Reprezentacja symboli w komputerze

Znaki i łańcuchy znakowe

Ostatnia aktualizacja: 26 stycznia 2026

ASCII

The American Standard Code for Information Interchange, 1968 r.

- 7 bitów, liczby z zakresu 0-127
- litery (alfabet angielski) `[a-zA-Z]`
- cyfry `[0-9]`,
- znaki przestankowe, np. `,.;'`
- inne symbole drukowalne, np. `*^$`
- białe znaki: spacja, tabulacja, ...
- polecenia sterujące: znak nowej linii `\n`, nowej strony, koniec tekstu, koniec transmisji,...
- 8-bitowe rozszerzenia ASCII: cp1250, latin2, ...

n	znak	n	znak	n	znak	n	znak	n	znak	n	znak
0	NUL '\0'	22	SYN	44	,	66	B	88	X	110	n
1	SOH	23	ETB	45	-	67	C	89	Y	111	o
2	STX	24	CAN	46	.	68	D	90	Z	112	p
3	ETX	25	EM	47	/	69	E	91	[	113	q
4	EOT	26	SUB	48	0	70	F	92	\	114	r
5	ENQ	27	ESC	49	1	71	G	93	]	115	s
6	ACK	28	FS	50	2	72	H	94	^	116	t
7	BEL '\a'	29	GS	51	3	73	I	95	_	117	u
8	BS '\b'	30	RS	52	4	74	J	96	‘	118	v
9	HT '\t'	31	US	53	5	75	K	97	a	119	w
10	LF '\n'	32	SPACE	54	6	76	L	98	b	120	x
11	VT '\v'	33	!	55	7	77	M	99	c	121	y
12	FF '\f'	34	"	56	8	78	N	100	d	122	z
13	CR '\r'	35	#	57	9	79	O	101	e	123	{
14	SO	36	$	58	:	80	P	102	f	124	|
15	SI	37	%	59	;	81	Q	103	g	125	}
16	DLE	38	&	60	<	82	R	104	h	126	~
17	DC1	39	’	61	=	83	S	105	i	127	DEL
18	DC2	40	(	62	>	84	T	106	j		
19	DC3	41	)	63	?	85	U	107	k		
20	DC4	42	*	64	@	86	V	108	l		
21	NAK	43	+	65	A	87	W	109	m		

- typ `char`, 1 bajt, liczba całkowita $[-128, 127]$
- **stała znakowa** - znak zapisany w apostrofach, np.:
  - `'A'` to liczba 65,
  - `'A'+1` to liczba 66 (kod litery `'B'`),
  - nowy wiersz `'\n'` to liczba 10,
  - tabulator `'\t'` to liczba 9,
  - znak `'\0'` ma wartość 0
  - powrót karetki `'\r'`
    W systemie MS Windows koniec linii to dwa bajty `\r\n`
  - znaki o specjalnym zapisie:
    ukośnik w lewo `'\\'`,
    apostrof `'\''`,
    cudzysłów `'\"'`,
    znak zapytania `'\?'`
- Uwaga: `"A"` nie jest pojedynczym znakiem lecz tablicą znaków!

```c
#include<stdio.h>

int main()
{
    char a;

    printf("Podaj znak: ");
    scanf("%c",&a);

    printf("znak %c, dec %d, oct %o, hex %x\n",a,a,a,a);
    return 0;
}
```

☞ char.c

Wykrywanie małej litery

```
int islower(int x)
{
    if(x >= 'a' && x <= 'z' ) return 1;
    return 0;
}
```

Zamiana małej litery na dużą

```
int toupper(int x)
{
    if( islower(x) ) x = x - ('a' - 'A');
    return x;
}
```

Plik nagłówkowy: `ctype.h`

Klasyfikacja znaków - funkcja zwraca 0 lub 1	
`int isalnum(int c)`	litery alfabetu lub cyfry `[A-Za-z0-9]`
`int isalpha(int c)`	litery alfabetu `[A-Za-z]`
`int islower(int c)`	małe litery alfabetu `[a-z]`
`int isupper(int c)`	wielkie litery alfabetu `[A-Z]`
`int isdigit(int c)`	cyfry `[0-9]`
`int isgraph(int c)`	znaki drukowalne (ze spacją)
`int isspace(int c)`	znaki białe
`int isprint(int c)`	znaki drukowalne (bez spacji)
`int ispunct(int c)`	znaki przestankowe (drukowalne bez liter i cyfr)

