Karakterkódolás

Amikor leírjuk azt, hogy `motor`, betűket látunk a képernyőn. A számítógép azonban a szöveget is bitek formájában tárolja. Hogyan lehet betűket nullákkal és egyesekkel leírni? Ehhez először számokat rendelünk a betűkhöz, majd meghatározzuk e számok tárolásának módját.

Ebben a fejezetben megvizsgáljuk, hogyan működik ez a megfeleltetés, miért van többféle karakterkódolás, és mitől romolhatnak el az ékezetek egy fájl megnyitásakor.

1. Hogyan lesz a betűből szám?

A szövegben szereplő betűket, számjegyeket, írásjeleket és egyéb jeleket összefoglalóan **karaktereknek** nevezzük. Az A, a 7 és a ? egyaránt karakter. A szóköz is karakter, bár nem rajzolódik ki látható jelként.

Készítsünk egy nagyon egyszerű, **kitalált kódtáblázatot**! Állapodjunk meg abban, hogy három betűnek a következő számokat feleltetjük meg:

Betű	Hozzárendelt szám
A	1
B	2
C	3

Ezzel a táblázattal az ABA szöveget az `1, 2, 1` számsorozattal írhatjuk le. Aki ismeri a táblázatot, a számsorozatból vissza tudja állítani az eredeti betűket.

Mi történne, ha a fogadó fél egy másik táblázatot használna, amelyben az 1 a B betűt, a 2 pedig az A betűt jelenti? Ugyanabból az `1, 2, 1` sorozatból a BAB szöveget olvasná ki.

Az adó és a fogadó félnek ugyanazokat a kódolási szabályokat kell használnia. Ez a valódi szövegfájlok kezelésére is igaz.

2. Hogyan tároljuk a hozzárendelt számokat?

A számítógép a számokat kettes számrendszerben, nullák és egyesek sorozataként tárolja. Egy ilyen számjegy a **bit**. Nyolc bit együtt egy **bájt**.

Nyolc biten 256 különböző érték különböztethető meg. Ha ezeket előjel nélküli egész számként értelmezzük, a lehetséges értékek 0-tól 255-ig terjednek.

Az előbbi kitalált példában tároljunk minden betűkódot egy bájton:

Betű	Hozzárendelt szám	A szám nyolc biten
A	1	`00000001`
B	2	`00000010`
C	3	`00000011`

Az ABA szöveg így három bájtot foglal: az első bájt az 1-et, a második a 2-t, a harmadik ismét az 1-et

tárolja.

Ezzel már egy teljes tárolási szabályt adtunk meg: megmondtuk, melyik betűhöz milyen szám tartozik, és azt is, hogy egy számot hány biten tárolunk.

A **karakterkódolás** meghatározza, hogyan alakítjuk a szöveg karaktereit tárolható bájtsorozattá. A visszaalakítást **dekódolásnak** nevezzük.

A saját táblázatunk csak szemléltetésre szolgál. A gyakorlatban közösen elfogadott, szabványos kódolásokat használunk, hogy a különböző programok és számítógépek értelmezni tudják egymás szövegeit.

3. ASCII: egy közösen elfogadott kódtáblázat

Az egyik ilyen szabvány az **ASCII**. Teljes neve American Standard Code for Information Interchange. Az ASCII meghatározza, hogy az általa kezelt karakterekhez mely számok tartoznak.

Például a nagy A betűhöz a **65**, a nagy B betűhöz a **66** számot rendeli. Ezeket az értékeket megállapodás rögzíti: nem a betű alakjából számítjuk ki őket.

Karakter	ASCII-kód tízes számrendszerben	Tárolása egy bájton
A	65	`01000001`
B	66	`01000010`
a	97	`01100001`
0	48	`00110000`
Szóköz	32	`00100000`

Az ABBA szöveg ASCII szerint a 65, 66, 66, 65 értékű bájtokkal tárolható. A kisbetű és a nagybetű külön karakter: ezért az A és az a kódja eltér.

Az ASCII **128 kódértéket** használ, 0-tól 127-ig. Ennyi lehetőséghez 7 bit szükséges, mert 2 a hetediken 128. Ezért mondjuk, hogy az ASCII **7 bites kód**. A gyakorlatban rendszerint egy bájtot használunk egy ASCII-karakter tárolásához; ennek első, legnagyobb helyi értékű bitje 0.

Mi fér bele az ASCII-ba?

