

Informatyka 1

Politechnika Białostocka - Wydział Elektryczny
Elektrotechnika, semestr II, studia stacjonarne I stopnia
Rok akademicki 2017/2018

Wykład nr 2 (05.03.2018)

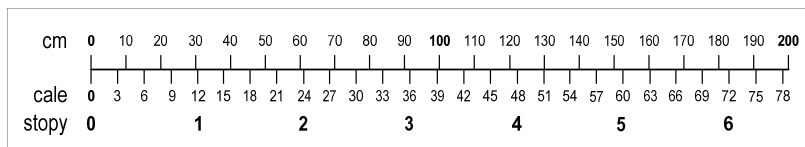
dr inż. Jarosław Forenc

Plan wykładu nr 2

- Język C
 - identyfikatory (nazwy), słowa kluczowe
 - typy danych, stałe liczbowe
 - deklaracje zmiennych i stałych
 - operatory, priorytet operatorów
 - wyrażenia i instrukcje
 - wyrażenia arytmetyczne
 - funkcje printf i scanf
- Konwersje między systemami liczbowymi
- Kodowanie znaków
 - ASCII, ISO/IEC 646, ISO 8859
 - EBCDIC, Windows-1250, Unicode

Przykład: zamiana wzrostu w cm na stopy i cale

- Wybrane jednostki długości w brytyjskim systemie miar:
 - 1 cal (inch) [in] = 2,54 [cm]
 - 1 stopa (foot) [ft] = 12 cali = 30,48 [cm]



- 1 jard (yard) [yd] = 3 stopy = 91,44 [cm]
- 1 furlong [fur] = 660 stóp = 201,168 [m]
- 1 mila (mile) [mi] = 8 furlongów = 1609,344 [m]

Przykład: zamiana wzrostu w cm na stopy i cale

```
#include <stdio.h>

int main(void)
{
    float cm;      /* wzrost w cm */
    float stopy;   /* wzrost w stopach */
    float cale;    /* wzrost w calach */

    printf("Podaj wzrost w cm: ");
    scanf("%f", &cm);

    stopy = cm / 30.48f;
    cale = cm / 2.54f;

    printf("%f [cm] = %f [ft]\n", cm, stopy);
    printf("%f [cm] = %f [in]\n", cm, cale);

    return 0;
}
```

Podaj wzrost w cm: 175
175.000000 [cm] = 5.741470 [ft]
175.000000 [cm] = 68.897636 [in]

Język C - identyfikatory (nazwy)

- Dozwolone znaki: **A-Z, a-z, 0-9, _** (podkreślenie)
- Długość nie jest ograniczona (rozdzielalne są 63 pierwsze znaki)
- Poprawne identyfikatory:

```
temp  u2  u_2  pole_kola  alfa  Beta  XyZ
```

- Pierwszym znakiem nie może być cyfra
- W identyfikatorach nie można stosować spacji, liter diakrytycznych
- Błędne identyfikatory:

```
2u  pole kola  pole_koła
```

Język C - identyfikatory (nazwy)

- Nie zaleca się, aby pierwszym znakiem było podkreślenie
- Identyfikatory nie powinny być zbyt długie

```
_temp  __temp  temperatura_w_skali_Celsjusza
```

- Nazwa **zmiennej** powinna być związana z jej zawartością
- Język C rozróżnia wielkość liter więc poniższe zapisy oznaczają inne identyfikatory

```
tempc  Tempc  TempC  TEMPC  TeMpC
```

- Jako nazw zmiennych nie można stosować **słów kluczowych** języka C

Język C - słowa kluczowe języka C

- W standardzie C11 zdefiniowane są 43 słowa kluczowe

```
auto      extern   short    while  
break     float    signed   _Alignas  
case      for      sizeof   _Alignof  
char      goto     static   _Bool  
const     if       struct   _Complex  
continue  inline   switch   _Generic  
default   int      typedef  _Imaginary  
do        long     union    _Noreturn  
double    register unsigned _Static_assert  
else      restrict void      _Thread_local  
enum      return   volatile
```

Język C - Typy danych

Nazwa	Rozmiar (bajty)	Zakres wartości
char	1	-128 ... 127
int	4	-2147483648 ... 2147483647
float	4	$-3,4 \cdot 10^{38} \dots 3,4 \cdot 10^{38}$
double	8	$-1,7 \cdot 10^{308} \dots 1,7 \cdot 10^{308}$
void	-	-

- Słowa kluczowe wpływające na typy:
 - **signed** - liczba ze znakiem (dla typów **char** i **int**), np. **signed char**
 - **unsigned** - liczba bez znaku (dla typów **char** i **int**), np. **unsigned int**
 - **short, long, long long** - liczba krótka/długa (dla typu **int**), np. **short int**
 - **long** - większa precyzja (dla typu **double**), **long double**

Język C - Typy danych

- Zależnie od środowiska programistycznego (kompilatora) zmienne typów `int` i `long double` mogą zajmować różną liczbę bajtów

Środowisko	int (bajty)	long double (bajty)
Microsoft Visual Studio 2008	4	8
Microsoft Visual Studio 2015	4	8
Dev-C++ 5.11	4	12
Code::Blocks 16.01	4	12
Borland Turbo C++ 2006	4	10
Borland C++ 3.1	2	10

Język C - Typy danych (sizeof)

- `sizeof` - operator zwracający liczbę bajtów zajmowanych przez obiekt lub zmienną podanego typu

```
sizeof(nazwa_typu)
sizeof(nazwa_zmiennej)
sizeof nazwa_zmiennej
```

- Operator `sizeof` zwraca wartość typu `size_t`
- Zależnie od środowiska programistycznego typ `size_t` może odpowiadać typowi `unsigned int` lub `unsigned long int`
- W standardach C99 i C11 wprowadzono specyfikator formatu `%zd` przeznaczony do wyświetlania wartości typu `size_t` (Uwaga: nie działa w Visual Studio 2008)

Język C - Typy danych (sizeof)

```
#include <stdio.h>

int main(void)
{
    int x;

    printf("int: %d\n", sizeof(int));
    printf("int: %d\n", sizeof(x));
    printf("int: %d\n", sizeof x);

    printf("long double: %d\n", sizeof(long double));

    return 0;
}
```

```
int: 4
int: 4
int: 4
long double: 8
```

Język C - Stałe liczbowe (całkowite)

- Liczby całkowite** (ang. integer) domyślnie zapisywane są w systemie dziesiętnym i mają typ `int`

```
1    100    -125    123456
```

