

コンピュータ・サイエンスI

第8回 コンピュータでの文字の扱い(1)

人間科学科コミュニケーション専攻
白銀 純子

Copyright (C) Junko Shinozumi, Tokyo Woman's Christian University, 2015. All rights reserved.

第8回の内容

- コンピュータでの文字の扱い方(1)

Copyright (C) Junko Shinozumi, Tokyo Woman's Christian University, 2015. All rights reserved.

前々回の質問の回答

Copyright (C) Junko Shinozumi, Tokyo Woman's Christian University, 2015. All rights reserved.

16進数

- 練習問題: $(55)_8$ を16進数になおすこと

$$(55)_8 = (45)_{10}$$

$$\begin{array}{r} 16 \overline{) 45} \\ 16 \cdot 2 \cdots \text{余り: } 13 \\ \underline{ 0} \cdots \text{余り: } 2 \end{array}$$

$$(13)_{10} = (D)_{16}$$

$$(55)_8 = (2D)_{16}$$

16進数、つまり $(2D)_{16}$ は数!

- 1つ1つの桁の文字がなんであるかによって、数の大きさを表現
- 各桁の文字の並びの順序が違ってしまうと、違う数
- ✓ $(45)_{10}$ と $(54)_{10}$ は違う数!

$(2D)_{16}$ と $(D2)_{16}$ は違う数!

- 余りを並べる順序を間違わないように注意

Copyright (C) Junko Shinozumi, Tokyo Woman's Christian University, 2015. All rights reserved.

前回の復習

Copyright (C) Junko Shinozumi, Tokyo Woman's Christian University, 2015. All rights reserved.

小数を表現する方法(p. 10)

- 固定小数点方式
- 浮動小数点方式

Copyright (C) Junko Shinozumi, Tokyo Woman's Christian University, 2015. All rights reserved.

固定小数点方式[1](p. 10)

- 小数部分の桁数をあらかじめ決めておく方法

Ex. 2進数で表現した数の右から2ビットを小数部分とする場合

10進数の小数	2進数の小数	
0.00	00	00
0.25	00	01
0.50	00	10
0.75	00	11
1.00	10	00
1.25	10	01
1.50	10	10
...	...	...
3.75	11	11

➡ 小数部分は $1/2^2$ 刻み(0.25刻み)で表現

Copyright (C) Junko Shimomura, Tokyo Woman's Christian University, 2015. All rights reserved.

固定小数点方式[2](p. 10)

- 小数部分の桁数がnの場合: 2進数では、小数部分が $1/2^n$ 刻みで表現



nを大きくすると、それだけ小数部分を細かく表現可能

※ただし、実際コンピュータは小数も2進数で考えているが、人間が考えるときの便宜上、10進数で考えることが多い

Copyright (C) Junko Shimomura, Tokyo Woman's Christian University, 2015. All rights reserved.

固定小数点方式[3](p. 10)

- 小数を表す桁数が決まっていると...

Ex. 右から2桁を小数部分とすると...

$$(2 \div 1000)_{10} = (0.002)_{10} \rightarrow (0.00)_{10}$$

小数を正確に表現できない

何桁分の小数部分を持っているかは数値によって異なる
= 固定小数点方式で小数を表せる場合は少ない

➡ 浮動小数点方式

Copyright (C) Junko Shimomura, Tokyo Woman's Christian University, 2015. All rights reserved.

浮動小数点方式[1](p. 10)

- 小数: $D \times 10^n$ と表現できる

Ex.

$$\begin{aligned} &= 0.5 = 5 \times 10^{-1} \\ &= -0.0625 = -6.25 \times 10^{-2} \\ &= 0.0000000084 = 8.4 \times 10^{-9} \end{aligned}$$

どの数でも「 $\times 10^n$ 」の「10」の部分は同じ



小数を「 $D \times 10^n$ 」の形と考え、「D」と「n」だけ記憶しておく

Copyright (C) Junko Shimomura, Tokyo Woman's Christian University, 2015. All rights reserved.

浮動小数点方式[2](p. 10)

- 浮動小数点方式:
小数を「 $D \times 10^n$ 」と考え、「D」と「n」を記憶することで小数を表す方式

Ex.