Zamiana znaków	
`int tolower(int c)`	z wielkiej litery na małą
`int toupper(int c)`	z małej litery na wielką

Przykłady w C: zamiana wielkości znaków

```c
#include <stdio.h>
#include <ctype.h>

int main()
{
    int a;

    while( (a = getchar()) != EOF )
        putchar(toupper(a));

    return 0;
}
```

☞ toupper2.c

`int getchar()`
zwraca pojedynczy znak ze standardowego wejścia lub `EOF`

`void putchar(int c)`
wypisuje znak

`EOF` (*end of file*)
bajt końca pliku

```
toupper2 < plik.txt
```

`EOF` w terminalu:
`Ctrl+Z` (Windows)
`Ctrl+D` (Unix/Linux)

- **Łańcuch znakowy** (*string*, napis) - skończony ciąg symboli alfabetu
- W języku C brak typu `string`, występuje w C++
- Łańcuch to tablica zawierająca ciąg znaków zakończony znakiem `'\0'` (wartość liczbowa zero)

A	l	a		m	a		k	o	t	a	\0

- Dostęp do znaku jak w tablicach: `t[i]`
- Stała napisowa (literał znakowy): `"Ala ma kota"`
- Stała znakowa `'A'` to liczba 65
- Stała napisowa `"A"` to tablica

A	\0

Inicjowanie stałych napisowaych

Zmienna wskaźnikowa wskazująca na stały napis (nie można go zmodyfikować)

```
char *s1 = "nieciekawy fragment tekstu.";
```

Tablica zawierająca napis (może być modyfikowana)

```
char s2[]="Ala ma kota";
```

Tablica zawierająca napis "Ala"

```
char s3[] = {'A', 'l', 'a', '\0'};
```

```c
#include<stdio.h>

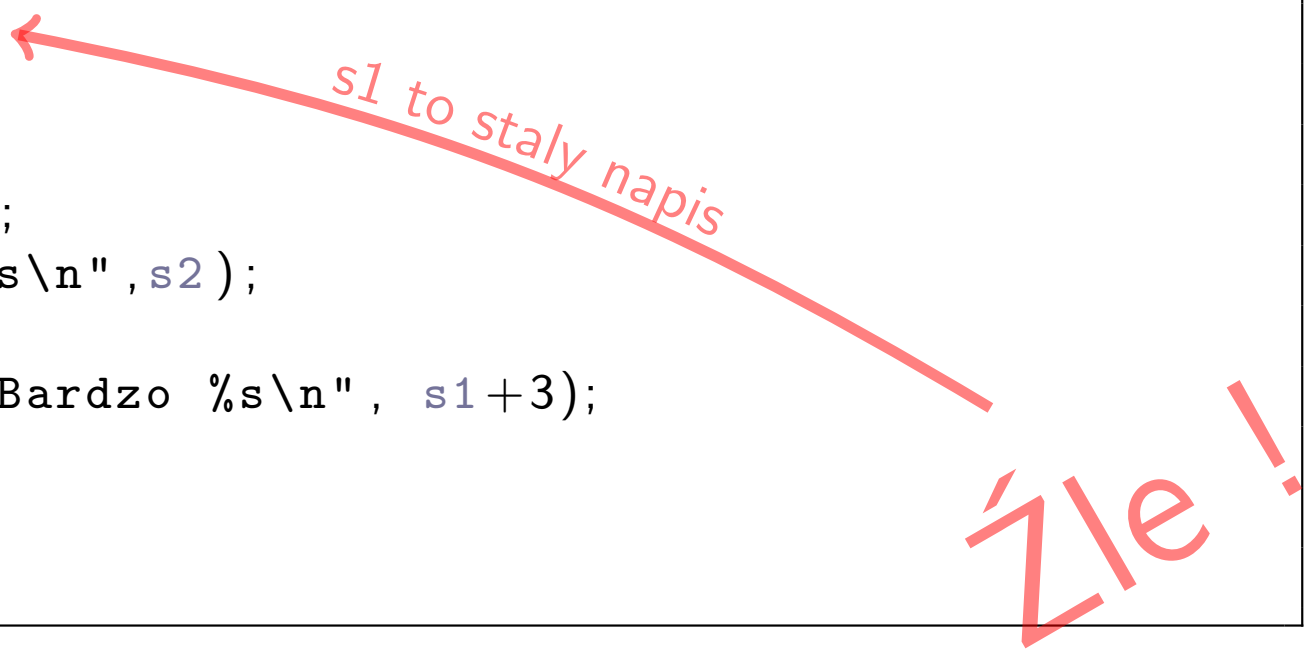
int main()
{
    char *s1="nieciekawy fragment tekstu.";
    char s2[]="Ala ma kota";

    s1[0]='N';
    s2[0]='E';

    printf(s1);
    printf(" %s\n",s2);

    printf("\nBardzo %s\n", s1+3);

    return 0;
}
```