- Zapis liczb w innych systemach liczbowych
 - ósemkowy: 0 na początku, np. `011`, `024`
 - szesnastkowy: `0x` na początku, np. `0x2F`, `0xab`
- Przyrostki na końcu liczby zmieniają typ
 - `l` lub `L` - typ `long int`, np. `10l`, `10L`, `011L`, `0x2FL`
 - `ll` lub `LL` - typ `long long int`, np. `10ll`, `10LL`, `011LL`, `0x2FLL`
 - `u` lub `U` - typ `unsigned`, np. `10u`, `10U`, `10IU`, `10LLU`, `0x2FUll`

Język C - Stałe liczbowe (rzeczywiste)

- Domyślny typ liczb rzeczywistych to **double**
- Format zapisu **stałych zmiennoprzecinkowych** (ang. floating-point)

`-2.41e+15` `-2.41e+15` `+4.123E-3` `+4.123E-3`

znak plus/minus	mantysa (ciąg cyfr z kropką dziesiętną)	e lub E	wykładnik ze znakiem
-----------------	-----------------------------------------	---------	----------------------

- W zapisie można pominąć:
 - znak plus, np. `-2.41e15`, `4.123E-3`
 - kropkę dziesiętną lub część wykładniczą, np. `2e-5`, `14.15`
 - część ułamkową lub część całkowitą, np. `2.e-5`, `.12e4`

Język C - Stałe liczbowe (rzeczywiste)

- W środku stałej zmiennoprzecinkowej nie mogą występować spacje
- Błędnie zapisane stałe zmiennoprzecinkowe:

`- 2.41e+15` `-2.41 e+15` `-2.41e +15`

- Przyrostki na końcu liczby zmieniają typ:
 - `l` lub `L` - typ **long double**, np. `2.5L`, `1.24e7l`
 - `f` lub `F` - typ **float**, np. `3.14f`, `1.24e7f`

Język C - Deklaracje zmiennych i stałych

- **Zmienne** (ang. variables) - zmieniają swoje wartości podczas pracy programu
- **Stałe** (ang. constants) - mają wartości ustalone przed uruchomieniem programu i pozostają niezmienione przez cały czas jego działania
- **Deklaracja** nadaje zmiennej / stałej nazwę, określa typ przechowywanej wartości i rezerwuje odpowiednio obszar pamięci

■ Deklaracje zmiennych:

```
int x;  
float a, b;  
char zn1;
```

■ Deklaracje stałych:

```
const int y = 5;  
const float c = 1.25f;  
const char zn2 = 'Q';
```

■ Inicjalizacja zmiennej:

```
int x = -10;
```

Język C - Stałe symboliczne (#define)

- Dyrektywa preprocesora **#define** umożliwia definiowanie tzw. stałych symbolicznych

`#define nazwa_stalej wartość_stalej`

```
#define PI 3.14  
#define KOMUNIKAT "Zaczynamy!!!\n"
```

- Wyrażenia stałe zazwyczaj pisze się wielkimi literami
- Wyrażenia stałe są obliczane przed właściwą kompilacją programu
- W kodzie programu w miejscu występowania stałej wstawiana jest jej wartość

Język C - Stałe symboliczne (#define)

```
#include <stdio.h>
#define PI 3.14
#define KOMUNIKAT "Zaczynamy!!!\n"

int main(void)
{
    double pole, obwod;
    double r = 1.5;

    printf(KOMUNIKAT);
    pole = PI * r * r;
    obwod = 2 * PI * r;

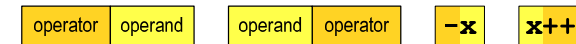
    printf("Pole = %g\n", pole);
    printf("Obwod = %g\n", obwod);

    return 0;
}
```

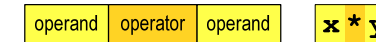
Zaczynamy!!!
Pole = 7.065
Obwod = 9.42

Język C - Operatory

- Operator - symbol lub nazwa operacji
- Argumenty operatora nazywane są operandami
- Operator jednoargumentowy



- Operator dwuargumentowy



- Operator trójargumentowy



- Operator wieloargumentowy



Język C - Operatory

Typ	Symbol
Arytmetyczne	+ - * / %
Inkrementacji / dekrementacji	++ --
Porównania (relacyjne)	< > <= >= == !=
Logiczne	&& !
Bitowe	& ^ << >> ~
Przypisania	= += -= *= /= %= <<= >>= &= = ^=
Inne	() [] & * -> . , ? : sizeof (typ)

Język C - Priorytet operatorów (1/2)

Priorytet	Operator / opis
1	++ -- (przyrostki) () [] . ->
2	++ -- (przedrostki) sizeof (typ) + - ! ~ * & (jednoargumentowe)
3	* / %
4	+ - (dwuargumentowe)
5	<< >>
6	< > <= >=
7	== !=
8	& (bitowy)
9	^

Język C - Priorytet operatorów (2/2)

Priorytet	Operator / opis
10	
11	&&
12	
13	? :
14	= += -= *= /= %= <<= >>= &= = ^=
15	, (przecinek)

Język C - Wyrażenia

- **Wyrażenie** (ang. expression) - kombinacja operatorów i operandów

4 -6 4+2.1 x=5+2 a>3 x>5&& x<8

- Każde wyrażenie ma **typ** i **wartość**

Wyrażenie	Typ	Wartość
4	int	4
-6	int	-6
4 + 2.1	double	6.1
x = 5 + 2	typ x	7
a > 3	int	1 (prawda) / 0 (fałsz)
x > 5 && x < 8	int	1 (prawda) / 0 (fałsz)

Język C - Instrukcje

- **Instrukcja** (ang. statement) - główny element, z którego zbudowany jest program, kończy się średnikiem

Wyrażenie:

x = 5

Instrukcja:

x = 5;

- Język C za instrukcję uznaje każde wyrażenie, na którego końcu znajduje się średnik

```
8;  
x;  
3 + 4;  
a > 5;
```

- Powyższe instrukcje są poprawne, ale nie dają żadnego efektu

Język C - Instrukcje

- Podział instrukcji:
 - **proste** - kończą się średnikiem
 - **złożone** - kilka instrukcji zawartych pomiędzy nawiasami klamrowymi

- Typy instrukcji prostych:

- deklaracji:

```
int x;
```

- przypisania:

```
x = 5;
```

- wywołania funkcji:

```
printf("Witaj swiecie\n");
```

- strukturalna:

```
while(x > 0) x--;
```

- pusta:

```
;
```

Język C - Wyrażenia arytmetyczne

- Wyrażenia arytmetyczne mogą zawierać:
 - stałe liczbowe, zmienne, stałe
 - operatory: `+` `-` `*` `/` `%` `=` `()` i inne
 - wywołania funkcji (plik nagłówkowy `math.h`)
- Kolejność wykonywania operacji wynika z priorytetu operatorów