D = 6.25, n = -3の場合: 0.00625
D = 6.25, n = -2の場合: 0.0625
D = 6.25, n = -1の場合: 0.625
D = 6.25, n = 0の場合: 6.25

nの数値が何かで、小数点が仮数の中を動くように見えるから「浮動小数点」と名づけられた

➡ D: 仮数部
➡ n: 指数部
と呼ぶ

Copyright (C) Junko Shimomura, Tokyo Woman's Christian University, 2015. All rights reserved.

浮動小数点方式[4](p. 10)

- コンピュータでは、指数部は2の累乗
→ 小数を「 $D \times 2^n$ 」として考え、「D」と「n」を記憶

Ex.

$$\begin{aligned} &\supset 0.5 = 1.0 \times (1/2) = 1.0 \times 2^{-1} \\ &\supset -0.0625 = -1.0 \times (1/16) = -1.0 \times 2^{-4} \end{aligned}$$

Copyright (C) Junko Shimomura, Tokyo Woman's Christian University, 2015. All rights reserved.

大きな数の表現(p. 10)

- 浮動小数点方式を利用して表現

Ex.

➢ $2000000000000 = 2 \times 10^{12}$

➢ $-42500000000000000 = -4.25 \times 10^{17}$

指数部が「+」の数になる

↓

コンピュータは「2」と「+12」、「-4.25」と「+17」を記憶しておく
(実際には、「 $\times 10^n$ 」ではなく「 $\times 2^n$ 」で表現)

Copyright (C) Iwate Shoinen, Tokyo Woman's Christian University, 2015. All rights reserved.

大きな数の表現[例](p. 10)

- Windowsの電卓のあるモード(オーバーフローをなかなかしないモード)で...



Ex. $1000000000000000 \times 1000000000000000$
 $= 1.e+30$

1.0×10^{30} という意味
(浮動小数点方式での表現)

※オーバーフローしにくいモードは、大きな数を浮動小数点方式で表現する
それだけたくさん桁がある数を表現できるので、オーバーフローしにくい

Copyright (C) Jyuko Shingo, Tokyo Woman's Christian University, 2015. All rights reserved.

桁落ち(1)

- 小数部分が無限のものを扱えるわけではない
 - 例えば割り算で割り切れない数や円周率

→ 小数部分を適当なところで切り捨てる
(四捨五入ではない)

例えば... $1 \div 3$: コンピュータは「0.3333...333」と考える

本来はこの後も無限に続く

コンピュータが扱える小数の桁数:
「有効桁数」と呼ぶ

Copyright (C) Imaho Shinzou, Tokyo Woman's Christian University, 2015. All rights reserved.

桁落ち(2)

- 小数部分が無限のものは適当なところまでで切り捨てられる
本来の数よりも、小数の桁数が小さくなる現象

桁落ち

Copyright (C) Tokyo Shoinen Gakuin Christian University 2015. All rights reserved.

桁落ちが起ること...

- 数が本来の数よりも小さくなってしまう
 - 微妙な数値が必要な場合には要注意
- 桁落ちをした数に大きな数をかけると、本来の数に大きな数をかけたときとの差が大きくなる
 - 例えば... $1 \div 3$ の結果が桁落ちし、 0.3331 になるとすると
 - 0.3331 に 100000 をかけたと 33300
 - 1×100000 を 3 で割ると(計算の順序を変えると) 33333.333
 - 本来の $1/3$ に 100000 をかけたとき、 $33333.333...$

➤ コンピュータで計算をするときは、計算の順番に注意
(割り算はなるべく後にすること)

➤ 例えば「 $1 \div 3 \times 100000$ 」の計算は、「 1×100000 」をしてから 3 で割る

Copyright (C) Esho-Shimizu, Tokyo Woman's Christian University 2015. All rights reserved.