Az ASCII tartalmazza az angol ábécé nagy- és kisbetűit, a számjegyeket, írásjeleket, a szóközt és úgynevezett **vezérlőkaraktereket**. A vezérlőkarakterek nem betűként jelennek meg: például a szöveg sorokra tagolásában vagy tabulátorral történő elrendezésében van szerepük.

A magyar ékezetes betűk viszont nem szerepelnek benne. Az `alma` leírható ASCII-val, az `árvíz` már nem.

A kódolás által kezelt karakterek összességét **karakterkészletnek** nevezzük.

4. Hogyan kerültek be az ékezetes betűk?

Kézenfekvő megoldás volt a teljes bájt kihasználása: 7 bit helyett 8 bit használatával 128 helyett 256 kódérték áll rendelkezésre. A további helyekre ékezetes betűket és egyéb jeleket lehetett felvenni.

Az összes nyelv betűi azonban 256 helyre sem férnek el. Ezért különböző nyelvekhez és régiókhoz eltérő táblázatokat készítettek. Ezeket **kódlapoknak** nevezzük.

Például a **Windows-1250** közép-európai karaktereket, köztük magyar ékezetes betűket tartalmaz. A **Windows-1252** elsősorban nyugat-európai nyelvekhez készült.

Nézzünk egy valódi eltérést:

A fájlban tárolt bájt értéke	Windows-1250 szerint	Windows-1252 szerint
245	ő	õ

Ha egy Windows-1250 szerint mentett fájlt Windows-1252 szerint olvasunk, az ő helyén õ jelenhet meg. **A tárolt szám nem változott meg; a hozzá rendelt karakter változott meg.**

További régi kódlapnevekkel is találkozhatunk:

Név	Hol találkozhatunk vele?
IBM CP852	Régi DOS-os, közép-európai szövegekben.
ISO-8859-1, más néven Latin-1	Nyugat-európai szövegekben; az ő és az ű nincs benne.
ISO-8859-2, más néven Latin-2	Közép- és kelet-európai szövegekben; a magyar ékezetes betűket is tartalmazza.

Ezek nevét nem szükséges bemagolni. A lényeg az, hogy **a kódlap neve határozza meg, melyik táblázat szerint értelmezzük a bájtokat**.

Egyes programok az **ANSI** nevet használják egy régi Windowsos kódlap megjelölésére. Ez önmagában nem egyértelmű: a pontos kódlap környezettől függhet.

5. Unicode: legyen közös azonosítójuk a karaktereknek!

A kódlapokkal két probléma adódik. Egyrészt ugyanaz a szám különböző karaktereket jelenthet. Másrészt egyetlen ilyen kódlapba nem férnek bele a sokféle nyelv betűi és a műszaki jelek.

A **Unicode** olyan nemzetközi szabvány, amely közös rendszerben rendel számazonosítókat a karakterekhez. Tartalmazza számos mai és történeti írásrendszer betűit, valamint matematikai, műszaki és egyéb jeleket is.

Az alapgondolat egy nagy közös jegyzékhez hasonlítható: az A, az ő és a görög Ω is saját azonosítót kap benne. Nem kell nyelvenként más táblázatra áttérni.

Karakter	Unicode-azonosító tízes számrendszerben
A	65
ó	243
ő	337
Ω	937

Karakter	Unicode-azonosító tízes számrendszerben
€	8364

A karakterhez rendelt ilyen azonosítót **kódpontnak** nevezzük. Például az ő betű kódpontjának értéke 337. Ez egy azonosító, nem a betű mérete és nem a tárolásához szükséges bájtok száma.

A Unicode megtartotta az ASCII karaktereinek számértékeit: ezért az A kódja itt is 65.

Miért látunk betűket is a kódpontok leírásában?

A szakmai leírások a kódpontokat rendszerint **tizenhatos, más néven hexadecimális számrendszerben** adják meg. Ez csupán a számok rövidebb leírásának módja.

Ebben a számrendszerben a 0–9 számjegyeken kívül az A, B, C, D, E és F jelet is használjuk. Ezek rendre a 10, 11, 12, 13, 14 és 15 értéket jelentik. A helyi értékek jobbról balra 1, 16, 256 és így tovább.

Például a hexadecimális F3 értéke tízes számrendszerben: **15 × 16 + 3 = 243**.

