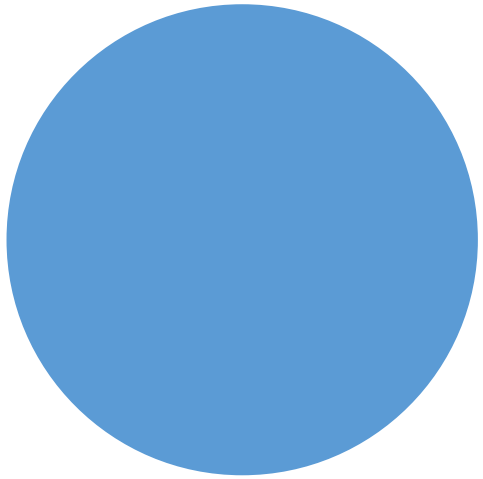
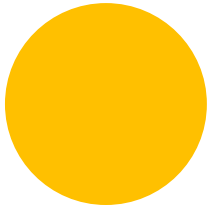



CSARCH Lecture Series: Character Data Representation

Sensei RL Uy

College of Computer Studies

De La Salle University

Manila, Philippines

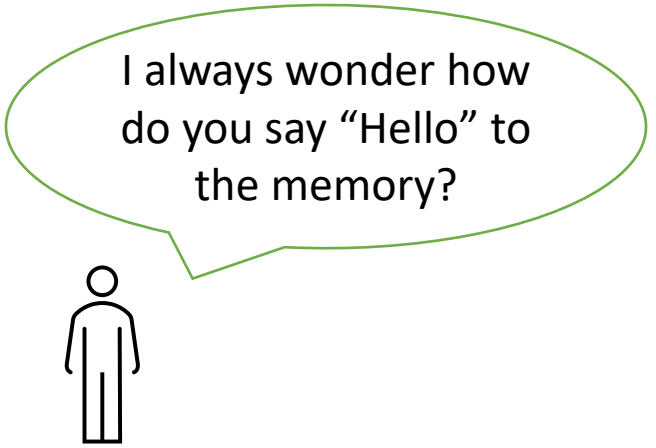
Copyright Notice

Overview

Reflect on the following question:

- How do you represent the character “A” or the string “Hello”?

```
int main()
{
    char var[] =“Hello”;
}
```



Overview

- This sub-module introduces the concept of representing character data using ASCII and Unicode
- The objectives are as follows:
  - ✓Describe the process of representing character data using ASCII
  - ✓Describe the process of representing character data using UNICODE

Character Data Representation

- There are two methods of representing character data:
  - ASCII
  - Unicode

ASCII

- American Standard Code for Information Interchange (ASCII) standard for character representation was approved in 1963 with major revisions in 1967 and 1986
- Uses 7-bit binary to encode 128 characters

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Divided into 4 groups of 32 characters:

- ASCII 00h to 1Fh: control characters (e.g., carriage return, line feed)
- ASCII 20H to 3Fh: punctuation symbols, numeric symbols and other special characters
- ASCII 40h to 5Fh: upper case alphabet characters and other special characters
- ASCII 60h to 7Fh: lower case alphabet characters and other special characters

source: Lanoxxth Shaddow via Wikipedia commons (CC0)

Example

```
int main()
{
    char var[] =“Hello”;
}
```

label	Address (hex)	Memory data (binary)	Comment
	0005	0000 0000	\0 (null)
	0004	0110 1111	“o”
	0003	0110 1100	“l”
	0002	0110 1100	“l”
	0001	0110 0101	“e”
var	0000	0100 1000	“H’

