Reprezentacja symboli w komputerze.

Znaki alfabetu i łańcuchy znakowe.

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, ...

USASCII code chart

b_7 b_6 b_5 → / Bits b_4 b_3 b_2 b_1 ↓	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Column → / Row ↓	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

http://en.wikipedia.org/wiki/ASCII

Znaki w C

- 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 MS Windows koniec linii to `\r\t`)
  - znaki o specjalnym zapisie:
    ukośnik w lewo `'\\'`,
    apostrof `'\''`,
    cudzysłów `'\"'`,
    znak zapytania `'\?'`
- ☞ man 7 ascii
- Uwaga: `"A"` nie jest pojedynczym znakiem lecz tablicą znaków!

Przykłady w C

```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

- `isalnum` - litera alfabetu lub cyfra `[A-Za-z0-9]`
- `isalpha` - litera alfabetu `[A-Za-z]`
- `islower`, `isupper` - mała / duża litera alfabetu
- `isdigit` - cyfra
- `isgraph` - znaki drukowalne (ze spacją)
- `isspace` - znaki białe
- `isprint` - znaki drukowalne (bez spacji)
- `ispunct` - znaki przestankowe (drukowalne bez liter i cyfr)

Manipulacje na znakach

- `tolower` , `toupper` - zamiana wielkości liter alfabetu

Przykłady w C: zamiana wielkości znaków

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

int main()
{
    int a;

    do
    {
        a=getchar();
        putchar(toupper(a));
    }while( a != EOF );

    return 0;
}
```

☞ toupper2.c

- funkcja `getchar()` zwraca pojedynczy znak ze standardowego wejścia lub `EOF`
- funkcja `putchar()` wypisuje znak
- `EOF` - koniec pliku (*end of file*)
- `toupper2 < plik.txt`
- EOF w terminalu:
  `CTRL+Z` (Windows)
  `CTRL+D` (Unix/Linux)

8 BITOWE STRONY KODOWE

- **Strona kodowa:** kodowanie binarne znaków pisarskich
- Polskie ogonki, 8 bitowe rozszerzenia ASCII:
  ☞ ISO 8859-2 (latin2), alfabet łaciński dla Europy środkowej i wschodniej, UNIX/GNU Linux
  ☞ CP-1250, języki środkowoeuropejskie, MS Windows
  ☞ CP-852, MS DOS
- ☞ Kodowanie polskich znaków diakrycznych

Unikod

- **Unicode** - standard z założenia obejmujący wszystkie języki świata (obecnie ponad 110k znaków ze 100 grup językowych).
- Określa przydział przestrzeni numeracyjnej poszczególnym grupom znaków
- Kodowanie znaków: UCS (Universal Character Set), UTF (Unicode Transformation Format), UTF-8, UTF-16, UTF-32
- UTF-8 - znaki kodowane za pomocą 1, 2, 3 bajtów
- Każdy tekst w ASCII jest tekstem w UTF-8
- Znaki ASCII od 0 do 127 są mapowane jako jeden bajt.
- Znaki spoza ASCII (np. polskie znaki) są mapowane jako dwa bajty (lub więcej).
- Ten sam znak można zapisać na kilka sposobów - wybieramy najkrótszy zapis.