A Unicode-kódpontok szokásos jelölése U+ előtaggal kezdődik, ezt követi a hexadecimális szám:

Karakter	Azonosító tízes számrendszerben	Ugyanaz a kódpont szokásos jelöléssel
A	65	U+0041
ó	243	U+00F3
ő	337	U+0151

A bevezető nullák a szám értékét nem változtatják meg. Az U+00F3 tehát azt jelenti: **a 243 értékű Unicode-kódpont**. Az U+ és az utána írt számjegyek magyarázó jelölések; nem ezeket a karaktereket mentjük el az ó helyett.

6. UTF-8: hogyan tároljuk a Unicode-karaktereket?

A közös azonosítók bevezetésével még nem oldottunk meg mindent. Az ő azonosítója például 337, miközben egy bájtban csak 0 és 255 közötti egész szám tárolható. Meg kell tehát határoznunk, hogyan írjuk le a karaktereket egy vagy több bájttal.

Erre szolgál például az **UTF-8**. Ez egy pontosan meghatározott szabályrendszer, amely a Unicode-kódpontokhoz tárolható bájtsorozatokat rendel.

A Unicode azt adja meg, melyik karakterről van szó. Az UTF-8 azt határozza meg, hogyan tároljuk azt bájtokkal.

Az UTF-8 egy kódolható kódponthoz 1, 2, 3 vagy 4 bájtot rendel. Az ASCII-karaktereket egy bájton, eredeti számértékükkel tárolja. A többi karakterhez több bájtot használ. Az alábbi példákban minden betűt egyetlen Unicode-kódponttal írunk le.

Karakter	Unicode-azonosító tízes számrendszerben	UTF-8-ban tárolt bájtok értékei, szintén tízes számrendszerben	Bájtok száma
A	65	65	1
ó	243	195, 179	2
ő	337	197, 145	2
€	8364	226, 130, 172	3

A táblázatban a vesszők a külön bájtokat választják el; nem részei a tárolt adatnak. Például az ó helyére két bájt kerül: az első értéke 195, a másodiké 179. A program a két bájtot együtt értelmezve állítja vissza az ó betűt.

Miért kell két bájt az ó betűhöz, ha a 243 még elférne egy bájtban? Mert az UTF-8 nem egyszerűen a kódpont számértékét írja egy bájtba. Az egybájtos alakot a 0–127 közötti ASCII-kódoknak tartja fenn; a többi karakter tárolására külön szabályt alkalmaz. Ennek bitenkénti működését később, kiegészítésként vezetjük le.

Hány bájt egy szó?

Nézzük a `mérő` szót! Az `m` és az `r` ASCII-karakter, ezért egy-egy bájtot foglal. Az é és az ő két-két bájtot igényel.

Betű	UTF-8-ban szükséges hely
m	1 bájt
é	2 bájt
r	1 bájt
ő	2 bájt
Összesen	**6 bájt**

A szó tehát **4 betűből áll, de 6 bájtot foglal**. Itt csak a négy betű tárolási helyét számoltuk; nem adtunk hozzá sortörést vagy fájleleji jelzést.

Kiegészítés: A Unicode egyes ékezetes betűket alapbetű és külön ékezet egymás utáni megadásával is képes leírni. A képernyőn ez egyetlen betűként jelenhet meg, miközben két kódpont tartozik hozzá. A fejezet számítási példáiban az ékezetes betűk önálló kódpontos alakját használjuk.

7. Mit jelent mindez a gyakorlatban?

Miért jelenik meg hibás szöveg?

Korábban láttuk, hogy két régi kódlap eltérően értelmezheti ugyanazt a bájtot. Hasonló hiba történhet akkor is, ha egy UTF-8-fájlt egy régi kódlap szerint nyitunk meg.

Az ó UTF-8-bájtjai tízes számrendszerben 195 és 179. Ha a program Windows-1252 szerint olvassa ezeket, két külön karakternek tekinti őket: a 195 értékhez az Ã, a 179 értékhez a ³ jelet rendeli. Így jelenik meg az ó helyén az Ã³ szöveg.

A fájl adatai még lehetnek helyesek; a program rossz szabály szerint olvassa őket.

A `.txt` és a `.csv` fájlnév-végződés önmagában nem árulja el a kódolást. A program néha megpróbálja felismerni azt, de a felismerés nem mindig biztos.

Megnyitás és átkódolás

Hibás ékezetek esetén először őrizzük meg az eredeti fájlt, és próbáljuk meg a készítésekor használt kódolással megnyitni.