`scanf("%s",...);`

```
#include<stdio.h>
int main()
{
    char text[10];
    scanf("%s", text);
}
```

- niebezpieczeństwo przepełnienia tablicy
- `scanf("%10s", text);` poprawnie, wczyta max. 10 znaków
- odczytuje tylko pierwszy wyraz napisu, do wystąpienia białego znaku

```
char *gets(char *s);
```

- funkcja `gets()` czyta linię tekstu ze standardowego wejścia i umieszcza ją w tablicy `s`
- Uwaga: brak zabezpieczenia przed przepełnieniem tablicy, nie należy stosować tej funkcji

```
#include<stdio.h>
int main()
{
    char text[10];
    gets(text);
}
```

`char *fgets(char *s, int n, FILE *f);`

- czyta linię tekstu (ale nie więcej niż `n` znaków) ze strumienia `f` i umieszcza tekst w tablicy `s`
- obowiązkowe podanie parametru `n` - użycie funkcji jest bezpieczne
- standardowy strumień wejściowy to `stdin`
- znak `'\n'` również umieszczony jest na końcu napisu

```
#include<stdio.h>
int main()
{
    char text[10];
    fgets(text, 10, stdin);
}
```

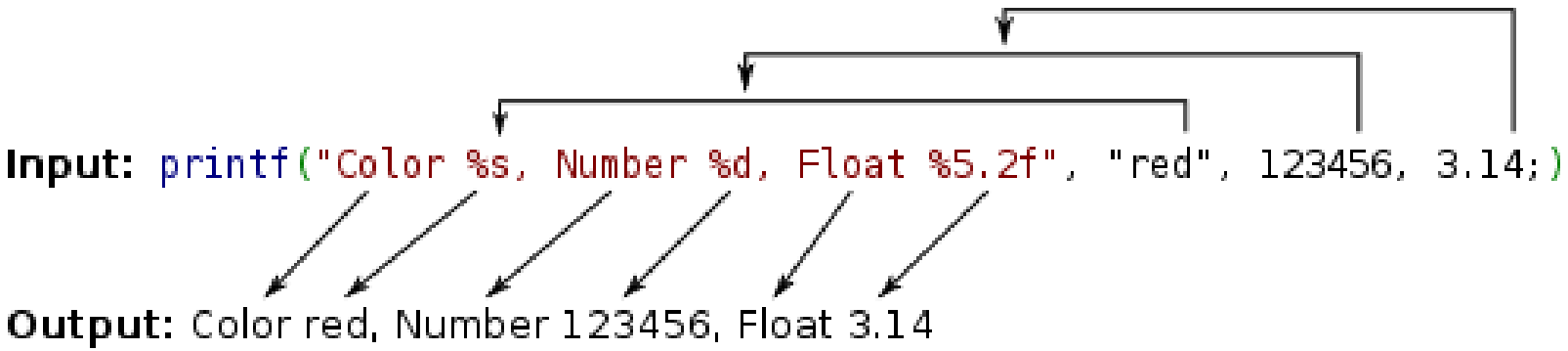
Wypisywanie łańcucha znakowego

```
int puts(char *s);
```

- wypisuje łańcuch i dodaje znak nowej lini

```
int printf(char *s, ...);
```

- wypisuje napis zgodnie z zadanym formatem



Specyfikacja formatu

`%[flagi][szerokość][.precyzja][długość]specyfikator`

http://wikipedia.org

%[flagi][szerokość][.precyzja][długość]specyfikator

specyfikator	znaczenie	przykład	wynik
d lub i	liczba całkowita ze znakiem	`printf("%d", -123)`	-123
u	liczba całkowita bez znaku	`printf("%u", 123)`	123
o	liczba całkowita bez znaku ósemkowa	`printf("%o", 123)`	173
x lub X	liczba całkowita bez znaku szesnastkowo	`printf("%o", 123)`	7b
f lub F	zmiennopozycyjna w zapisie dziesiętnym	`printf("%f", 12.6)`	12.600000
e lub E	notacja naukowa	`printf("%e", 12.6)`	1.260000e+01
g lub G	krótszy zapis %f lub %e	`printf("%g", 12.6)`	12.6
c	pojedynczy znak (ASCII)	`printf("%c", 'a')`	a
s	łańcuch znakowy	`printf("%s", "Ala ma kota")`	Ala ma kota
p	adres (wskaźnik), szesnastkowo	`printf("%p", &a)`	ff01ffab
%	wypisuje znak %	`printf("%%")`	%