<code>w = a + b;</code>	<code>+</code> → <code>=</code>
<code>w = a + b * c;</code>	<code>*</code> → <code>+</code> → <code>=</code>
<code>w = (a + b) * c;</code>	<code>(+)</code> → <code>*</code> → <code>=</code>
<code>w = (a + b) * (c + d);</code>	<code>(+) lub (+)</code> → <code>*</code> → <code>=</code>

Język C - Wyrażenia arytmetyczne

- Kolejność wykonywania operacji

<code>w = a + b + c;</code>	→	<code>w = ((a + b) + c);</code>
<code>w = x = y = a + b;</code>	→	<code>w = (x = (y = (a + b)));</code>

- Zapis wyrażeń arytmetycznych

$w = \frac{a+b}{c+d}$	<code>w = a + b / c + d;</code>	ŹLE
	<code>w = (a + b) / (c + d);</code>	DOBRZE
$w = \frac{a+b}{c \cdot d}$	<code>w = (a + b) / c * d;</code>	ŹLE
	<code>w = (a + b) / (c * d);</code>	DOBRZE

Język C - Wyrażenia arytmetyczne

- Podczas dzielenia liczb całkowitych odrzucana jest część ułamkowa

$w = \frac{5}{4}$	<code>5 / 4 = 1</code>
	<code>5.0 / 4 = 1.25</code>
	<code>5 / 4.0 = 1.25</code>
	<code>5.0 / 4.0 = 1.25</code>
	<code>5.0f / 4 = 1.25</code>
	<code>5. / 4 = 1.25</code>
	<code>(float) 5 / 4 = 1.25</code>

Rzutowanie: (typ)

Język C - Funkcje matematyczne (math.h)

- Plik nagłówkowy `math.h` zawiera definicje wybranych stałych

Nazwa	Wartość	Znaczenie
<code>M_PI</code>	3.14159265358979323846	liczba pi
<code>M_E</code>	2.71828182845904523536	e - liczba Eulera
<code>M_LN2</code>	0.693147180559945309417	ln 2
<code>M_SQRT2</code>	1.41421356237309504880	$\sqrt{2}$

- W środowisku Visual Studio 2008 użycie stałych wymaga definicji odpowiedniej stałej (przed `#include <math.h>`)

```
#define _USE_MATH_DEFINES
#include <math.h>
```

Język C - Funkcje matematyczne (math.h)

- Wybrane funkcje matematyczne:

Nazwa	Nagłówek	Znaczenie
<code>abs</code>	<code>int abs(int x);</code>	moduł x (x - całkowite)
<code>fabs</code>	<code>double fabs(double x);</code>	moduł x (x - rzeczywiste)
<code>sqrt</code>	<code>double sqrt(double x);</code>	pierwiastek kwadratowy x
<code>pow</code>	<code>double pow(double x, double y);</code>	x^y - x do potęgi y
<code>sin</code>	<code>double sin(double x);</code>	sinus argumentu x w radianach
<code>atan</code>	<code>double atan(double x);</code>	arcus tangens argumentu x
<code>atan2</code>	<code>double atan2(double y, double x);</code>	arcus tangens ilorazu y/x

- Wszystkie funkcje mają po trzy wersje - dla argumentów typu: `float`, `double` i `long double`

Język C - Funkcja printf

- Ogólna składnia funkcji `printf`

```
printf("łańcuch_sterujący", arg1, arg2, ...);
```

- W najprostszej postaci `printf` wyświetla tylko tekst

```
printf("Witaj świecie");
```

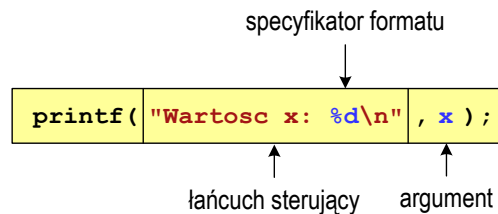
```
Witaj świecie
```

- Do wyświetlenia wartości zmiennych konieczne jest zastosowanie **specyfikatorów formatu**, określających typ oraz sposób wyświetlania argumentów

```
%[znacznik][szerokość][.precyzja][modyfikator]typ
```

Język C - Funkcja printf

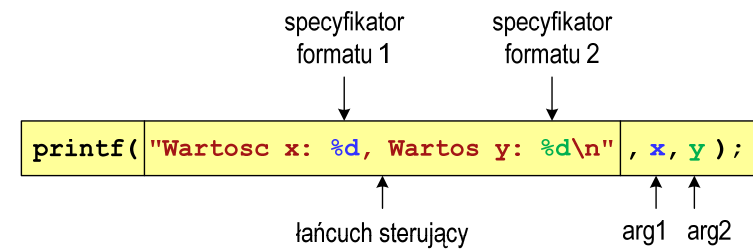
```
int x = 10;  
printf("Wartosc x: %d\n", x);
```



```
Wartosc x: 10
```

Język C - Funkcja printf

```
int x = 10, y = 20;  
printf("Wartosc x: %d, Wartosc y: %d\n", x, y);
```



```
Wartosc x: 10, Wartosc y: 20
```


Język C - Specyfikatory formatu (printf)

Typ w C	Specyfikator	Uwagi
char	%c	pojedynczy znak
	%d	kod ASCII znaku, liczba całkowita
char *	%s	łańcuch znaków, napis
int	%d %i	liczba całkowita, dziesiętna
	%o %O	liczba całkowita, ósemkowa
	%x %X	liczba całkowita, szesnastkowa
float double	%f	liczba rzeczywista
	%e %E	liczba rzeczywista, format naukowy
	%g %G	liczba rzeczywista (%f lub %e)

Język C - Funkcja printf

```
int x = 123; float y = 1.23456789f;
```

```
printf("x = [%d], y = [%f]\n", x, y);  
printf("x = [], y = []\n", x, y);  
printf("x = [%d], y = [%d]\n", x, y);
```

```
x = [123], y = [1.123457]  
x = [], y = []  
x = [123], y = [-536870912]
```

Język C - Funkcja printf

```
int x = 123; float y = 1.23456789f;
```

```
printf("x = [%6d], y = [%12f]\n", x, y);  
printf("x = [%6d], y = [%12.3f]\n", x, y);  
printf("x = [%6d], y = [%.3f]\n", x, y);
```

```
x = [ 123], y = [ 1.123457]  
x = [ 123], y = [ 1.123]  
x = [ 123], y = [1.123]
```

