = wsk, str);
        // 1. wskaznik do zaalokowanego obszaru
        // 2. wstaw wiersz do tego obszaru
         else
          { printf("błąd przydziału pamięci \n"); exit(-1);    }
         }
      else break;                  // koniec pętli gdy ciąg pusty
}
  printf("\n\nKoniec wprowadzania \n");
  printf("\n\nLista wprowadzonych ciagow \n");
  for(i=0; i<count; i++)  printf("-- ciag numer %d: %s\n", i+1, lines[i]);
return 0;   }
```


Tablice wskaźników do znaków

```
#define MAX_DLUGOSC   81          // maksymalna długość napisu
#define MAX_ILOSC 30              // maksymalna liczba napisów

// Odczytuje ciągi znaków, sortuje je alfabetycznie
int main (void)
{
  char  imie[MAX_ILOSC][MAX_DLUGOSC],      // tablica imion ("statyczna")
       *lines[MAX_ILOSC],                  // tablica wskaźników do imion
       *wsk;                               // tymczasowy wskaźnik
  int count=0, i, j                        // liczba ciągów

  while ( count < MAX_ILOSC )              // wprowadzanie danych
    {
      printf( "Podaj ciag nr %d: ", count+1 );
      gets( imie[count] );
      if ( strlen(imie[count]) == 0 )  break;
      lines[count++] = imie[count];                  // UWAGA NA TA LINIE
    }
  for(i=0; i<count-1; i++) // sortowanie przez przestawianie wskaźników
    for(j=i+1; j<count; j++)
      if(strcmp(lines[i], lines[j]) > 0)
        {
          wsk = lines[j];
          lines[j] = lines[i];
          lines[i] = wsk;
        }
  // wydruk jak w przykładzie poprzednim
return 0; }
```

count-y wiersz tablicy `imie`

- przerywnik ...

Makra

– uwaga na temat makr z parametrami

```
// demonstrates macros, using printf() statements

#include <stdio.h>
#define PR(n) printf("%.2f\n", n);     // macro definition
#define PR (n) printf("%.2f\n", n);    // ZLE
         ^^^
void main(void)
   {
   float num1 = 27.25;
   float num2;

   printf("\n");
   num2 = 1.0 / 3.0;
   PR(num1);                          // calls to macro
   PR(num2);
   }
```

Makra

– uwaga na temat makr z parametrami

```
# define SUM(x,y) x + y
...
ans = 10 * SUM(3,4);      //ZLE ans = 10 * 3 + 4;

# define SUM(x,y) (x + y)
...
ans = 10 * SUM(3,4);      //DOBRZE ans = 10 * (3 + 4);



# define MULT(x,y) (x * y)
...
ans = MULT(2+3,4);                //ZLE ans = (2+3*4);

# define MULT(x,y) ((x) * (y))
...
ans = MULT(2+3,4);                //DOBRZE ans = ((2+3)*4);
```