# 4.4. Кодування тексту

Розвиток кодування текстів відбувався одночасно з формуванням галузі IT і за цей час зазнав багато змін. Історично першим було кодування EBCDIC, яке дозволяло кодувати букви латинського алфавіту, арабські цифри і знаки пунктуації з керуючими символами.

## Стандарт кодування ASCII

Базовою точкою для розвитку сучасних кодувань текстів вважають кодування **ASCII** (*American Standard Code for Information Interchange*). Це кодування, що використовує 1 байт для опису одного символу. В ASCII описано 128 поширених символів - латинські літери, арабські цифри, розділові і деякі службові знаки: дужки, решітка, зірочка тощо (рис.1):

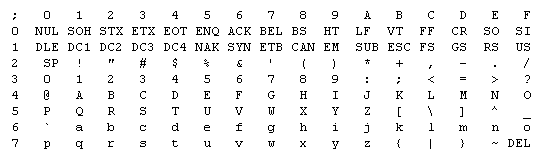


Рис.1. Базова таблиця кодування ASCII

Саме ці 128 символів з ASCII стали стандартом і в такому порядку присутні в інших кодуваннях. Але, за допомогою одного байту інформації можна закодувати не 128 (27), а 256 (28) різних значень, тому слідом за базовою версією ASCII з'явився цілий ряд розширених кодувань ASCII, в яких можна було крім 128 основних символів закодувати ще й символи національного кодування (наприклад, кирилицю).

Але в таблиці з символами кодування ASCII ці символи представлено в шістнадцятковому кодуванні. Наприклад, символ "зірочка" відповідає в ASCII шістнадцятковому числу 2A. Тут окрім арабських цифр використовуються латинські букви від A (десять) до F (п'ятнадцять).

## Розширені ASCII кодування CP866, KOI8-R з псевдографікою

Символи на екрані комп'ютера формуються на основі двох речей - наборів векторних форм (представлень) різних символів (вони знаходяться у файлах з шрифтами, які встановлено на комп'ютері) і коду, який дозволяє витягнути з цього набору векторних форм (файлу шрифту) саме той символ, який потрібно вставити.

За векторні форми символів відповідають файли шрифтів, а за кодування відповідає операційна система і програми, що в ній використовуються. Тобто, любий текст на комп'ютері представлено набором байтів, в кожному з яких закодовано один символ тексту.

Програма, що відображає текст на екрані (текстовий редактор, браузер тощо), при розборі коду зчитує кодування чергового символу і шукає відповідну для нього векторну форму в потрібному файлі шрифту, який під’єднано для відображення даного текстового документа.

Щоб закодувати потрібний символ (наприклад, національного алфавіту) має бути виконано дві умови - векторна форма цього символу повинна бути у вживаному шрифті і цей символ можна було б закодувати в розширених кодуваннях ASCII в один байт (рис.2-3). Тому, розширених кодувань ASCII існує багато. Лише для кодування символів кирилиці існує кілька варіантів розширеного кодування ASCII.

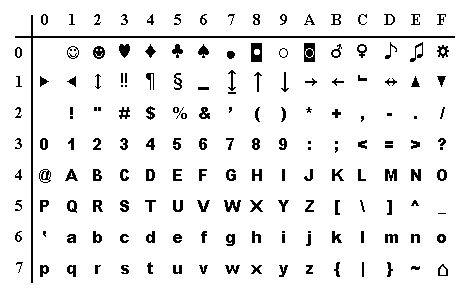


Рис.2. Символи розширеної ASCII (перша половина)

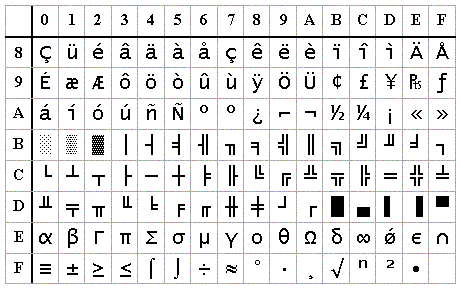


Рис.3. Розширене кодування IBM

### Кириличне кодування CP866

Першим з'явилося кодування тексту CP866, яке поширила компанія IBM. В ньому була можливість використовувати символи російського алфавіту і це кодування було розширеною версією кодування ASCII. Його верхня частина повністю збігалася з базовою версією ASCII (128 символів латиниці, цифр і знаків), яка представлена ​​на наведеному вище скріншоті, а нижня частина таблиці з кодуванням CP866 мала зазначений на скріншоті нижче вид і дозволяла закодувати ще 128 символів (російські літери і псевдографіка) (рис.4):