%[znacznik][szerokość][.precyzja][modyfikator]typ

Język C - Funkcja printf

```
int x = 123; float y = 1.23456789f;
```

```
printf("x = [%+6d], y = [%+12f]\n", x, y);  
printf("x = [%-6d], y = [%-12f]\n", x, y);  
printf("x = [%06d], y = [%012f]\n", x, y);
```

```
x = [ +123], y = [ +1.123457]  
x = [123 ], y = [1.123457 ]  
x = [000123], y = [00001.123457]
```

%[znacznik][szerokość][.precyzja][modyfikator]typ

Język C - Funkcja printf

```
int x = 123; float y = 1.23456789f;
```

```
printf("x = [%d], y = [%f]\n", x, y);  
printf("x = [%d], y = [%f]\n", x+321, y*25.5f);  
printf("x = [%d], y = [%f]\n", 123, 2.0f*sqrt(y));
```

```
x = [123], y = [1.123457]  
x = [444], y = [28.648149]  
x = [123], y = [2.119865]
```

Język C - Funkcja scanf

- Ogólna składnia funkcji `scanf`

```
scanf("specyfikatory", adresy_argumentów);
```

- Składnia **specyfikatora formatu**

```
 %[szerokość] [modyfikator] typ
```

- Argumenty są adresami obszarów pamięci, dlatego muszą być poprzedzone znakiem `&`

```
int x;  
scanf("%d", &x);
```

Język C - Funkcja scanf

- Specyfikatory formatu** w większości przypadków są takie same jak w przypadku funkcji `printf`
- Największa różnica dotyczy typów `float` i `double`

Typ w C	Specyfikator	Uwagi
float	%f	liczba rzeczywista
	%e %E	liczba rzeczywista, format naukowy
	%g %G	liczba rzeczywista (%f lub %e)
double	%lf	liczba rzeczywista
	%le %lE	liczba rzeczywista, format naukowy
	%lg %lG	liczba rzeczywista (%f lub %e)

Język C - Funkcja scanf

```
int a, b, c;  
scanf("%d %d %d", &a, &b, &c);
```

- Wczytywane argumenty mogą być oddzielone od siebie dowolną liczbą białych (niedrukowalnych) znaków: `spacja`, `tabulacja`, `enter`

```
15 20 -30
```

```
15 20 -30<enter>
```

```
15 20 -30
```

```
15 20 -30<enter>
```

```
15  
20  
-30
```

```
15<enter>  
20<enter>  
-30<enter>
```

Konwersja na system dziesiętny

- $p = 4, D = \{0, 1, 2, 3\}$

$$21302_{(4)} = ?_{(10)}$$

$$21302_{(4)} = 2 \cdot 4^0 + 0 \cdot 4^1 + 3 \cdot 4^2 + 1 \cdot 4^3 + 2 \cdot 4^4$$

$$21302_{(4)} = 2 \cdot 1 + 0 \cdot 4 + 3 \cdot 16 + 1 \cdot 64 + 2 \cdot 256$$

$$21302_{(4)} = 2 + 0 + 48 + 64 + 512 = 626_{(10)}$$

- $p = 17, D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, G\}$

$$AC24_{(17)} = ?_{(10)}$$

$$AC24_{(17)} = 4 \cdot 17^0 + 2 \cdot 17^1 + 12 \cdot 17^2 + 10 \cdot 17^3$$

$$AC24_{(17)} = 4 \cdot 1 + 2 \cdot 17 + 12 \cdot 289 + 10 \cdot 4913$$

$$AC24_{(17)} = 4 + 34 + 3468 + 49130 = 52636_{(10)}$$

Konwersja na system dziesiętny (schemat Hornera)

- $p = 4, D = \{0, 1, 2, 3\}$

$$21302_{(4)} = w_{(10)} \quad x_4 \ x_3 \ x_2 \ x_1 \ x_0 = w_{(10)}$$

$$w_{(10)} = 0$$

$$w_{(10)} = x_4 + w_{(10)} \cdot p = 2 + 0 \cdot 4 = 2$$

$$w_{(10)} = x_3 + w_{(10)} \cdot p = 1 + 2 \cdot 4 = 9$$

$$w_{(10)} = x_2 + w_{(10)} \cdot p = 3 + 9 \cdot 4 = 39$$

$$w_{(10)} = x_1 + w_{(10)} \cdot p = 0 + 39 \cdot 4 = 156$$

$$w_{(10)} = x_0 + w_{(10)} \cdot p = 2 + 156 \cdot 4 = 626_{(10)}$$

Konwersja z systemu dziesiętnego na dowolny

- zamiana liczby z systemu $p = 10$ na system $p = 2$

$$626_{(10)} = ?_{(2)} \quad 626_{(10)} = 1001110010_{(2)}$$

$626/2 = 313$	$reszta$	0	↑ kolejność odczytywania cyfr liczby w systemie dwójkowym
$313/2 = 156$	$reszta$	1	
$156/2 = 78$	$reszta$	0	
$78/2 = 39$	$reszta$	0	
$39/2 = 19$	$reszta$	1	
$19/2 = 9$	$reszta$	1	
$9/2 = 4$	$reszta$	1	
$4/2 = 2$	$reszta$	0	
$2/2 = 1$	$reszta$	0	
$1/2 = 0$	$reszta$	1	

Konwersja z systemu dziesiętnego na dowolny

- zamiana liczby z systemu $p = 10$ na system $p = 7$

$$626_{(10)} = ?_{(7)} \quad 626_{(10)} = 1553_{(7)}$$

$626/7 = 89$	$reszta$	3	↑
$89/7 = 12$	$reszta$	5	
$12/7 = 1$	$reszta$	5	
$1/7 = 0$	$reszta$	1	

- zamiana liczby z systemu $p = 10$ na system $p = 14$

$$626_{(10)} = ?_{(14)} \quad 626_{(10)} = 32A_{(14)}$$

$626/14 = 44$	$reszta$	$10 \rightarrow A$	↑
$44/14 = 3$	$reszta$	2	
$3/14 = 0$	$reszta$	3	