桁落ち(補足)[1]

- 小数の桁が切り落とされたために、本来の数よりも小さくなってしまう現象
 - Ex. $1 \div 7$ の計算
 - 小数点第3位まで表現できるコンピュータ: 0.142

コンピュータの結果ではこの部分がなくなる
 - 本来の(人間が手で計算した)計算結果: 0.142857142857...
- 小数が出てくる計算では、**割り算の順序が重要**

Ex. 1: ある試験の得点(96, 68, 76, 92, 70, 88)の平均点を、小数点第1位まで表現できるコンピュータで計算

桁落ちが発生する時点

順序1(先に割り算): $96 \div 6 + 68 \div 6 + 76 \div 6 + 92 \div 6 + 70 \div 6 + 88 \div 6 = 16 + 11.3 + 12.6 + 15.3 + 11.6 + 14.6 = 81.4$

順序2(後で割り算): $(96 + 68 + 76 + 92 + 70 + 88) \div 6 = 490 \div 6 = 81.6$

割り算の順序によって計算結果が違う
(割り算を後にするると本来の計算結果に近い)

Copyright (C) Junichi Shigenaga, Tokyo University of Christianity, 2014. All rights reserved.

桁落ち[2]

Ex. 2: あるショッピングセンターでの売り上げ予測

➤ ある月に13209人来て、289157341円売り上げがあった。このとき、1000000人來場したときの売り上げ予測は? (小数点第1位まで表現できるコンピュータで計算)

普通に考えると... 売り上げ ÷ 実際の來場人数 × 1000000人
桁落ちが発生する時点

$$289157341 \div 13209 \times 1000000$$

$$= 21890.9 \times 1000000 = 21890900000$$

割り算を後にすると...

$$289157341 \times 1000000 \div 13209$$

$$= 289157341000000 \div 13209 = 21890933530.1$$

割り算の順序によって、
計算結果が大きく違う

Copyright (C) Iwako Shimomura, Tokyo Woman's Christian University, 2015. All rights reserved.

文字の表現

Copyright (C) Iwako Shimomura, Tokyo Woman's Christian University, 2015. All rights reserved.

文字の符号化(p. 13)

- 文字: コンピュータは整数に置き換えて扱う(番号をつけて扱う)
 - 文字を2進数で表現する(「**符号化**」と呼ぶ)
- 2進数で表現される文字集合
 - 半角英数文字
 - 図形文字
 - 制御文字
 - 多バイト文字
 - 図形文字

※文字集合: 文字の集まり

Copyright (C) Iwako Shimomura, Tokyo Woman's Christian University, 2015. All rights reserved.

図形文字と制御文字(p. 13)

- 図形文字**: 通常、画面に表示される文字
 - 人間が明示的に書いたり読んだりする文字
 - アルファベット, 数字, ひらがな, 漢字, 記号, etc.
- 制御文字**: 通常、画面に表示されない文字
 - コンピュータに何らかの制御をするための文字
 - 改行, TAB, ESC, etc.

Copyright (C) Iwako Shimomura, Tokyo Woman's Christian University, 2015. All rights reserved.

ASCII文字

Copyright (C) Iwako Shimomura, Tokyo Woman's Christian University, 2015. All rights reserved.

ASCII文字(p. 13)

- ASCII**: American Standard Code for Information Interchange
- 半角文字を表す文字集合
 - アルファベット大文字(26文字)
 - アルファベット小文字(26文字)
 - 数字(10文字)
 - 記号(スペース, 「.」, 「,」, etc.)

1文字を表すために、最低限7ビット必要
(6ビット: 64種類の情報, 7ビット: 128種類の情報)

※1文字を表す2進数の桁数(ビット数)は、どの文字でも同じ(つまり7ビット)

Copyright (C) Iwako Shimomura, Tokyo Woman's Christian University, 2015. All rights reserved.

図形文字(p. 13)

- ASCII: 情報量が7ビットで収まるように、扱う文字を取り決めた文字集合
 - アルファベット(大文字・小文字): 52文字
 - 数字: 10文字
 - 記号(スペースを含む): 33文字

図形文字: 合計95文字

Copyright (C) Iwate Shoinen, Tokyo Woman's Christian University, 2015. All rights reserved.

番号例(p. 14)

番号	文字	番号	文字	番号	文字
47	0	65	A	97	a
48	1	67	B	98	b
49	2	68	C	99	c
50	3	69	D	100	d
51	4	70	E	101	e
52	5	71	F	102	f
53	6	72	G	103	g
54	7	73	H	104	h
55	8	74	I	105	i
56	9	75	J	106	j

「数」としての0~9ではなく、「文字」としての0~9

Copyright (c) 1999 Shingene, Tokyo Waseda Christian University. 2011-12 slide changed.