%[flagi][szerokość][.precyzja][długość]specyfikator

szerokość	znaczenie	przykład	wynik
liczba	minimalna ilość znaków (szerokość pola), brakujące miejsca są dopełniane spacjami. Jeżeli szerokość pola jest za mała to wynik nie jest obcinany.	`printf("_%5d_", 42)`	`_ 42_`
*	szerokość zależna od argumentu funkcji `printf`	`printf("_%*d_", 5, 42)`	`_ 42_`

flaga	znaczenie	przykład	
-	wyrównanie do lewej (względem podanej szerokości)	`printf("_%-5d_", 42)`	`_42 _`
+	liczby dodatnie poprzedzone znakiem +	`printf("_%-+5d_", 42)`	`_+42 _`
spacja	wstawia spację zamiast znaku +	`printf("_%- 5d_", 42)`	`_ 42 _`
0	wypełnia podaną szerokość znakiem 0	`printf("_%05d_", 42)`	`_00042_`
#	liczby ósemkowe i szesnastkowe poprzedza 0, 0x, 0X	`printf("%#x, 42)`	`0x2a`

%[flagi][szerokość][.precyzja][długość]specyfikator

precyzja	znaczenie	przykład	wynik
.liczba	ilość cyfr wypisywanych po przecinku. Dla napisów oznacza maksymalną liczbę wypisanych znaków (łańcuch jest obcinany)	`printf("%.2f", 1.66666)`	`1.67`
`.*`	precyzja liczby zmiennopozycyjnej nie jest podawana w formacie lecz poprzez argument funkcji `printf`.	`printf("%.*f", 2, 1.66666)`	`1.67`

długość	znaczenie	przykład
brak	typy `char`, `short`, `int`, `double`, `float`	`printf("%f", x)`
l	typ `long int`	`printf("%ld", x)`
ll	typ `long long int`	`printf("%lld", x)`
L	typ `long double`	`printf("%Lf", x)`

```c
#include <stdio.h>

int main()
{
    printf ("Znaki                    : %c %c \n", 'A', 65);
    printf ("Liczby calkowite         : %d \n", 123);
    printf ("  ze znakiem             : %+d \n", 123);
    printf ("  szesnastkowo           : %x %#x \n", 123, 123);
    printf ("  dopelnienie spacjami   : %20d \n", 123);
    printf ("  dopelnienie zerami     : %020d \n", 123);
    printf ("Zmienopozycyjne          : %f \n", 3.1416);
    printf ("  notacja naukowa        : %e \n", 3.1416);
    printf ("  precyzja               : %.3f \n", 3.1416);
    printf ("  dopelnienie            : %20.3f \n", 3.1416);
    printf ("Szerokosc pola  *        : %*d \n", 20, 123);
    printf ("Precyzja  .*             : %20.*f \n", 3, 3.1416);
    printf ("Napis                    : %s \n", "Ala ma kota");
    printf ("  dopelnienie            : %20s \n", "Ala ma kota");
    printf ("  obciecie               : %20.5s \n", "Ala ma kota");

    return 0;
}
```