Szybkie konwersje: $2 \rightarrow 4, 8, 16$ $4, 8, 16 \rightarrow 2$

$2 \rightarrow 4$

$$110110011_{(2)} = ?_{(4)}$$

$$\begin{array}{c|c|c|c|c} 01 & 10 & 11 & 00 & 11 \\ \hline 1 & 2 & 3 & 0 & 3 \end{array}$$

$$110110011_{(2)} = 12303_{(4)}$$

$2 \rightarrow 8$

$$10110011_{(2)} = ?_{(8)}$$

$$\begin{array}{c|c|c} 010 & 110 & 011 \\ \hline 2 & 6 & 3 \end{array}$$

$$10110011_{(2)} = 263_{(8)}$$

$2 \rightarrow 16$

$$1011010_{(2)} = ?_{(16)}$$

$$\begin{array}{c|c} 0101 & 1010 \\ \hline 5 & A \end{array}$$

$$1011010_{(2)} = 5A_{(16)}$$

$4 \rightarrow 2$

$$12303_{(4)} = ?_{(2)}$$

$$\begin{array}{c|c|c|c|c} 01 & 10 & 11 & 00 & 11 \\ \hline 1 & 2 & 3 & 0 & 3 \end{array}$$

$$12303_{(4)} = 110110011_{(2)}$$

$8 \rightarrow 2$

$$263_{(8)} = ?_{(2)}$$

$$\begin{array}{c|c|c} 010 & 110 & 011 \\ \hline 2 & 6 & 3 \end{array}$$

$$263_{(8)} = 10110011_{(2)}$$

$16 \rightarrow 2$

$$5A_{(16)} = ?_{(2)}$$

$$\begin{array}{c|c} 0101 & 1010 \\ \hline 5 & A \end{array}$$

$$5A_{(16)} = 1011010_{(2)}$$

Kod ASCII

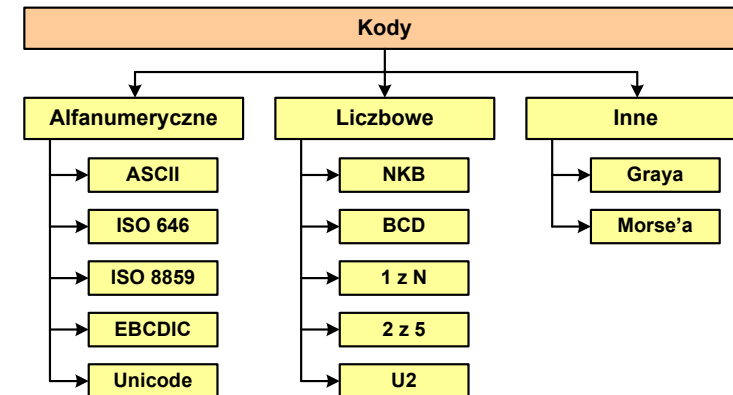
■ ASCII - American Standard Code for Information Interchange

- 7-bitowy kod przypisujący liczby z zakresu 0-127:
 - literom (alfabet angielski)
 - cyfrom
 - znakom przestankowym
 - innym symbolom
 - poleceniom sterującym.

Dec Hex Char	Dec Hex Char	Dec Hex Char	Dec Hex Char
0 0 NUL	32 20 Space	64 40 @	96 60 `
1 1 SOH	33 21 !	65 41 A	97 61 a
2 2 STX	34 22 "	66 42 B	98 62 b
3 3 ETX	35 23 #	67 43 C	99 63 c
4 4 EOT	36 24 \$	68 44 D	100 64 d
5 5 ENQ	37 25 %	69 45 E	101 65 e
6 6 ACK	38 26 &	70 46 F	102 66 f
7 7 BEL	39 27 ^	71 47 G	103 67 g
8 8 BS	40 28 (72 48 H	104 68 h
9 9 TAB	41 29)	73 49 I	105 69 i
10 A LF	42 2A *	74 4A J	106 6A j
11 B VT	43 2B +	75 4B K	107 6B k
12 C FF	44 2C ,	76 4C L	108 6C l
13 D CR	45 2D -	77 4D M	109 6D m
14 E SO	46 2E .	78 4E N	110 6E n
15 F SI	47 2F /	79 4F O	111 6F o
16 10 DLE	48 30 0	80 50 P	112 70 p
17 11 DC1	49 31 1	81 51 Q	113 71 q
18 12 DC2	50 32 2	82 52 R	114 72 r
19 13 DC3	51 33 3	83 53 S	115 73 s
20 14 DC4	52 34 4	84 54 T	116 74 t
21 15 NAK	53 35 5	85 55 U	117 75 u
22 16 SYN	54 36 6	86 56 V	118 76 v
23 17 ETB	55 37 7	87 57 W	119 77 w
24 18 CAN	56 38 8	88 58 X	120 78 x
25 19 EM	57 39 9	89 59 Y	121 79 y
26 1A SUB	58 3A :	90 5A Z	122 7A z
27 1B ESC	59 3B ;	91 5B [123 7B {
28 1C FS	60 3C <	92 5C \	124 7C
29 1D GS	61 3D =	93 5D]	125 7D }
30 1E RS	62 3E >	94 5E ^	126 7E ~
31 1F US	63 3F ?	95 5F _	127 7F DEL

Kodowanie

- **Kodowanie** - proces przekształcania jednego rodzaju postaci informacji na inną postać



Kod ASCII - Kody sterujące

- Kody sterujące - 33 kody, o numerach: 0-31, 127

Dec Hex Char	Dec Hex Char
0 0 NUL (null)	16 10 DLE (data link escape)
1 1 SOH (start of heading)	17 11 DC1 (device control 1)
2 2 STX (start of text)	18 12 DC2 (device control 2)
3 3 ETX (end of text)	19 13 DC3 (device control 3)
4 4 EOT (end of transmission)	20 14 DC4 (device control 4)
5 5 ENQ (enquiry)	21 15 NAK (negative acknowledge)
6 6 ACK (acknowledge)	22 16 SYN (synchronous idle)
7 7 BEL (bell)	23 17 ETB (end of trans. block)
8 8 BS (backspace)	24 18 CAN (cancel)
9 9 TAB (horizontal tab)	25 19 EM (end of medium)
10 A LF (NL line feed, new line)	26 1A SUB (substitute)
11 B VT (vertical tab)	27 1B ESC (escape)
12 C FF (NP form feed, new page)	28 1C FS (file separator)
13 D CR (carriage return)	29 1D GS (group separator)
14 E SO (shift out)	30 1E RS (record separator)
15 F SI (shift in)	31 1F US (unit separator)
	127 7F DEL

- W języku C:

0 (NULL) - \0 7 (BEL) - \a 8 (BS) - \b
9 (TAB) - \t 10 (LF) - \n 13 (CR) - \r

Kod ASCII - Pliki tekstowe

- Elementami pliku tekstowego są **wiersze**, mogą one mieć różną długość
- W systemie Windows każdy wiersz pliku zakończony jest parą znaków:
 - CR**, ang. carriage return - powrót karetki, kod ASCII - $13_{(10)} = 0D_{(16)}$
 - LF**, ang. line feed - przesunięcie o wiersz, kod ASCII - $10_{(10)} = 0A_{(16)}$
- Załóżmy, że plik tekstowy ma postać:

Pierwszy wiersz pliku
Drugi wiersz pliku
Trzeci wiersz pliku

- Rzeczywista zawartość pliku jest następująca:

```
00000000: 50 69 65 72 77 73 7A 79|20 77 69 65 72 73 7A 20 | Pierwszy wiersz
00000010: 70 6C 69 68 75 0D 0A 44|72 75 67 69 20 77 69 65 | plikuDrugi wie
00000020: 72 73 7A 20 70 6C 69 68|75 0D 0A 54 72 7A 65 63 | rsz plikuTrzec
00000030: 69 20 77 69 65 72 73 7A|20 70 6C 69 68 75 0D 0A | i wiersz pliku
```

- Wydruk zawiera:
 - przesunięcie od początku pliku (szesnastkowo)
 - wartości poszczególnych bajtów pliku (szesnastkowo)
 - znaki odpowiadające bajtom pliku (traktując bajty jako kody ASCII)

Kod ASCII - Pliki tekstowe

- W systemie Linux znakiem końca wiersza jest tylko **LF** o kodzie ASCII - $10_{(10)} = 0A_{(16)}$
- Załóżmy, że plik tekstowy ma postać:

Pierwszy wiersz pliku
Drugi wiersz pliku
Trzeci wiersz pliku

- Rzeczywista zawartość pliku jest następująca:

```
00000000: 50 69 65 72 77 73 7A 79|20 77 69 65 72 73 7A 20 | Pierwszy wiersz
00000010: 70 6C 69 68 75 0A 44 72|75 67 69 20 77 69 65 72 | plikuDrugi wiersz
00000020: 73 7A 20 70 6C 69 68 75|0A 54 72 7A 65 63 69 20 | sz plikuTrzeci
00000030: 77 69 65 72 73 7A 20 70|6C 69 68 75 0A | wiersz pliku
```

- Podczas przesyłania pliku tekstowego (np. przez protokół **ftp**) z systemu Linux do systemu Windows pojedynczy znak **LF** zamieniany jest automatycznie na parę znaków **CR** i **LF**
- Błędne przesłanie pliku tekstowego (w trybie binarnym) powoduje nieprawidłowe jego wyświetlanie:

Pierwszy wiersz plikuDrugi wiersz plikuTrzeci wiersz pliku

ISO/IEC 646

- ISO/IEC 646** - norma definiująca modyfikację 7-bitowego kodowania ASCII, stosowana w latach 70-tych i 80-tych
- W normie określono 10 pozycji na znaki w języku kraju, który przyjął tę normę oraz 2 pozycje na znaki walut

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	Znaki kontrolne															
10	Znaki kontrolne															
20	SP	!	"	#	\$	%	&	\	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	<	=	>	?	
40	e	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
60	^	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	

- żółty - znaki narodowe
- niebieski - znaki walut

- Wszystkie pozostałe znaki są zgodne z ASCII

ISO/IEC 646 - odmiany narodowe

USA

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	Znaki kontrolne															
10	Znaki kontrolne															
20	SP	!	"	#	\$	%	&	\	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	<	=	>	?	
40	e	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
60	^	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	

Niemcy

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	Znaki kontrolne															
10	Znaki kontrolne															
20	SP	!	"	#	\$	%	&	\	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	<	=	>	?	
40	e	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	ä	ö	ü	^	_
60	^	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	ä	ö	ü	ß	

Francja

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	Znaki kontrolne															
10	Znaki kontrolne															
20	SP	!	"	#	\$	%	&	\	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	<	=	>	?	
40	a	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	ç	ç	ç	ç	ç
60	^	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	é	ù	é	"	

Polska

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	Znaki kontrolne															
10	Znaki kontrolne															
20	SP	!	"	#	\$	%	&	\	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	<	=	>	?	
40	e	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	ł	ł	ł	ł	ł
60	^	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	ó	ł	ł	ł	

ISO/IEC 8859

- ISO/IEC 8859 - zestaw standardów służących do kodowania znaków za pomocą 8-bitów
- Wszystkie zestawy ISO 8859 mają znaki $0_{(10)}-127_{(10)}$ ($00_{(16)}-7F_{(16)}$) takie same jak w kodzie ASCII
- Pozycjom $128_{(10)}-159_{(10)}$ ($80_{(16)}-9F_{(16)}$) przypisane są dodatkowe kody sterujące, tzw. C1 (obecnie nie są używane)
- Od czerwca 2004 roku ISO 8859 nie jest rozwijane.

ISO/IEC 8859

- Stosowane standardy ISO 8859:
 - ISO 8859-1 (Latin-1) - alfabet łaciński dla Europy zachodniej
 - ISO 8859-2 (Latin-2) - łaciński dla Europy środkowej i wschodniej
 - ISO 8859-3 (Latin-3) - łaciński dla Europy południowej
 - ISO 8859-4 (Latin-4) - łaciński dla Europy północnej
 - ISO 8859-5 (Cyrillic) - dla cyrylicy
 - ISO 8859-6 (Arabic) - dla alfabetu arabskiego
 - ISO 8859-7 (Greek) - dla alfabetu greckiego
 - ISO 8859-8 (Hebrew) - dla alfabetu hebrajskiego
 - ISO 8859-9 (Latin-5)
 - ISO 8859-10 (Latin-6)
 - ISO 8859-11 (Thai) - dla alfabetu tajskiego
 - ISO 8859-12 - brak
 - ISO 8859-13 (Latin-7)
 - ISO 8859-14 (Latin-8) - zawiera polskie litery
 - ISO 8859-15 (Latin-9)
 - ISO 8859-16 (Latin-10) - łaciński dla Europy środkowej, zawiera polskie litery

ISO/IEC 8859-1

- ISO/IEC 8859-1, Latin-1 („zachodnioeuropejskie“)
- kodowanie używane w Amerykach, Europie Zachodniej, Oceanii i większej części Afryki
- dostępne języki: albański, angielski, baskijski, duński, estoński, fiński, francuski, hiszpański, irlandzki, islandzki, kataloński, łaciński, niderlandzki, niemiecki, norweski, portugalski, retoromański, szkocki, szwedzki, włoski
- 191 znaków łacińskiego pisma.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	Znaki kontrolne															
10																
20	SP	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
80	Nie używane															
90																
A0	NB	ı	ı̇	ı̈	ı̉	ı̊	ı̋	ı̌	ı̍	ı̎	ı̏	ı̐	ı̑	ı̒	ı̓	ı̔
B0	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ü	Ý	Û	Ü	Ý	Þ	ß	
E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F0	ð	ñ	ò	ó	ô	õ	ö	÷	ü	ý	û	ü	ý	þ	ÿ	

SP - spacja
NBSP - twarda spacja
SHY - miękki dywiz (myślnik)