Teoria łańcuchów

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

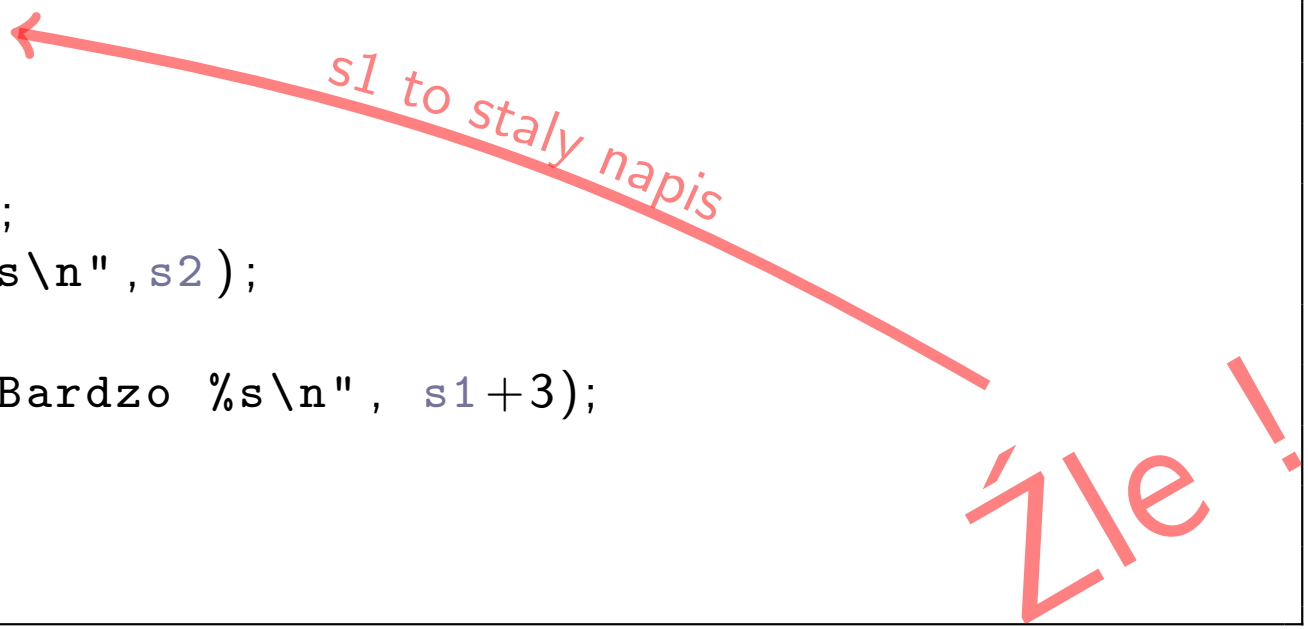
A	l	a		m	a		k	o	t	a	\0

- 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

Przykład w C

```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;
}
```



Wczytywanie łańcucha

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

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

- niebezpieczeństwo przepełnienia tablicy
- `scanf("%10s",text);`
- wczyta tylko pierwszy wyraz, nie wczyta całego łańcucha z odstępami: `"Ala ma kota"`

Wczytywanie łańcucha

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

- funkcja `gets()` czyta linię tekstu ze standardowego wejścia i umieszcza ją w tablicy `s`
- Uwaga: brak możliwości zabezpieczenia się przez przepełnieniem tablicy

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

Wczytywanie łańcucha

```
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`
- standardowy strumień wejściowy to `stdin`

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

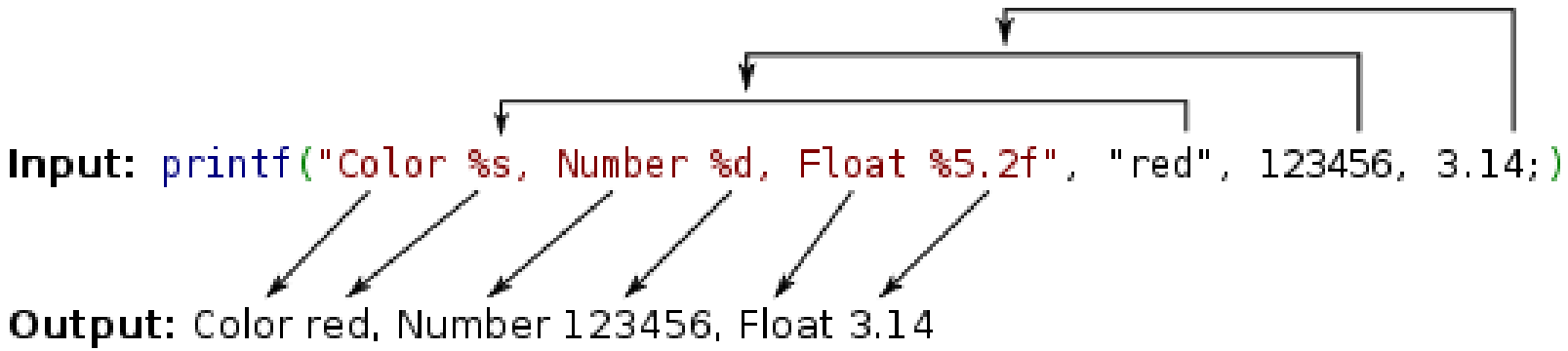
Wypisywanie łańcucha znakowego