☞ printf.c

```c
#include <stdio.h>

int main()
{
    printf ("Znaki                    : %c %c \n", 'A', 65);
    printf ("Liczby calkowite         : %d \n", 123);
    printf ("  ze znakiem             : %+d \n", 123);
    printf ("  szesnastkowo           : %x %#x \n", 123, 123);
    printf ("  dopelnienie spacjami   : %20d \n", 123);
    printf ("  dopelnienie zerami     : %020d \n", 123);
    printf ("Zmienopozycyjne          : %f \n", 3.1416);
    printf ("  notacja naukowa        : %e \n", 3.1416);
    printf ("  precyzja               : %.3f \n", 3.1416);
    printf ("  dopelnienie            : %20.3f \n", 3.1416);
    printf ("Szerokosc pola  *        : %*d \n", 20, 123);
    printf ("Precyzja  .
    printf ("Napis
    printf ("  dopelnier
    printf ("  obciecie

    return 0;
}
```

```
Znaki                    : A A
Liczby calkowite         : 123
  ze znakiem             : +123
  szesnastkowo           : 7b 0x7b
  dopelnienie spacjami   :                  123
  dopelnienie zerami     : 00000000000000000123
Zmienopozycyjne          : 3.141600
  notacja naukowa        : 3.141600e+00
  precyzja               : 3.142
...
```

```c
#include <stdio.h>

int main()
{
    printf ("Znaki
    printf ("Liczby calk
    printf ("  ze znakie
    printf ("  szesnastk
    printf ("  dopelnier
    printf ("  dopelnienie zerami  : %020d \n", 123);
    printf ("Zmienopozycyjne       : %f \n", 3.1416);
    printf ("  notacja naukowa     : %e \n", 3.1416);
    printf ("  precyzja            : %.3f \n", 3.1416);
    printf ("  dopelnienie         : %20.3f \n", 3.1416);
    printf ("Szerokosc pola  *     : %*d \n", 20, 123);
    printf ("Precyzja  .*          : %20.*f \n", 3, 3.1416);
    printf ("Napis                 : %s \n", "Ala ma kota");
    printf ("  dopelnienie         : %20s \n", "Ala ma kota");
    printf ("  obciecie            : %20.5s \n", "Ala ma kota");

    return 0;
}
```

```
...
Zmienopozycyjne        : 3.141600
  notacja naukowa      : 3.141600e+00
  precyzja             : 3.142
  dopelnienie          :                3.142
Szerokosc pola  *      :                  123
Precyzja  .*           :                3.142
Napis                  : Ala ma kota
  dopelnienie          :          Ala ma kota
  obciecie             :                Ala m
```

☞ printf.c

Podstawowe operacje na łańcuchach

Plik nagłówkowy `string.h`

- długość łańcucha `s`

```
int strlen(const char *s);
```

- konkatenacja (łączenie łańcuchów), na końcu `dest` doklej `src`

```
char *strcat(char *dest, const char *src);
```

- porównywanie łańcuchów

```
int strcmp(const char *s1, const char *s2);
```

  wynik 0 gdy `s1` i `s2` są takie same, wynik -1 gdy `s1 < s2`, lub 1 gdy `s1 > s2`

- kopiowanie łańcuchów, umieszcza `src` w tablicy `dest`

```
char *strcpy(char *dest, const char *src);
```

- wyszukiwanie wzorca `wzor` w tekscie `napis`

```
char *strstr(const char *napis, const char *wzor);
```

  wynikiem jest adres pierwszego wsytąpienia wzorca lub `NULL` gdy wzorzec nie występuje w napisie

Problem: wyszukiwanie podciągu (*pattern matching*).
W ciągu $\mathcal{T}$ znajdź wystąpienie wzorca $\mathcal{W}$.

Tekst $\mathcal{T}$ = "programowanie"

p	r	o	g	r	a	m	o	w	a	n	i	e	\0

Wzorzec $\mathcal{W}$ = "gra"