番号の決まり(p. 14)

- 7ビットで1文字を表現 = 128文字表現可能
 - 図形文字: 95文字
 - 32番～126番までが図形文字
 - 残り33文字: 制御文字を表現
 - 0番～31番と127番が制御文字
 - ※32番のスペースを、制御文字と考えることもある
- アルファベットの大文字と小文字の番号に規則
 - 小文字の番号は、大文字の番号に $32(=2^5)_{10} = (100000)_2$ を足したもの
 - Ex. A: $(65)_{10} = (01000001)_2$, a: $(97)_{10} = (01100001)_2$
 - 大文字 $\Leftrightarrow$ 小文字の変換はやりやすい

Copyright (C) Jun-ichi Shiragami, Tokyo Woman's Christian University, 2015. All rights reserved.

制御文字例(p. 14)

- BS(8番): Back Space
- HT(9番): TAB
- **LF(10番): UNIX系OSでの改行**
- **CR(13番): Mac OSでの改行**
 - Windowsでの改行は、13番と10番を組み合わせて「**CRLF**」の2つの制御文字で表現される
 - ✓ つまり、Windowsでは1つの改行が2文字分(2ビット)
- DEL(127番): Delete

※OS: Operating System(オペレーティングシステム)

Copyright (C) Iwako Shirogane, Takuya Watanabe, Christian University, 2015. All rights reserved.

ビット数[1](p. 14)

- コンピュータでは8ビットを1つの単位として扱うことが多い
→ ASCII文字も8ビットで表現すると扱いやすい
 - 8ビットのうち、7ビット分(2進数で7桁目まで)で文字を表現する
 - 残りビット(2進数で8桁目)に常に0を入れておく
 - ASCII文字としては無駄なビット
 - 日本語を表現するとき利用

例えば...

A: 65番(10進数)

= 1000001番(2進数)

= 01000001番(2進数、コンピュータ内での表現)

↑

ASCII的には無駄な(何も利用していない)ビット

Copyright (C) Jinko Shirogane, Tokyo Woman's Christian University, 2015. All rights reserved.

ビット数[2](p. 14)

- 8ビットで1文字を表現 = 1バイトで1文字を表現

「1バイト文字」と呼ばれる

Ex. 「Hello, my name is John.」

- アルファベット: 17文字
- 記号: 2文字
- スペース: 4文字

} 23文字 = 23バイト

Copyright (C) Inooka Shingane, Tokyo Woman's Christian University, 2015. All rights reserved.

ちなみに...

- アスキーアートも文字コードのASCIIから(ASCII art)
 - アスキーアート: 文字だけで作った絵
 - 感情を表す「(^_^)」のような単純なものから、人や動物に見えるものまで様々
- アスキーアートの例
 - <http://ja.wikipedia.org/wiki/%E3%82%A2%E3%82%B9%E3%82%AD%E3%83%BC%E3%82%A2%E3%83%BC%E3%83%88>
 - <http://bhdaa.sakura.ne.jp/zukan/>

Copyright (C) Iwako Shimozono, Tokyo Woman's Christian University, 2015. All rights reserved.

多バイト文字

Copyright (C) Iwako Shimozono, Tokyo Woman's Christian University, 2015. All rights reserved.

背景[1](p. 15)

- コンピュータは主にアメリカで作られ発展
 - 使う人も、専門家だけだった
 - 当初は扱う文字はアルファベット・数字・いくつかの記号でよかった
- コンピュータが全世界に普及
 - 使う人も、専門家だけではなくなった
 - 英語圏以外の言語圏に対応する必要が出てきた
 - 様々な言語圏の文字に対応

Copyright (C) Iwako Shimozono, Tokyo Woman's Christian University, 2015. All rights reserved.

背景[2](p. 15)

- 様々な言語圏の文字: 英語圏の文字と同様に2進数で表現する必要性
 - 英語圏の文字: 128文字で表現可能
 - 1バイト分(256文字分)のうち、128文字分は英語圏の文字
 - 英語圏以外の文字: 128文字以上必要な場合も
 - 日本語
 - 中国語
 - 韓国語
 - etc.

1文字を複数のバイト(多バイト)で表現

Copyright (C) Iwako Shimozono, Tokyo Woman's Christian University, 2015. All rights reserved.

文字化け(p. 15)

- 多バイト文字の出現により、文字化けが発生
- 文字化けの原因
 - フォントの問題
 - 文字集合の符号化方式の問題

「文字コード」と呼ぶ

詳細な理由は何であれ...