- **Más kódolással történő megnyitáskor** ugyanazokat a bájtokat értelmezzük más szabály szerint. A fájl tartalmát ettől még nem írjuk át.
- **Átkódoláskor** a már helyesen felismert szövegből más kódolású bájtsorozatot hozunk létre és mentünk el.

Például egy Windows-1250-fájl UTF-8-ra alakításakor az ő egyetlen, 245 értékű bájtja helyett két bájt, 197 és 145 kerül az új fájlba. A betű ugyanaz marad, csak a tárolása változik.

Ha a helytelenül megjelenített szöveget mentjük újra, a hibát is elmenthetjük. Ha pedig olyan kódlapot választunk, amelyből egy szükséges karakter hiányzik, a program például kérdőjellel helyettesítheti azt. Ez már információvesztést okozhat.

Mérési adatok megnyitása

A **CSV** olyan szöveges fájlformátum, amely táblázatos adatokat tárol: a sorokon belüli mezőket például vessző vagy pontosvessző választja el. Egy mérési eredményeket tartalmazó fájlban ilyen oszlopnév is szerepelhet: `Hőmérséklet [°C]`.

A fájl beolvasásakor három külön beállításra lehet szükség:

- **Karakterkódolás:** hogyan értelmezze a program a betűket és egyéb jeleket?
- **Mezőelválasztó:** melyik jel választja szét az oszlopok adatait?
- **Tizedesjel:** pont vagy vessző választja el a szám egész és tört részét?

A hibás ékezet és az egy oszlopba összecsúszott táblázat tehát nem ugyanaz a probléma.

A betűtípus szerepe

A karakterkódolás mellett a megjelenítésnek is helyesen kell működnie. A **betűtípus** adja meg a jelek kirajzolható alakját. Előfordulhat, hogy a program helyesen azonosít egy karaktert, de a választott betűtípusban nincs hozzá rajzolat. Ekkor például üres négyzet jelenhet meg.

Az értelmetlen ékezetes betűsor gyakran kódolási problémára, a hiányzó jel helyén álló négyzet pedig megjelenítési problémára utalhat.

8. Kiegészítés: hogyan készül az UTF-8-bájtsorozat?

Az UTF-8 bájtjai nemcsak a kódpont bitjeit tartalmazzák. Bizonyos bitek azt jelzik, hogy egyetlen bájtról, több bájtból álló sorozat kezdetéről vagy annak folytatásáról van-e szó.

A következő táblázatban a leírt 0-k és 1-ek rögzített **jelzőbitek**. Az x helyekre a kódpont számértékének bitjei kerülnek:

Teljes hossz	Bitminta
1 bájt	`0xxxxxxx`
2 bájt	`110xxxxx 10xxxxxx`
3 bájt	`1110xxxx 10xxxxxx 10xxxxxx`
4 bájt	`11110xxx 10xxxxxx 10xxxxxx 10xxxxxx`

Például a `110` kezdet azt jelzi, hogy összesen két bájt tartozik össze. A második bájt `10` kezdete azt mutatja, hogy ez egy sorozat folytatása. Ezt nevezzük **folytató bájtnak**.

Az ó betű levezetése

Az ó kódpontja tízes számrendszerben 243. Kettes számrendszerben ez `11110011`, mert 128 + 64 + 32 + 16 + 2 + 1 = 243.

A 243 nagyobb 127-nél, ezért nem használhatjuk az egybájtos alakot. A kétbájtos mintában viszont 5 + 6 = 11 adatbitnek van helye, ami elegendő. A nyolc bit elé ezért három nullát írunk, majd az eredményt 5 és 6 bites részre bontjuk:

```
243 kettes számrendszerben: 11110011
11 bitre kiegészítve:      00011110011
Felbontás:                00011 110011
```

Ezután az első öt bit elé `110`, a következő hat bit elé `10` kerül:

```
Első bájt:    110 + 00011  = 11000011
Második bájt: 10  + 110011 = 10110011
```

Az első bájt értéke 195, a másodiké 179. Így kaptuk meg a korábbi táblázatban szereplő két számot.

Hexadecimálisan 195 = C3, 179 = B3, ezért ugyanez a bájtsorozat röviden **C3 B3** alakban írható. Egy bájtot két hexadecimális számjeggyel tudunk megadni.

A U+00F3 és a C3 B3 két különböző dolgot jelöl: az első az ó Unicode-azonosítója, a második az UTF-8-ban tárolandó két bájt értéke.