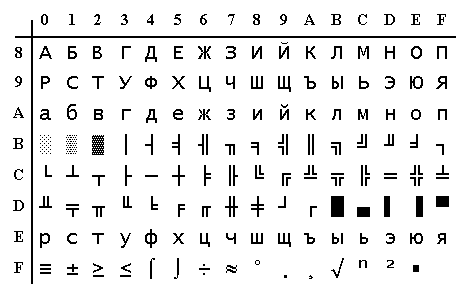


Рис.4. Таблиця кодування CP866

В правому стовпчику цифри кодування починаються з 8, тому цифри з 0 до 7 відносяться до базового кодування ASCII. Наприклад, російська буква «М» в кодуванні CP866 матиме код 9С (вона знаходиться на перетині відповідних рядка з 9 та стовпця з цифрою С в шістнадцятковій системі числення), який можна записати в одному байті інформації і за наявності відповідного шрифту з російськими символами ця буква без проблем відобразиться в тексті.

Кодування CP866 має велику кількість символів псевдографіки, оскільки розроблялося в роки, коли графічних операційних систем не було. А в DOS і подібних текстових операційних системах, псевдографіка дозволяла дещо урізноманітнити оформлення текстів. Тому, в кодуванні CP866 та інших розширених кодуваннях ASCII тих років так багато символів псевдографіки.

### Кириличне кодування KOI8-R

Кодування KOI8-R подібне до CP866 - кожен символ тексту кодується одним байтом. На скріншоті показано другу половину таблиці KOI8-R, оскільки перша половина повністю відповідає базовій версії ASCII (рис.5).

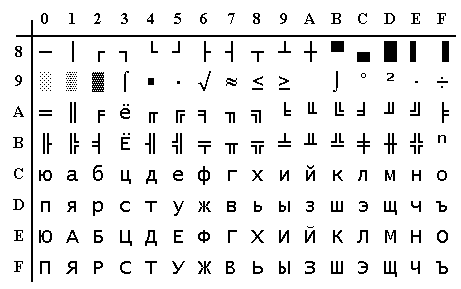


Рис.5. Таблиця кодування KOI8-R

Серед особливостей кодування KOI8-R можна відзначити те, що кириличні літери в її таблиці йдуть не в алфавітному порядку, як в кодуванні CP866. Якщо порівняти з базовою версією кодування ASCII (яка входить у всі розширені кодування), то можна помітити, що в KOI8-R кириличні літери розташовані в тих же елементах таблиці, що і співзвучні їм літери латинського алфавіту з першої частини таблиці. Це було зроблено для зручності переходу з кирилиці на латинку шляхом відкиданням всього одного біта (27 або 128).

Кодування KOI8-R використовувалася як транспортне кодування в Internet і як основне кодування в більшості безкоштовних операційних систем

### Кириличне кодування ISO 8859-5

Кодування ISO 8859-5 застосовується в більшості комерційних UNIX-сумісних операційних систем (рис.6).



Рис.6. Таблиця кодування ISO 8859-5

## Windows 1251 (розширена кодування ASCII)

Подальший розвиток кодувань тексту був пов'язаний з поширенням графічних операційних систем. Необхідність використання псевдографіки в текстах з часом зникла, тому її було забрано зі складу кодувань. В результаті виникла ціла група кодувань, які за своєю суттю і раніше були розширеними версіями ASCII (один символ тексту кодується всього одним байтом інформації), але вже без використання символів псевдографіки.

Такі кодування було розроблено американським інститутом стандартизації і відносилися до ANSI кодування. Для кирилиці популярним стало кодування **Windows 1251**.

Кодування Windows 1251 вигідно відрізнялося від ранішніх CP866 і KOI8-R. Тут, місце символів псевдографіки зайняли додаткові символи російської типографіки (окрім знака наголосу), а також символи, що використовуються в кириличних слов'янських мовах (української, білоруської тощо) (рис.7):

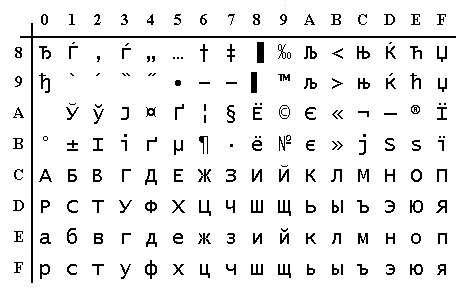


Рис.7. Таблиця кодування Windows 1251

## Заміна тексту на псевдосимволи

Через надмірну кількість кодувань текстів у виробників шрифтів і виробників програмного забезпечення постійно виникали проблеми. У разі невірно використаного кодування в тексті, у відвідувачів замість тексту міг відображалися безглуздий набір псевдосимволів.

Якщо символи, які закодовано за допомогою CP866 спробувати відобразити, використовуючи кодову таблицю Windows 1251, то з’являється набір псевдосимволів і повністю заміняє собою текст повідомлення (рис.8).