要は文字を表す2進数(0と1の並び)を、コンピュータが理解していないために発生

➢ その2進数をどのような形でディスプレイに表示して良いかをコンピュータが理解していないため

Copyright (C) Iwako Shimozono, Tokyo Woman's Christian University, 2015. All rights reserved.

フォントの問題[3](p. 15)

- **機種依存文字**: コンピュータによって表現のしかたが違う文字
 - それぞれの文字を表現するビット列が、コンピュータによって異なる
 - 1文字1文字を表現するビット列は、JIS(日本の国家規格)などで決まっている
 - コンピュータの環境に依存しない
 - 規格で決められた文字に含まれていない文字もある
- **機種依存文字**
 - Ex. 丸付き数字(①, ②, ...), ローマ数字(I, II, ...), etc.
- **外字**: 登録されていない文字を、利用者が作ったもの
 - 人名漢字などを作ることが多い
 - 作ったコンピュータでしか使えない

Copyright (C) Iwako Shimozono, Tokyo Woman's Christian University, 2015. All rights reserved.

符号化方式の問題[1](p. 15)

- 符号化: 1つの文字を2進数(ビット列)として表現すること
- ある1つの文字を表現するビット列が複数通り存在する場合
 - 半角英数の文字はASCIIの1通りだけ
 - ・他にも存在するが、ASCIIが世界標準
 - ・大部分のコンピュータはASCIIを利用
 - 日本語は複数通り存在

Copyright (C) Junko Shimamura, Tokyo Woman's Christian University, 2015. All rights reserved.

符号化方式の問題[2](p. 15)

- ある文書で使われている符号化方式をコンピュータが判別できないときに文字化け
 - 判別できなければ、コンピュータが普段使っている符号化方式で表示しようとすることが多い
- Ex. Windowsで文書を開いた場合
 - 文書はJISで書いてあり、WindowsはJISであると判別できなかった
 - WindowsはShift_JISとして開こうとする
 - 文書は文字化けして表示される

※ソフトウェアによっても、符号化方式を判別できるものとできないものがある
(Ex. Windowsのメモ帳は、JISなどは判別できない)

Copyright (C) Junko Shimamura, Tokyo Woman's Christian University, 2015. All rights reserved.

日本語の符号化方式

Copyright (C) Junko Shimamura, Tokyo Woman's Christian University, 2015. All rights reserved.

日本語の文字(p. 15)

- 日本語固有の文字
 - ひらがな
 - カタカナ
 - 漢字
 - かぎカッコ
 - 句読点
 - etc.

Copyright (C) Junko Shimamura, Tokyo Woman's Christian University, 2015. All rights reserved.

日本語の文字(p. 15)

- ASCII
 - 1文字を8ビットで表現→全部で256文字分表現可能
 - 現状で128文字存在(128文字分利用されている)
- 日本語
 - ひらがな: あ〜ん(ゐ、ゑなどの旧字を含む)、濁音・半濁音、小文字(「ぁ」、「ぃ」など)
 - カタカナ: ア〜ン(ヰ、ヱなどの旧字を含む)、濁音・半濁音(ヴを含む)、小文字(「ァ」、「ィ」、「カ」、「ケ」など)

169文字

ひらがな・カタカナだけでもASCIIでは表現できない

Copyright (C) Junko Shimamura, Tokyo Woman's Christian University, 2015. All rights reserved.

日本語文字集合の規格(p. 16)

- 現状での日本語文字集合の規格: JIS X 0208:1997
 - ひらがな・カタカナ・漢字・非漢字文字で6879個

JIS第1水準(使用頻度の高い漢字): 2965個
JIS第2水準(使用頻度の低い漢字): 3390個

➢ $2^{13} = 8192$ なので、13ビットで表現可能
➢ コンピュータ処理では、バイト単位(8ビット単位)が好都合

16ビット(2バイト)で日本語1文字を表現

Copyright (C) Junko Shimamura, Tokyo Woman's Christian University, 2015. All rights reserved.