Megjegyzés: A táblázat a szerkezetet mutatja. Az UTF-8 további érvényességi szabályokat is meghatároz; például minden kódpontot a legrövidebb megengedett alakjával kell kódolni.

9. Kiegészítés: UTF-16 és a fájl eleji jelzés

Ugyanazokat a Unicode-karaktereket az UTF-8-tól eltérő módon is lehet tárolni. Ilyen kódolás az **UTF-16**, amely egy kódolható kódponthoz 2 vagy 4 bájtot használ.

Az A betű UTF-16-ban két bájtot foglal. Az egyik bájt értéke 0, a másiké 65. A két bájt sorrendjében azonban meg kell állapodni:

Sorrend neve	Az A tárolt bájtjai tízes számrendszerben	Magyarázat
Big-endian	0, 65	A nagyobb helyi értékű bájt kerül előre.
Little-endian	65, 0	A kisebb helyi értékű bájt kerül előre.

A két változat neve **UTF-16BE**, illetve **UTF-16LE**. A bájtsorrend itt a két bájt sorrendjét jelenti, nem az egy bájton belüli bitek megfordítását.

A fájl elejére speciális jelzés is kerülhet, amely segít a bájtsorrend felismerésében. Ezt **BOM**-nak (Byte Order Mark, bájtsorrendjelző) nevezzük. A BOM nem a szöveg részeként olvasandó betű vagy szó.

A BOM használata nem minden esetben kötelező: a kódolást és a sorrendet más módon is megadhatják. UTF-8-fájl elején is előfordulhat BOM, de ott csak a kódolásra utaló jelzés, mert UTF-8-nál nincs választható bájtsorrend.

10. Base64: hogyan legyen egy fájlból továbbítható szöveg?

Eddig szövegek tárolásáról volt szó. Most más feladatot vizsgálunk: van egy tetszőleges fájlunk, például egy képünk, és annak bájtjait olyan mezőben szeretnénk továbbítani, amely szöveget vár.

A fájl bájtjait nem kezelhetjük egyszerűen betűkként. Tetszőleges értékeik lehetnek, köztük olyanok, amelyeket a fogadó program vezérlőjelként értelmezne. Célszerű ezért az adatot egy szűk, előre rögzített készletből választott, nyomtatható jelekkel leírni.

Erre szolgál a **Base64**. Tetszőleges bemeneti bájtokat alakít át betűkből, számjegyekből és néhány további jelből álló szöveggé. Ilyen megoldást használnak például számos e-mail-melléklet átvitelekor.

Az UTF-8 szöveges karaktereket alakít bájtokká. A Base64 tetszőleges bájtokat alakít szöveges jelekké.

A Base64 nem titkosítás: aki ismeri a szabályt, kulcs nélkül vissza tudja állítani az eredeti adatot.

Miért éppen 64?

A Base64 a bemeneti biteket **hatos csoportokba** rendezi. Hat bit 64 különböző értéket adhat, 0-tól 63-ig. Mindegyik értékhez egy nyomtatható jelet rendel:

A hat bit értéke	Hozzárendelt jel
0–25	A–Z

A hat bit értéke	Hozzárendelt jel
26–51	a–z
52–61	0–9
62	+
63	/

Az értékek sorrendben követik egymást: 0 jelentése A, 1 jelentése B, 2 jelentése C, és így tovább.

Ez a Base64 saját táblázata, nem az ASCII-táblázat. Például a T itt a 19 értékű hatbites csoportot jelöli. Amikor magát a kimeneti T betűt ASCII-szövegként tároljuk, annak bájtértéke 84.

Példa: Man

Használjuk bemenetként a Man szöveg három ASCII-bájtját! Az ASCII-táblázat szerint az M értéke 77, az a értéke 97, az n értéke 110. Először ezeket írjuk fel nyolc biten:

```
M: 77  = 01001101
a: 97  = 01100001
n: 110 = 01101110
```

Ez összesen 24 bit. A bitek sorrendjén nem változtatunk, csak a csoporthatárokat helyezzük át: három nyolcbites csoport helyett négy hatbites csoportot készítünk.

```
Három bájt:         01001101 01100001 01101110
Négy hatos csoport: 010011 010110 000101 101110
```

A hatbites csoportokat számként értelmezzük, majd megkeressük a hozzájuk tartozó Base64-jelet:

Hat bit	Értéke tízes számrendszerben	Base64-jel
010011	19	T
010110	22	W
000101	5	F
101110	46	u

Például 010011 = 16 + 2 + 1 = 19, a táblázatban pedig a 19-hez a T tartozik. Az eredmény tehát **TWFu**.