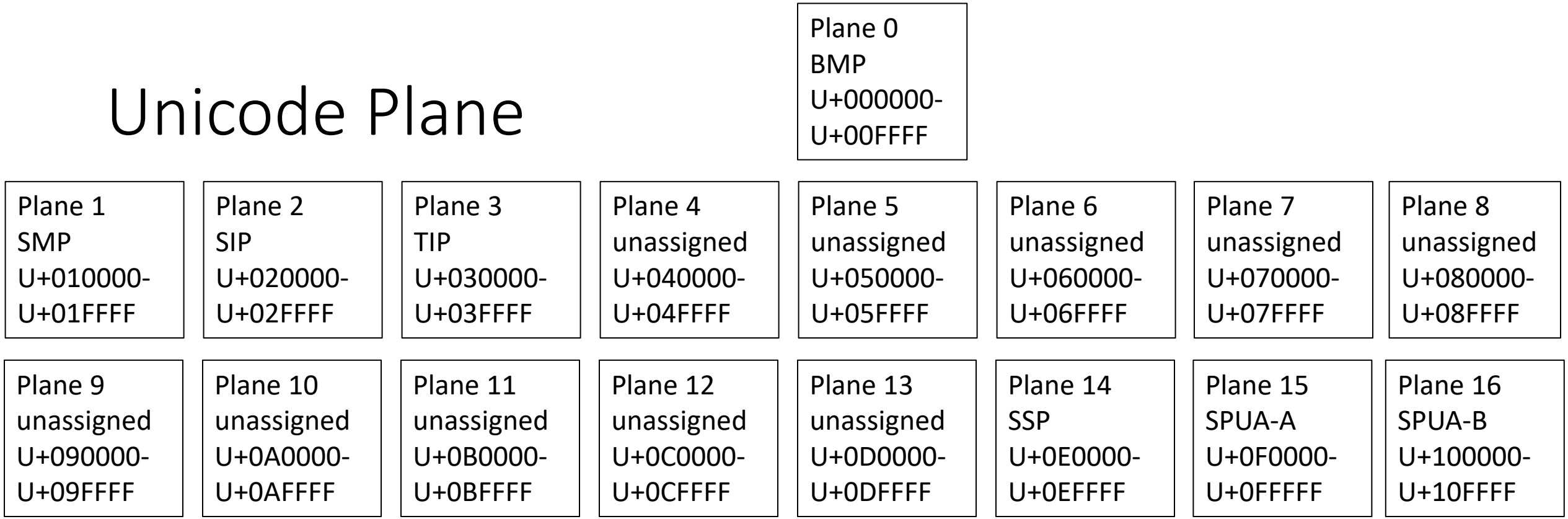
ASCII

- ASCII can only represent 128 characters.
- Other characters such as CJK (Chinese, Japanese, Korean) cannot be represented
- Thus, another representation scheme is needed to represent more symbols

Unicode

- Unicode: 32-bit character set
  - Each character (or code unit) is either 1,2 or 4 bytes
  - Introduced as a standard (Unicode 1.0) on October, 1991
  - Current version is Unicode 13.0 (as of March, 2020)
  - Most of the world's alphabets, plus symbols
    - roughly 154 scripts such as Han, Latin, Arabic and others
    - 143,800+ characters with defined code point
  - Example:
    - ₱ (Philippine peso) is U+20B1
    - Σ (Greek capital sigma) is U+03A3
    - A (Capital A) is U+0041

Unicode Plane



- In Unicode, character code points are not assigned linearly
- It uses a concept of Unicode “plane”
- 17 Unicode planes, each plane has 65536 code points → 1,114,112 code points of which 974,530 are for public assignment

Basic Multilingual Plane (BMP)

- Basic Multilingual Plane (BMP)
  - Plane 0 is the Basic Multilingual Plane (BMP)
  - Of interest:
    - contains characters for almost all modern languages
    - Code point 0000 – 007F (C0 Controls and Basic Latin) is the original ASCII
    - Code point 0080 – 00FF (C1 Controls and Latin-1 supplement) is the original “extended ASCII” also known as LATIN-1 (punctuation, mathematic, currency)
    - Code point 0370 – 03FF (Greek and Coptic)
    - Code point 4E00 – 9FFF (CJK Unified Ideographs)

Plane 0
BMP
U+000000-
U+00FFFF

Other planes

- Plane 1 – Supplementary Multilingual Plane (SMP)
- Plane 2 – Supplementary Ideographic Plane (SIP)
- Plane 3 – Tertiary Ideographic Plane (TIP)
- Plane 14 – Supplementary Special-purpose Plane (SSP)
- Plane 15 – Supplementary Private Use Area (SPUA-A)
- Plane 16 – Supplementary Private Use Area (SPUA-B)

Unicode Transformation Format (UTF)

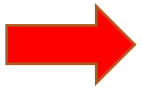
- Unicode code points are encoded using UTF. There are 3 encodings:
  - UTF-8: variable-length encoding
  - UTF-16: variable-length encoding
  - UTF-32: fixed-length encoding

UTF-8

- Variable length encoding from 1 to 4 bytes
- 8-bit code unit is called an octet
- Representation as follows

Bits of code point	First code point	Last code point	# of bytes	Byte 1	Byte 2	Byte 3	Byte 4
7	U+0000	U+007F	1	0xxxxxxx			
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx		
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx	
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

UTF-8



Bits of code point	First code point	Last code point	# of bytes	Byte 1	Byte 2	Byte 3	Byte 4
7	U+0000	U+007F	1	0xxxxxxx			
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx		
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx	
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Example: U+41

binary: 100 0001 (needs 1 byte)

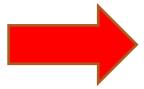
UTF-8:

01000001

hex: 0x41

UTF-8

Bits of code point	First code point	Last code point	# of bytes	Byte 1	Byte 2	Byte 3	Byte 4
7	U+0000	U+007F	1	0xxxxxxx			
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx		
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx	
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx



Example: U+20AC

binary: 0010 0000 1010 1100 (needs 3 bytes)

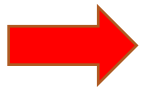
UTF-8:

11100010 10000010 10101100

hex: 0xE282AC

UTF-8

Bits of code point	First code point	Last code point	# of bytes	Byte 1	Byte 2	Byte 3	Byte 4
7	U+0000	U+007F	1	0xxxxxxx			
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx		
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx	
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx



Example: U+245D6

binary: 0010 0100 0101 1101 0110 (needs 4 bytes)

UTF-8:

11110000 10100100 10010111 10010110

hex: 0xF0A49796

UTF-16

- Variable length encoding either 1 or 2 16-bit code units
- 16-bit code unit is called a wyde (wide)

UTF-16

- Code points from U+0000 to U+FFFF, represent as is
- Code points from U+010000 to U+10FFFF
  - 0x010000 is subtracted from the code point, leaving a 20-bit number in the range 0x00000 to 0xFFFFF.
  - The top ten bits (a number in the range 0x0000 to 0x03FF) are added to 0xD800 to give the first 16-bit code unit or *high surrogate*, which will be in the range 0xD800..0xDBFF.
  - The low ten bits (also in the range 0x0000 to 0x03FF) are added to 0xDC00 to give the second 16-bit code unit or *low surrogate*, which will be in the range 0xDC00..0xDFFF.

Example: U+41

UTF-16: 0x0041

UTF-16

- Code points from U+0000 to U+FFFF, represent as is
- Code points from U+010000 to U+10FFFF
  - 0x010000 is subtracted from the code point, leaving a 20-bit number in the range 0x00000 to 0xFFFFF.
  - The top ten bits (a number in the range 0x0000 to 0x03FF) are added to 0xD800 to give the first 16-bit code unit or *high surrogate*, which will be in the range 0xD800..0xDBFF.
  - The low ten bits (also in the range 0x0000 to 0x03FF) are added to 0xDC00 to give the second 16-bit code unit or *low surrogate*, which will be in the range 0xDC00..0xDFFF.

Example: U+20AC

UTF-16: 0x20AC

UTF-16

- Code points from U+0000 to U+FFFF, represent as is
- Code points from U+010000 to U+10FFFF
  - 0x010000 is subtracted from the code point, leaving a 20-bit number in the range 0x00000 to 0xFFFFF.
  - The top ten bits (a number in the range 0x0000 to 0x03FF) are added to 0xD800 to give the first 16-bit code unit or *high surrogate*, which will be in the range 0xD800..0xDBFF.
  - The low ten bits (also in the range 0x0000 to 0x03FF) are added to 0xDC00 to give the second 16-bit code unit or *low surrogate*, which will be in the range 0xDC00..0xDFFF.

Example: U+245D6

```
  245D6
-10000
------
  145D6  [0001 0100 0101 1101 0110]
```

```
0001010001          0111010110

   051                 1D6
+ D800              + DC00
 -----               -----
  D851                DDD6
```

UTF-16: 0xD851 DDD6

UTF-32

- Fixed length encoding of one 32-bit code unit

Example: U+41

UTF-32: 0x0000 0041

Example: U+20AC

UTF-32: 0x0000 20AC

Example: U+245D6

UTF-32: 0x0002 4546

UTF

- Observations:
  - UTF-32 is fixed encoding of 32-bit → wasteful
  - UTF-8 uses 1 byte only for the first 128 code units. The rest of the BMP characters will take 2 to 3 bytes.
  - UTF-16 uses 2 bytes for all BMP characters → best option

Glyph	Unicode	UTF-8	UTF-16	UTF-32
A	U+0041	0x41	0x0041	0x0000 0041
€	U+20AC	0xE2 82AC	0x20AC	0x0000 20AC
𤭖	U+245D6	0xF0A4 9796	0xD851 DDD6	0x0002 45D6

Try

Unicode	UTF-8	UTF-16	UTF-32
U+1CAFE			

Try

Unicode	UTF-8	UTF-16	UTF-32
U+1CAFE	0xF09C ABBE	0xD832 DEFE	0x0001 CAFE

```
 1CAFE
-10000
------
```

CAFE [1100 1010 1111 1110]

0000110010 1011111110

binary: 0001 1100 1010 1111 1110 (needs 4 bytes)

UTF-8:

11110000 10011100 10101011 10111110

hex: 0xF09C ABBE

```
  032        2FE
+ D800     + DC00
-----      -----
  D832       DEFE
```

UTF-16: 0xD832 DEFE

To recall …

- What have we learned:
  - ✓Describe the process of representing character data using ASCII
  - ✓Describe the process of representing character data using Unicode