```
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
`d` lub `i`	liczba całkowita ze znakiem	`-123`
`u`	liczba całkowita bez znaku	`123`
`o`	liczba całkowita bez znaku ósemkowa	`123`
`x` lub `X`	liczba całkowita bez znaku szesnastkowo	`fa1 FA1`
`f` lub `F`	zmiennopozycyjna w zapisie dziesiętnym	`123.45678`
`e` lub `E`	notacja naukowa	`1.2345678e+2`
`g` lub `G`	krótszy zapis `%f` lub `%e`	`123.4657`
`c`	pojedynczy znak (ASCII)	`a`
`s`	łańcuch znakowy	`Ala ma kota`
`p`	adres (wskaźnik), szesnastkowo	`ff01ffab`
`%`	wypisuje znak %	`%`

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

flaga	znaczenie	przykład
-	wyrównanie do lewej (względem podanej szerokości)	`123 `
+	liczby dodatnie poprzedzone znakiem +	`+123`
spacja	wstawia spację zamiast znaku +	` 123`
0	wypełnia podaną szerokość znakiem 0	`000123`
#	liczby ósemkowe i szesnastkowe poprzedza `0`, `0x`, `0X`	`0123 0xfa1`

szerokość	znaczenie
liczba	określa minimalną ilość znaków wypisanych (szerokość pola), brakujące miejsca są dopełniane spacjami. Jeżeli szerokość pola jest za mała to wynik nie jest obcinany.
*	szerokość nie jest dana w formacie lecz poprzez argument funkcji `printf`

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

precyzja	znaczenie
.*liczba*	ilość cyfr wypisywanych po przecinku
`.*`	precyzja liczby zmiennopozycyjnej nie jest podawana w formacie lecz poprzez argument funkcji `printf`. W przypadku łańcuchów znakowych oznacza maksymalną liczbę wypisanych znaków (łańcuch jest obcinany)

długość	znaczenie
brak	liczby `int`, `double`, `float`
l	liczby `long int`
ll	liczby `long long int`
L	liczby `long double`

```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 ("  dopelnienie
    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 calkow
    printf ("  ze znakiem
    printf ("  szesnastkow
    printf ("  dopelnienie
    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 łancucha
  ```
  int strlen(const char *s);
  ```
- łączenie dwóch łańcuchów
  ```
  char *strcat(char *dest, const char *src);
  ```
- porównywanie łańcuchów
  ```
  int strcmp(const char *s1, const char *s2);
  ```
- kopiowanie łańcuchów
  ```
  char *strcpy(char *dest, const char *src);
  ```
- wyszukiwanie wzorca
  ```
  char *strstr(const char *napis, const char *wzor);
  ```

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

Algorytm 1 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, ..., |T| − |W|  wykonuj
2:     k ← 0
3:     dopóki  T[i + k] = W[k] i k < |W|  wykonuj
4:         k ← k + 1
5:     jeżeli  k = |W|  wykonaj
6:         zwróć i
7: zwróć −1
```