ASCII文字との区別(p. 16)

日本語の文書

- 日本語の2バイト文字
- ASCIIの1バイト文字

混在

日本語の2バイト文字(JIS X 0208)とASCIIの1バイト文字は区別する必要
(1つの文書の中で、どれが2バイト文字でどれが1バイト文字か)

- モード切り替えによる区別方法
- ASCII文字の番号を避ける区別方法

Copyright (C) Junko Shimozono, Tokyo Woman's Christian University, 2015. All rights reserved.

モード切り替え(p. 16)

文字集合切り替えのための特別な記号を用意

- ここから先はASCII文字
- ここから先は日本語文字
- ここから先は中国語漢字
- etc.

エスケープシーケンス

通常の文書では頻りに文字集合が切り替わることがなく、同じ文字集合に属する文字が現れることが多いという性質を利用

- 国際標準規格: ISO-2022
- 日本語に適用したもの: ISO-2022-JP

Copyright (C) Junko Shimozono, Tokyo Woman's Christian University, 2015. All rights reserved.

ISO-2022-JPの例(p. 16)

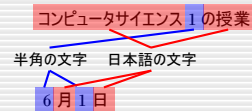
ESC \$B	F	KY	\$N	ESC (B	JP	ESC \$B	\$@	!#	ESC (B	Yn
	日	本	は		JP		だ	。		

- 「ESC \$B」や「ESC (B」、「Yn」などがエスケープシーケンス
- 「F」や「KY」、「\$N」などは、2バイト文字をASCII文字で表現した場合の文字
(2バイト文字は、1バイト文字2文字の組み合わせで表現できる)
- エスケープシーケンス「ESC \$B」や「ESC (B」は、3バイトずつ
- 「Yn」は改行を表し、半角文字の扱いなので、改行の前に2バイト文字がある場合は、改行と2バイト文字との間にもエスケープシーケンス
- 文章の開始(終了)が1バイト文字の場合は、文章の先頭(終了)にエスケープシーケンスはなし
- 文章の開始が2バイト文字の場合は、文章の先頭にエスケープシーケンスあり

Copyright (C) Junko Shimozono, Tokyo Woman's Christian University, 2015. All rights reserved.

モード切り替えの考え方[1](p. 16)

同じ文字集合に属する文字が現れることが多い



- 上側の文章: 日本語文字がいくつか続いた後、半角文字が少しあり、また日本語文字が続く
- 下側の文章: 日本語文字と半角の文字が交互にある

Copyright (C) Junko Shimozono, Tokyo Woman's Christian University, 2015. All rights reserved.

モード切り替えの考え方[2](p. 16)

普通の日本語の文章は、日本語の文字がずっと続き、たまに半角文字が出てくることが多い

ある言語の文章では、その言語の文字がずっと続き、別の言語の文字はところどころに出てくる

- 頻りに文字集合が切り替わるわけではない(ある言語と別の言語の文字が1文字ずつ交互に出てきたり、ということは少ない)
- 文字集合がどこで切り替わっているか、わかるようにしておけば良い

Copyright (C) Junko Shimozono, Tokyo Woman's Christian University, 2015. All rights reserved.

モード切り替えの考え方[6](p. 16)

- 日本語文字: 1文字2バイト
- ASCII文字: 1文字1バイト
 - 改行はASCII文字扱い
 - Mac OSやUNIX系OSだとASCII文字1文字分 = 1バイト
 - WindowsだとASCII文字2文字分 = 2バイト
- エスケープシーケンス: 1つ3バイト
 - 東女はTWCUです。
 - 日本語文字: 6文字 = 12バイト
 - ASCII文字: 4文字 = 4バイト
 - エスケープシーケンス: 3個 = 9バイト

※この文章は改行なしとする

Copyright (C) Junko Shimozono, Tokyo Woman's Christian University, 2015. All rights reserved.

モード切り替えの問題(p. 17)

- 文書を先頭から順番に見ていく場合には問題ない
- 文書を途中から見ていくときに問題が生じる
 - 見始めた途中の文字が、ASCII文字か日本語文字か、エスケープシーケンスかが判別できない

Ex. 見始めた途中の文字が「70」番だった場合

- ASCII文字の「F」?
- 日本語文字の一部?
- 韓国語の一部?

検索や置換などの文書処理に時間がかかる

Copyright (C) Junko Shimozono, Tokyo Woman's Christian University, 2015. All rights reserved.