ISO/IEC 8859-2

- ISO/IEC 8859-2, Latin-2 („środkowo“, „wschodnioeuropejskie“)
- dostępne języki: bośniacki, chorwacki, czeski, węgierski, polski, rumuński, serbski, serbsko-chorwacki, słowacki, słoweński, górno- i dolnołużycki
- możliwość przedstawienia znaków w języku niemieckim i angielskim
- 191 znaków łacińskiego pisma
- kodowanie zgodne z **Polską Normą**.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	Znaki kontrolne															
10																
20	SP	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
80	Nie używane															
90																
A0	NB	À	Á	Â	Ã	Ä	Å	Š	Ṧ	Š̈	Š̉	Š̊	Š̋	Š̌	Š̍	Š̎
B0	°	à	á	â	ã	ä	å	š	ṧ	š̈	š̉	š̊	š̋	š̌	š̍	š̎
C0	Ř	Š	Ṧ	Š̈	Š̉	Š̊	Š̋	Š̌	Š̍	Š̎	Š̏	Š̐	Š̑	Š̒	Š̓	Š̔
D0	Ď	Ň	Ń	Ō	Ó	Ô	Õ	×	Ř	Ů	Ú	Ů	Ú	Ý	Ť	ß
E0	ř	š	ṧ	š̈	š̉	š̊	š̋	š̌	š̍	š̎	š̏	š̐	š̑	š̒	š̓	š̔
F0	ď	ň	ň̇	ň̈	ň̉	ň̊	ň̋	ň̌	ň̍	ň̎	ň̏	ň̐	ň̑	ň̒	ň̓	ň̔

SP - spacja
NBSP - twarda spacja
SHY - miękki dywiz (myślnik)

Windows-1250

- **Windows-1250 (CP-1250)** - strona kodowa używana przez system Microsoft Windows do reprezentacji tekstów w językach środkowoeuropejskich używających alfabetu łacińskiego
- Obsługiwane języki: albański, chorwacki, czeski, polski, rumuński, słowacki, słoweński, węgierski (ale także niemiecki)
- Windows-1250 jest podobny do ISO 8859-2 - posiada wszystkie jego drukowalne znaki (a także kilka dodatkowych), lecz kilka z nich zajmuje inne miejsca.

ISO 8859-2 i Windows-1250 - porównanie

		ISO 8859-2																Windows-1250																	
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
00		Znaki kontrolne																																	
10		Znaki kontrolne																																	
20	SP	!	"	#	\$	%	&	'	()	*	+	,	-	.	/	!	"	#	\$	%	&	'	()	*	+	,	-	.	/				
30		0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?		
40		@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O		
50		P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_		
60		`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o		
70		p	q	r	s	t	u	v	w	x	y	z	{		}	~		p	q	r	s	t	u	v	w	x	y	z	{		}	~			
80		Nie używane																																	
90		Nie używane																																	
A0	NB SP	À	Á	Â	Ã	Ä	Å	Ā	Ą	Ć	Č	Ď	Ě	Š	Ť	Ž	Ž	Ž	À	Á	Â	Ã	Ä	Å	Ā	Ą	Ć	Č	Ď	Ě	Š	Ť	Ž	Ž	Ž
B0		à	á	â	ã	ä	å	ā	ą	ć	č	ď	ě	š	ť	ž	ž	ž	à	á	â	ã	ä	å	ā	ą	ć	č	ď	ě	š	ť	ž	ž	ž
C0		Ř	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ř	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā
D0		Đ	Ń	Ń	Ó	Ó	Ó	Ó	×	Ř	Ů	Ů	Ů	Ů	Ů	Ů	Ů	Ů	Đ	Ń	Ń	Ó	Ó	Ó	×	Ř	Ů	Ů	Ů	Ů	Ů	Ů	Ů	Ů	Ů
E0		č	á	á	á	á	á	á	í	č	č	č	č	č	č	č	č	č	č	á	á	á	á	á	á	á	á	í	č	č	č	č	č	č	č
F0		đ	ň	ň	ó	ó	ó	ó	÷	ř	ů	ů	ů	ů	ů	ů	ů	ů	đ	ň	ň	ó	ó	ó	÷	ř	ů	ů	ů	ů	ů	ů	ů	ů	ů

Problem kodowania polskich liter diakrytycznych

- Problem z wyświetlaniem polskich liter diakrytycznych
 - Tekst zapisany w standardzie ISO-8859-2:

Ā Ć Ę Ź Ń Ó Ś Ź Ź
 ā ć ę ź ń ó ś ź ź

- Tekst wyświetlony w Notatniku systemu Windows (Windows-1250):

Ā Ć Ę Ź Ń Ó Ā Ā Ā
 ā ć ę ź ń ó Ź Ź Ź

Unicode (Unikod)



- Komputerowy zestaw znaków mający obejmować wszystkie pisma i inne znaki (symbole techniczne, wymowy) używane na świecie
- Unicode przypisuje unikalny numer każdemu znakowi, niezależny od używanej platformy, programu czy języka
- Rozwijany przez konsorcjum utworzone przez firmy komputerowe, producentów oprogramowania oraz grupy użytkowników
 - <http://www.unicode.org>
- Pierwsza wersja: **Unicode 1.0** (10.1991)
- Ostatnia wersja: **Unicode 10.0** (20.06.2017)
 - The Unicode Consortium. The Unicode Standard, Version 10.0.0, (Mountain View, CA: The Unicode Consortium, 2017)
 - <http://www.unicode.org/versions/Unicode10.0.0/>
 - koduje 136.690 znaków



Unicode - Zakresy

Zakres:	Znaczenie:
U+0000 - U+007F	Basic Latin (to samo co w ASCII)
U+0080 - U+00FF	Latin-1 Supplement (to samo co w ISO/IEC 8859-1)
U+0100 - U+017F	Latin Extended-A
U+0180 - U+024F	Latin Extended-B
U+0250 - U+02AF	IPA Extensions
U+02B0 - U+02FF	Spacing Modifiers Letters
...	
U+0370 - U+03FF	Greek
U+0400 - U+04FF	Cyrillic
...	
U+1D00 - U+1D7F	Phonetic Extensions
U+1D80 - U+1DBF	Phonetic Extensions Supplement
U+1E00 - U+1EFF	Latin Extended Additional
U+1F00 - U+1FFF	Greek Extended
...	



Unicode

- Standard Unicode definiuje nie tylko kody numeryczne przypisane poszczególnym znakom, ale także określa sposób bajtowego **kodowania** znaków
- Kodowanie określa sposób w jaki znaki ze zbioru mają być zapisane w **postaci binarnej**
- Istnieją trzy podstawowe metody kodowania:
 - 32-bitowe: UTF-32
 - 16-bitowe: UTF-16
 - 8-bitowe: UTF-8gdzie: **UTF** - UCS Transformation Format
UCS - Universal Character Set
- Wszystkie metody obejmują wszystkie kodowane znaki w Unicode.