Dekódoláskor ugyanezt végezzük fordítva: a négy jelből négy hatbites csoportot kapunk, majd a 24 bitet három nyolcbites bájtra bontjuk vissza.

Miért jelenik meg egyenlőségjel a végén?

Nem minden fájl bájtszáma osztható hárommal. Ha a végén csak egy vagy két bájt marad, az utolsó hatbites csoporthoz nulla biteket kell hozzáadnunk. A négykarakteres kimeneti csoport hiányzó helyeit = jelekkel töltjük ki.

A = **kitöltő jel**, nem tartozik a 64 adatjelhez, és nem jelent egy újabb hatbites adatcsoportot.

Az utolsó csoportban maradt adat	Base64-adatjelek	Kitöltés
3 bájt	4 jel	Nincs
2 bájt	3 jel	Egy =
1 bájt	2 jel	Két =

Példa egyetlen bájttal: legyen a bemenet az 1 értékű bájt, vagyis `00000001`.

1. Hat bit után elvágjuk: `000000` és `01`.
2. A második csoportot négy nullával egészítjük ki: `010000`.
3. A két csoport értéke 0 és 16. A Base64-táblázat szerint ezek jele `A` és `Q`.
4. Két adatjelünk van, ezért még két = kitöltő jelet írunk utánuk.

Az eredmény **AQ==**. Visszaalakításkor a két egyenlőségjel mutatja, hogy az utolsó csoport csak egy eredeti bájtot tartalmazott. A két adatjelből visszakapott 12 bit utolsó négy nullája kitöltés, ezt elhagyjuk.

Mennyivel nő az adat mérete?

A Base64 három bemeneti bájtból négy kimeneti karaktert állít elő. Ha ezeket ASCII-val vagy UTF-8-cal tároljuk, négy bájtot foglalnak.

Például **300 bájt adatból 400 bájt** lesz, a sortöréseket és egyéb kiegészítéseket nem számítva. Ez körülbelül **33% méretnövekedés**. Kis bemenetnél a kitöltés miatt az arány nagyobb is lehet.

A Base64 tehát **nem tömörítés**. Előnye az, hogy a bináris adat szöveges környezetben is továbbítható. Azt, hogy a kódolt szöveg tartalmazhat-e sortörést, az adott alkalmazás szabályai határozzák meg.

Ellenőrző kérdések

1. Miért kell ugyanazt a kódtáblázatot használnia az írónak és az olvasónak?
2. Miért nem írható le az `árvíz` szó ASCII-val?
3. Mit jelent a Unicode-kódpont? Hogyan kapcsolódik hozzá az UTF-8?
4. Mit jelöl az `U+00F3`, és mit jelöl a `C3 B3`?
5. Hány bájtot foglal a `mérő` négy betűje UTF-8-ban a példában használt alakokkal?
6. Mi a különbség a más kódolással történő megnyitás és az átkódolás között?
7. Miért nem szabad titkosításként használni a Base64-et?
8. Miért lesz három bájtból négy Base64-jel?

Megoldások

1. Eltérő táblázattal ugyanazokból a számokból más karaktereket állíthatnak vissza.
2. Az ASCII nem tartalmazza az `á` és az `í` betűt.
3. A kódpont a karakter számazonosítója. Az UTF-8 meghatározza az ehhez tartozó tárolandó bájtsorozatot.
4. Az `U+00F3` az ó Unicode-kódpontja. A `C3 B3` a két UTF-8-bájtjának hexadecimális értéke.

5. Hat bájtot: 1 + 2 + 1 + 2.
6. Megnyitáskor a meglévő bájtokat értelmezzük; átkódoláskor a felismert karakterekhez más kódolású bájtsorozatot hozunk létre.
7. Mert a kódolt adat kulcs nélkül visszaállítható.
8. Három bájt 24 bit; ebből négy hatbites csoport készül, és minden csoporthoz egy jel tartozik.

From:
https://edu.iit.uni-miskolc.hu/ - **Institute of Information Science - University of Miskolc**

Permanent link:
https://edu.iit.uni-miskolc.hu/tanszek:oktatas:szamitastechnika:karakterkodolas

Last update: **2026/10/01 15:17**