ASCII文字の番号を避ける(p. 17)

- ASCIIで使われていない番号を2バイト文字の番号にあてる方法
 - EUC(日本語のものをEUC-JP)
 - 第1バイト(前半の8ビット)と第2バイト(後半の8ビット)両方でASCII領域が避けられている
 - SJIS(Shift JIS)
 - 第2バイト(後半の8ビット)ではASCII領域も使われている
 - 文章のバイト数は、単純に、日本語文字で2バイト、ASCII文字で1バイトで数えれば良い

Copyright (C) Junko Shimozono, Tokyo Woman's Christian University, 2015. All rights reserved.

EUCとSJIS[1](p. 17)

- ASCII文字: 8個の0と1で、1文字分を表現
 - 実際には、7個の0と1で1文字分を表現
 - 8ビット目は必ず0

0000000から1111111まで、フルに使ってASCII文字を1文字ずつ表現

XXXXXXXX

= ASCII文字は、0XXXXXXXX という形
Ex. 「a」を0と1で表現すると: 01100001

8ビット目が1になる番号(1XXXXXXXXという形の番号)はASCII文字ではない

Copyright (C) Junko Shimozono, Tokyo Woman's Christian University, 2015. All rights reserved.

EUCとSJIS[2](p. 17)

- ASCIIで使われていない番号を2バイト文字の番号にあてる方法
 - EUC(日本語のものをEUC-JP)
 - 第1バイト(前半の8ビット)と第2バイト(後半の8ビット)両方でASCII領域が避けられている
 - SJIS(Shift JIS)
 - 第2バイト(後半の8ビット)ではASCII領域も使われている
 - 文章のバイト数は、単純に、日本語文字で2バイト、ASCII文字で1バイトで数えれば良い

例えばある文章が...

1から始まっているから日本語文字

00110110100101101010010101101010

0から始まっているからASCII文字

Copyright (C) Junko Shimozono, Tokyo Woman's Christian University, 2015. All rights reserved.

Webやメールでの文字化け(p. 18)

- Webページや電子メール
 - 文字コードの指示が文書中に書かれている場合
 - 「charset=iso-2022-jp」や「charset=Shift_JIS」など
 - ソフトウェアは指示通りに文字コードを解釈し、表示
 - 文字コードの指示が文書中に書かれていない場合
 - ソフトウェアは文書のデータの特徴から文字コードを判別
- 文字化けが発生する場合
- 文書中の文字コードの指示が間違っている場合
 - 文字コードの指示がなく、文字コードを判別できなかった場合

Copyright (C) Junko Shimozono, Tokyo Woman's Christian University, 2015. All rights reserved.

Unicode

Copyright (C) Junko Shimozono, Tokyo Woman's Christian University, 2015. All rights reserved.

言語圏ごとの文字コード(p. 18)

- これまでの多バイト文字の扱い:
異なる言語圏ごとに文字集合を作成
 - 様々な文字集合ができてしまって不便
 - コンピュータネットワークの国際化が進んだ
 - コンピュータの資源が豊富になった
- ↓
- 国際文字集合規格として各文字集合を統一化

Copyright (C) Tokyo Shingaku, Tokyo Woman's Christian University, 2015. All rights reserved.

統一文字コード(p. 18)

- **Unicode**
 - ASCII
 - ラテン文字
 - 日本語
 - 韓国語
 - 中国語
 - ベトナム語
 - ギリシャ文字
 - 記号
 - etc.
- Unicodeバージョン5.2.0で
107361文字

Copyright (C) Tokyo Shingaku, Tokyo Woman's Christian University, 2015. All rights reserved.

UTF-8(p. 18)

- Unicodeでの代表的な符号化方式
- 1文字を1～6バイトの可変長(文字によってバイト数が異なる)で符号化する方式
 - ASCIIやISO-2022-JP、Shift JIS、EUC-JPは1文字を全て同じバイト数で表現している
- OS(WindowsやMacなどのオペレーティングシステム)でファイル名などの内部処理に利用
 - 半角英数を符号化した結果が、ASCII文字と全く同じになるため、従来のシステムと相性が良い

④ 現在、Unicodeへの移行が急速に進んでいる
➢ ただし、以前から使われてきたファイルを移行するのは大変なので、
完全移行には時間がかかる

Copyright (C) Tokyo Shingaku, Tokyo Woman's Christian University, 2015. All rights reserved.