Unicode

- Metody kodowania różnią się liczbą bajtów przeznaczonych do opisanego kodu znaku

A 00000041	Ω 000003A9	語 00008A9E	𐄀 00010384	UTF-32
A 0041	Ω 03A9	語 8A9E	𐄀 D800 DF84	UTF-16
A 41	Ω CE A9	語 E8 AA 9E	𐄀 F0 90 8E 84	UTF-8

źródło: The Unicode Consortium. The Unicode Standard, Version 8.0



Unicode - kodowanie UTF-32

- **UTF-32** - sposób kodowania standardu Unicode wymagający użycia 32-bitowych słów

A 00000041	Ω 000003A9	語 00008A9E	𐄀 00010384	UTF-32
---------------	---------------	---------------	---------------	--------

- Kod znaku ma zawsze stałą długość 4 bajtów i przedstawia numer znaku w tabeli Unikodu
- Kody obejmują zakres od 0 do 0x10FFFF (od 0 do 1 114 111)
- Kodowanie to jest jednak bardzo nieefektywne - zakodowane ciągi znaków są 2-4 razy dłuższe niż ciągi tych samych znaków zapisanych w innych kodowaniach.

Unicode - kodowanie UTF-16



- **UTF-16** - sposób kodowania standardu Unicode wymagający użycia 16-bitowych słów



- Dla znaków z przedziału od **U+0000** do **U+FFFF** używane jest jedno słowo, którego wartość jest jednocześnie kodem znaku w Unicode
- Dla znaków z wyższych pozycji używa się dwóch słów:
 - pierwsze słowo należy do przedziału: **U+D800 - U+DBFF**
 - drugie słowo należy do przedziału: **U+DC00 - U+DFFF**.

Unicode - kodowanie UTF-8



- **UTF-8** - kodowanie ze zmienną długością reprezentacji znaku wymagające użycia 8-bitowych słów



- Znaki Unikodu są mapowane na ciągi bajtów
 - 0x00 do 0x7F - bity 0xxxxxxx
 - 0x80 do 0x7FF - bity 110xxxxx 10xxxxxx
 - 0x800 do 0xFFFF - bity 1110xxxx 10xxxxxx 10xxxxxx
 - 0x10000 do 0x1FFFFF - bity 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
 - 0x200000 do 0x3FFFFFFF - bity 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
 - 0x4000000 do 0x7FFFFFFF - bity 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

Unicode



	010	011	012	013	014	015	016	017
0	Ā	Đ	Ġ	İ	ı	Ō	Š	Ū
1	ā	đ	ġ	ı	ı	ō	š	ū
2	Ă	Ē	Ģ	Ī	Ĳ	Ţ	Ū	
3	ă	ē	ģ	ij	ĳ	ţ	ų	
4	Ą	Ĕ	Ĥ	Ĵ	ń	Ŕ	Ŧ	Ű
5	ą	ė	ĥ	ĵ	ŋ	ŕ	ŧ	ű
6	Ć	Ė	Ħ	Ķ	ņ	Ŗ	Ŧ	Ŷ
7	ć	ė	ħ	ķ	ņ	ŗ	ŧ	ŷ

European Latin

0100	Ā	LATIN CAPITAL LETTER A WITH MACRON	≡ 0041 A 0304 ☐
0101	ā	LATIN SMALL LETTER A WITH MACRON	• Latvian, Latin, ... ≡ 0061 a 0304 ☐
0102	Ă	LATIN CAPITAL LETTER A WITH BREVE	≡ 0041 A 0306 ☐
0103	ă	LATIN SMALL LETTER A WITH BREVE	• Romanian, Vietnamese, Latin, ... ≡ 0061 a 0306 ☐
0104	Ą	LATIN CAPITAL LETTER A WITH OGONEK	≡ 0041 A 0328 ☐
0105	ą	LATIN SMALL LETTER A WITH OGONEK	• Polish, Lithuanian, ... ≡ 0061 a 0328 ☐
0106	Ć	LATIN CAPITAL LETTER C WITH ACUTE	≡ 0043 C 0301 ☐
0107	ć	LATIN SMALL LETTER C WITH ACUTE	• Polish, Croatian, ... → 045B ħ cyrillic small letter tshe ≡ 0063 c 0301 ☐

Unicode



27308 CJK Unified Ideographs Extension B 27342

27308 虫 142.8	𧈧	𧈨	𧈩	2731B 虫 142.8	𧈪	𧈫	𧈬	2732F 虫 142.8	𧈭	𧈮
UCS2003	GKX-1086.03	T4-4721	UCS2003	GKX-1088.15	T6-617B	UCS2003	GHC	UCS2003	GHC	GHC
27309 虫 142.8	𧈯	𧈰	𧈱	2731C 虫 142.8	𧈲	𧈳	𧈴	27330 虫 142.9	𧈵	𧈶
UCS2003	GKX-1086.05	T5-4955	UCS2003	GKX-1088.16	T6-6221	UCS2003	GHC	UCS2003	GHC	GHC
2730A 虫 142.8	𧈷	𧈸	𧈹	2731D 虫 142.8	𧈺	𧈻	𧈼	27331 虫 142.8	𧈽	𧈾
UCS2003	GKX-1086.08	T4-467D	UCS2003	GKX-1088.17	T5-4960	UCS2003	G4K	UCS2003	G4K	G4K
2730B 虫 142.8	𧈿	𧻀	𧻁	2731E 虫 142.7	𧻂	𧻃		27332 虫 142.8	𧻄	𧻅
UCS2003	GKX-1086.10	T6-6223	UCS2003	GKX-1088.18		UCS2003	GHC	UCS2003	GHC	GHC
2730C 虫 142.8	𧻆	𧻇	𧻈	2731F 虫 142.8	𧻉	𧻊	𧻋	27333 虫 142.8	𧻌	𧻍
UCS2003	GKX-1086.12	T5-495F	UCS2003	GKX-1088.19	T6-617A	UCS2003	GHC	UCS2003	GHC	GHC
2730D 虫 142.8	𧻎	𧻏	𧻐	27320 虫 142.8	𧻑	𧻒	𧻓	27334 虫 142.8	𧻔	𧻕
UCS2003	GKX-1086.22	T4-4677	UCS2003	GKX-1088.20	T6-617D	UCS2003	T5-4953	UCS2003	T5-4953	T5-4953

Koniec wykładu nr 2

Dziękuję za uwagę!
(następny wykład: 19.03.2018)