g	r	a	\0

Algorithm Naiwne wyszukiwanie wzorca

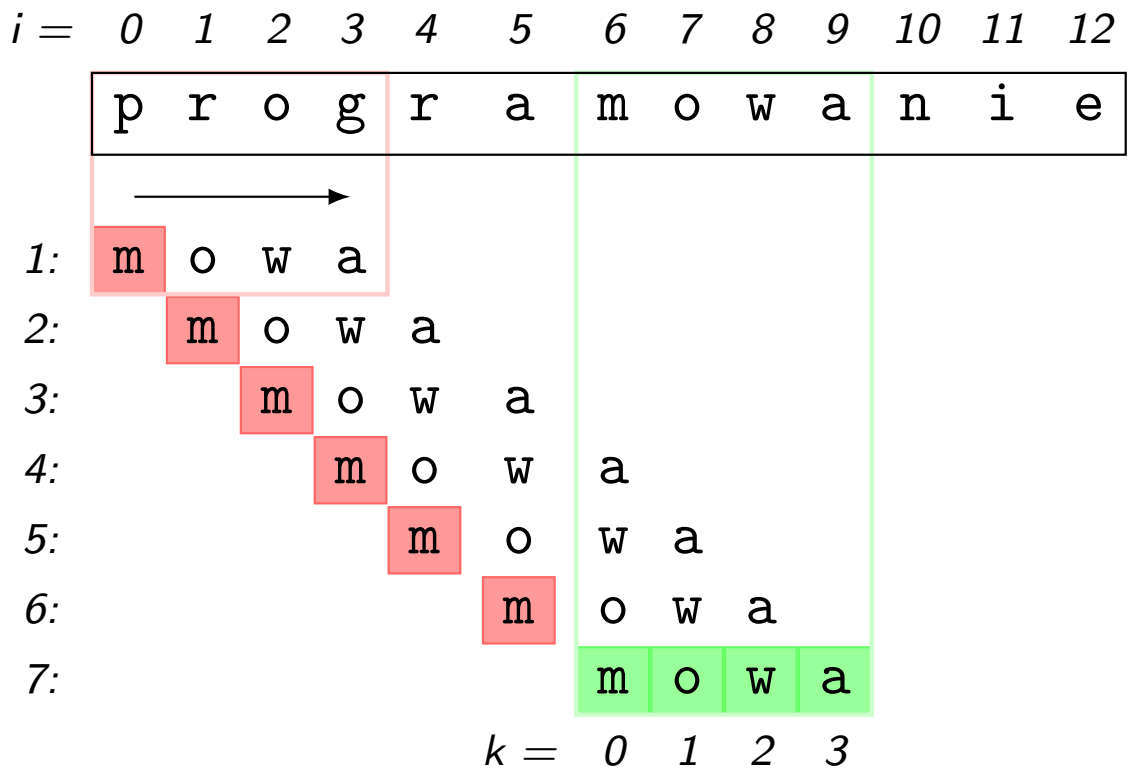
Dane wejściowe: łańcuch znaków $\mathcal{T}$, wzorzec $\mathcal{W}$

Wynik: pozycja tekstu $\mathcal{W}$ w $\mathcal{T}$ lub wartość -1, gdy brak

1: **dla każdego** $i = 0, 1, 2, \ldots, |\mathcal{T}| - |\mathcal{W}|$ **wykonuj**
2: $\quad k \leftarrow 0$
3: $\quad$ **dopóki** $\mathcal{T}[i + k] = \mathcal{W}[k]$ **i** $k < |\mathcal{W}|$ **wykonuj**
4: $\quad\quad k \leftarrow k + 1$
5: $\quad$ **jeżeli** $k = |\mathcal{W}|$ **wykonaj**
6: $\quad\quad$ **zwróć** i
7: **zwróć** -1

$|\mathcal{W}|$ oznacza długość łańcucha $\mathcal{W}$

Tekst $\mathcal{T}$ = "programowanie"
Wzorzec $\mathcal{W}$ = "mowa"



```c
int strindex(const char t[], const char w[])
{
    int i, k;

    i=0;
    while(t[i] != '\0')
    {
        k = 0;
        while (t[i+k] == w[k] && w[k] != '\0')
        {
            k = k + 1;
        }
        if (w[k] == '\0') return i;
        i = i + 1;
    }
    return -1;
}
```

☞ strindex.c

To samo z użyciem wskaźników

```
1  int strindex(const char *t, const char *w)
2  {
3      char *pt, *pw, *pt2;
4
5      for (pt=t; *pt != '\0'; pt++)
6      {
7          pw = w;
8          pt2 = pt;
9          while (*pt2 == *pw && *pw != '\0')
10         {
11             pw++;
12             pt2++;
13         }
14         if (*pw == '\0') return pt - t;
15     }
16     return -1;
17 }
```

☞ strindex3.c

- Typ znakowy jest liczbą całkowitą
- Łańcuch to tablica znaków zakończona znakiem `'\0'`
- Stała znakowa w apostrofach `'A'`
- Stała napisowa w cudzysłowach `"A"` jest tablicą
- Porównywanie łańcuchów: `t == "napis"` źle, trzeba znak po znaku (funkcja `strcmp()`)
- Kopiowanie napisów: `t = "napis"` źle, kopiowanie tablic (funkcja `srtcpy()`)

Literatura dodatkowa

- Linux Programmer's Manual
  `man 7 ascii unicode codepages iso_8859-2`
  `http://www.unix.com/man-page/`
- Wikipedia: ☞ Kodowanie polskich znaków diakrycznych
- Jerzy Wałaszek, *„Algorytmy. Struktury danych.”*,
  ☞ Łańcuchy znakowe.
- R.S. Boyer, J. Strother Moore, *A Fast String Searching Algorithm* Communications of the Association for Computing Machinery, 20(10), 1977, pp. 762-772.
  `www.cs.utexas.edu/~moore/publications/fstrpos.pdf`