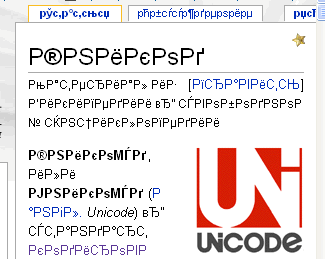


Рис.8. Відображення псевдосимволів у разі невідповідного кодування

Проблему вирішували в різний, але неефективний спосіб: користувачі часто використовували транслітерацію латинських букв; інтернет служби створювали складні перекодувальні таблиці.

Аналогічна ситуація часто виникає при створенні та налаштуванні сайтів, форумів або блогів, коли текст з кирилицею помилково зберігається не в тому кодуванні, яке використовується на сайті за замовчуванням або ж не в тому текстовому редакторі, який додає в код кодування від себе.

Зрештою така ситуація з множиною кодувань і постійними проблемами з відображенням символів національного алфавіту стала нестерпною і з'явилися передумови до створення нового універсального кодування, яке б замінило собою всі існуючі і вирішило б, нарешті, на корені проблему з появою не читаних псевдосимволів.

## Юнікод - універсальне кодування тексту (UTF 32, UTF 16 і UTF 8)

В світі окрім кирилиці існують інші алфавіти із значно більшою кількістю символів ніж в кирилиці. Наприклад, символів мовної групи південно-східної Азії є тисячі і їх неможливо описати в одному байті інформації, який виділяється для кодування символів в ASCII.

Для вирішення цієї проблеми було створено консорціум **Юнікод** (*Unicode - Unicode Consortium*) при співпраці багатьох лідерів IT індустрії (виробники програмного та апаратного забезпечення, розробники шрифтів), які були зацікавлені у появі універсального кодування тексту.

Першим кодуванням тексту, що вийшов під егідою консорціуму Юнікод, було кодування **UTF 32**. Цифра в назві кодування UTF 32 означає кількість біт, що використовується для кодування одного символу. Для кодування одного символу в новому універсальному кодуванні UTF 32 використано 4 байти інформації. В результаті чого файл з текстом, що закодований в UTF 32 матиме розмір в чотири рази більше ніж у розширеному кодуванні ASCII, зате з'явилася можливість закодувати 232 символів.

Але багатьом країнам з мовами європейської групи такої величезної кількості символів використовувати в кодуванні зовсім і не було необхідності, однак при використанні UTF 32 вони отримували чотириразове збільшення ваги текстових документів, а в результаті і збільшення обсягу Інтернет трафіку і обсягу збережених даних. Це багато і таке марнотратство собі ніхто не міг дозволити.

Подальшим розвитком універсального кодуванням стало **UTF 16**, яке вийшло настільки вдалим, що було прийнято за замовчуванням як базовий простір для всіх символів, які використовуються. Для кодування одного символу UTF 16 використовує два байти. В UTF 16 можна закодувати 65536 символів (216), що було прийнято за базовий простір в Юнікод.

Наприклад, в операційній системі Windows в Таблиці символів можна переглянути векторні форми всіх встановлених в системі шрифтів (рис.9).

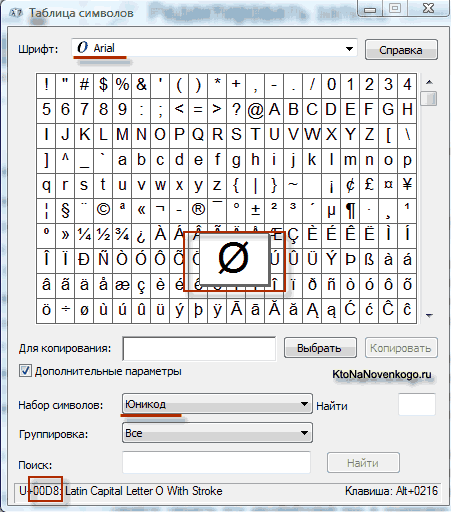


Рис.9. Перегляд шрифтів у Таблиці символів

Якщо вибрати в Додаткових параметрах набір символів Юнікод, то можна побачити для кожного шрифту окремо весь асортимент символів, що у ньому містяться. Якщо клацнути по будь-якому з цих символів можна побачити його багатобайтовий код в кодуванні UTF 16, що складається з чотирьох шістнадцяткових цифр.

Вдала версія UTF 16 знов не принесла переваг для європейських користувачів, бо у них після переходу від розширеної версії кодування ASCII до UTF 16 вага документів збільшувалася в два рази (один байт на один символ в ASCII і два байти на той же символ у кодуванні UTF 16).

Саме для вирішення цього в консорціумі Юнікод запропоновано кодування тексту змінної довжини **UTF 8**. UTF 8 є повноцінним кодуванням змінної довжини, тобто кожен символ тексту може бути закодований в послідовність довжиною від одного до шести байт. На практиці ж в UTF 8 використовується тільки діапазон від одного до чотирьох байт, оскільки більше за чотири байти коду нічого вже навіть теоретично не можливо уявити.