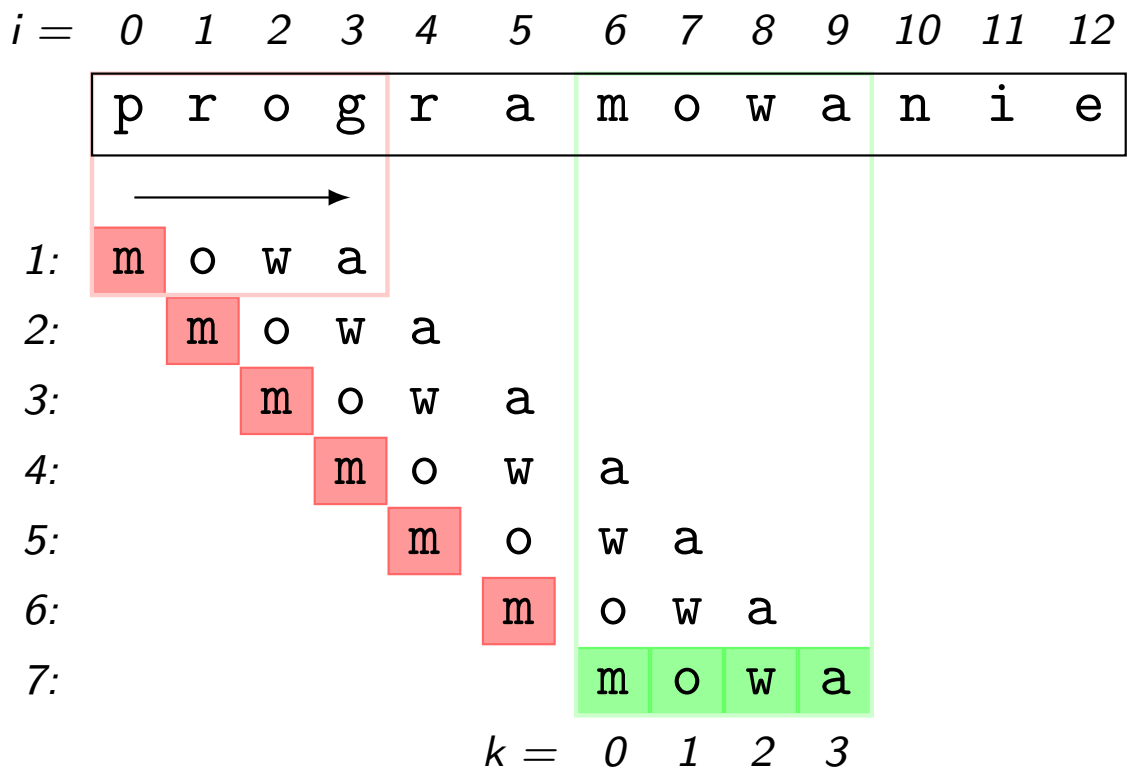
$|\mathcal{W}|$ oznacza długość łańcucha $\mathcal{W}$

Przykłady w C

```
1  int strindex(char t[], char w[])
2  {
3      int i, k;
4
5      i=0;
6      while(t[i] != '\0')
7      {
8          k=0;
9          while(t[i+k]==w[k] && w[k] != '\0')
10             k = k + 1;
11         if(w[k]=='\0') return i;
12         i = i + 1;
13     }
14     return -1;
15 }
```