В UTF 8 всі латинські символи кодуються в один байт, подібно до ASCII. У разі кодування тільки латиниці, навіть ті програми, які не розуміють Юнікод, все одно прочитають те, що закодовано в UTF 8. Тобто, базова частина кодування ASCII перейшла в UTF 8. Кириличні символи в UTF 8 кодуються в два байти, а, наприклад, грузинські - в три байти.

Після створення кодувань UTF 16 і UTF 8 консорціум Юнікод вирішив основну проблему - тепер в шрифтах існує єдиний кодовий простір. Виробникам шрифтів залишається тільки заповнювати цей кодовий простір розробленими векторними формами символів тексту.

У наведеній вище Таблиці символів видно, що різні шрифти підтримують різну кількість символів. Деякі насичені символами Юнікоду шрифти можуть мати велику вагу. Тепер шрифти відрізняються не тим, що вони створені для різних кодувань, а тим, що виробник шрифту заповнив або не заповнив єдиний кодовий простір векторними формами символів.

### Кодування UTF 8 з BOM

Коли розробляли кодування UTF 16, було вирішено додати до неї можливість записувати код символу, як в прямій послідовності (наприклад, 0A15), так і в зворотній (150A). Для того, щоб програми розуміли в якій саме послідовності читати код символу в UTF 16 застосовують **BOM** (*Byte Order Mark*) - сигнатуру, яка містить три байти і додається на початок документів.

Консорціум Юнікод в кодуванні UTF 8 жодних BOM не передбачав і додавання сигнатури (додаткових трьох байтів в початок документа) деякими програмами заважає читати кодування Юнікод. Тому, завжди при збереженні файлів в кодуванні UTF 8 варто вибирати варіант **без BOM** (без сигнатури). Це допоможе запобігти від можливого відтворення псевдосимволів.

Деякі програми в Windows не вміють цього робити (зберігати текст у UTF 8 без BOM), наприклад, Блокнот Windows. Він зберігає документ у UTF 8, але додає на його початок сигнатуру (три додаткових байти). Ці байти сигнатури UTF 8 будуть завжди примушувати читати код в прямій послідовності. На серверах через цю дрібницю може виникнути проблема – з’являються псевдосимволи.

Для редагування документів сайту варто завжди цікавитися в якому кодуванні буде збережено сторінку (рис.10).

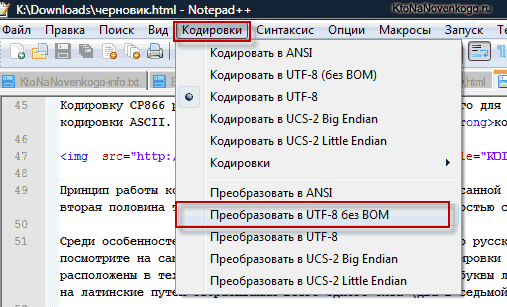


Рис.10. Вибір опції збереження у редакторі Блокнот

Інформацію про застосоване кодування слід прописати в HTML коді, щоб на сервері або локальному хості не виникло плутанини.

## Зазначення кодування тексту

Для зазначення кодування тексту в HTML використовується елемент META, який прописується між тегами HEAD:

<head>

<meta charset="utf-8">

</head>

Елемент META із зазначенням застосованого кодування краще ставити якомога вище в документі, щоб на момент появи в тексті першого символу не з базового кодування ANSI браузер вже повинен мати інформацію про те, як інтерпретувати коди цих символів.

## Контрольні питання

1. Зазначити відомості щодо кодування ASCII
2. З яких міркувань були створені розширені ASCII кодування?
3. В яких випадках відбувається заміна тексту на псевдосимволи?
4. Для вирішення яких проблем створено консорціум Юнікод?
5. Чим різняться між собою кодування UTF 32, UTF 16 та UTF 8?
6. Яку особливість кодування закладено в UTF 8?

## Використані джерела

1. Кодування тексту ASCII (Windows 1251, CP866, KOI8-R) и Юнікод (UTF 8, 16, 32) - <https://ktonanovenkogo.ru/vokrug-da-okolo/kodirovka-teksta-krakozyabry-ascii-yunikod-utf-8-rasshirennaya-ascii-windows-1251-cp866-koi8-r-problemy-s-kodirovkoj.html>
2. UTF-8 vs UTF-16 - <https://habr.com/ru/post/544084/>
3. Що таке кодування UTF-8 - <https://themewp.inform.click/chto-takoe-kodirovka-utf-8-rukovodstvo-dlja-neprogrammistov/>
4. Як змінити кодування текстового файлу на UTF-8 або Windows 1251 - <https://comp-security.net/как-изменить-кодировку-на-utf-8-windows-1251/>