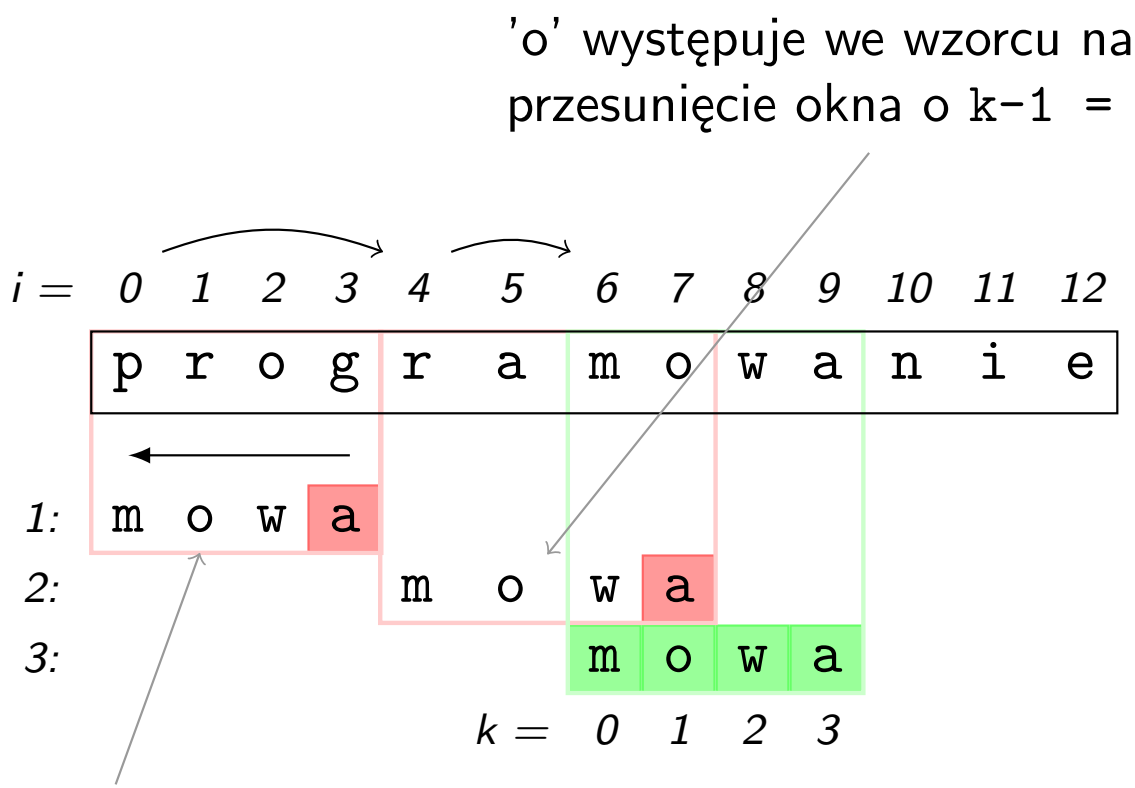
☞ strindex.c

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



Uproszczony algorytm Boyera-Moore'a

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



Tablica przesunięć

- informacja o tym czy znak występuje we wzorcu
- rozmiar tablicy odpowiada liczbie znaków danego alfabetu, dla kodu ASCII wystarczy 128 elementów
- przechowuje pozycję ostatniego wystąpienia znaku we wzorcu lub -1 gdy znak nie występuje
- `tp[w[k]] = k`
- przesunięcie okna o wartość `max( k-tp[w[k]] , 1 )`

Tablica przesunięć

```
1  void utworz_tp(int tp[], char w[])
2  {
3      int i=0;
4
5      while(i<128)
6      {
7          tp[i]=-1;
8          i=i+1;
9      }
10
11     i=0;
12     while(w[i] != '\0')
13     {
14         tp[w[i]]=i;
15         i=i+1;
16     }
17 }
```

Algorytm 2 Uproszczony algorytm Boyera-Moore'a

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

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

1: $\mathcal{P} \leftarrow utworz\_tp(\mathcal{W})$ ▷ tworzenie tablicy przesunięć
2: $i \leftarrow 0$
3: **dopóki** $i \leqslant |\mathcal{T}| - |\mathcal{W}|$ **wykonuj**
4: $k \leftarrow |\mathcal{W}|$ ▷ porównywanie od tyłu
5: **dopóki** $k \geqslant 0$ **i** $\mathcal{T}[i+k] = \mathcal{W}[k]$ **wykonuj**
6: $k \leftarrow k - 1$
7: **jeżeli** $k = -1$ **wykonaj**
8: **zwróć** i
9: **w przeciwnym wypadku**
10: $i \leftarrow i + \max(1, k - \mathcal{P}[kod(\mathcal{T}[i+k])])$
11: **zwróć** -1

gdzie $kod(x)$ oznacza kod liczbowy znaku x.

```c
int strindex2(char t[], char w[])
{
    int i, k, z, tl, wl;
    int tp[128];

    utworz_tp(tp,w);
    tl=strlen(t);
    wl=strlen(w);

    i=0;
    while(i <= tl-wl)
    {
        k=wl-1;
        while(k>=0 && t[i+k]==w[k] ) k=k-1;
        if(k==-1) return i;
        z=k-tp[t[i+k]];
        if(z<1) z=1;
        i = i + z;
    }
    return -1;
}
```

Podsumowanie

- Typ znakowy jest liczbą całkowitą
- Łańcuch to tablica znaków zakończona znakiem `'\0'`
- Stała znakowa w apostrofach `'A'`
- Stała napisowa w cudzysłowach `"A"` jest tablicą
- Dostęp do znaku `t[i]`
- `t == "napis"` tak nie wolno porównywać, trzeba znak po znaku

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`