

Characters

# How to represent text characters?

- How to associate bit patterns to integers?
  - base 2
  - 2's complement
- How to associate bit patterns to floats?
  - IEEE floating point representation (based on normalized scientific notation)
- How to associate bit patterns to characters?
  - by convention
  - ASCII, UTF

# ASCII: American Standard Code for Information Exchange

- Developed in 60s, based on the English alphabet
- use one byte (**with MSB=0**) to represent each character
- How many unique characters can be represented?

128

# 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]

# Example-1: convert upper to lowercase

```
// tolower returns the corresponding  
// lowercase character for c if c is an  
// uppercase letter. Otherwise, it returns c.  
char tolower(char c) {
```

```
}
```

```
int main() {  
    char c = 'A';  
    c = tolower(c);  
    ...  
}
```

# Example-1: convert upper to lower case

```
// tolower returns the corresponding
// lowercase character for c if c is an
// uppercase letter. Otherwise, it returns c.
char tolower(char c) {

    // test if c is an uppercase letter
    if (c < 'A' || c > 'Z') {
        return c;
    }

}
```

## Example-2: convert digit character to integer

```
// toDigit returns the corresponding integer for c  
// if c is a valid digit character, e.g '1', '2',  
// Otherwise, it returns -1.
```

```
int toDigit(char c) {
```

```
}
```

```
int main() {
```

```
    int d = toDigit('8');
```

```
    printf("d=%d, 2*d=%d\n", d, 2*d);
```

```
}
```

Dec	Hex	ASCII char
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9

# The Modern Standard: UniCode

- ASCII can only represent 128 characters
  - How about Chinese, Korean, all of the worlds languages? Symbols? Emojis?
- Unicode standard represents >135,000 characters

<a href="#">U+1F600</a>		grinning face
<a href="#">U+1F601</a>		beaming face with smiling eyes
<a href="#">U+1F602</a>		face with tears of joy
<a href="#">U+1F923</a>		rolling on the floor laughing
<a href="#">U+1F603</a>		grinning face with big eyes



# UTF-8, UTF-16, UTF-32

- UTF-32 is a fixed length (32-bit) encoding
  - UTF-8 is variable length (1, 2 or 4 byte)
    - UTF-8 one byte character is the same as ASCII
  - UTF-16 is originally fixed 2-bytes, then extended to 4-bytes
- 
- C's `char` is ASCII (1-byte)
  - Java's `char` is UTF-16 (2-byte)

# C Strings

# Strings

- String is represented as an array of chars.
  - Recall array has no associated length/capacity.
- How to determine string length?
  - One solution: explicitly pass an integer for length

```
// tolower_string turns every character in character array s  
// into lower case
```

```
void tolower_string(char *s, int len) {  
    for (int i = 0; i < len; i++) {  
        s[i] = tolower(s[i]);  
    }  
}
```

# Strings

- C's solution to determine string length:
  - Programmers are expected to store a NULL character at the end of the string (by convention)

```
void tolower_string(char *s) {  
    int i = 0;  
    while (s[i] != '\0') {  
        s[i] = tolower(s[i]);  
        i++;  
    }  
}
```

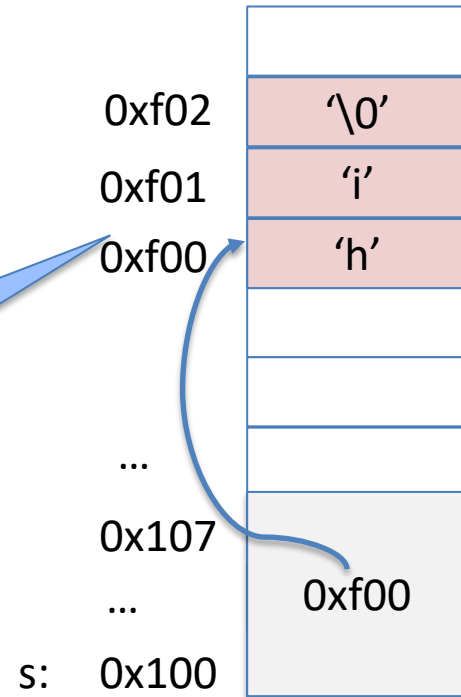
# String initialization

Method-1:

```
char *s = "hi";
```

```
s[0] = 'H';
```

- Character array allocated at program start, deallocated at program exit.
- Character array is read-only.



# String initialization

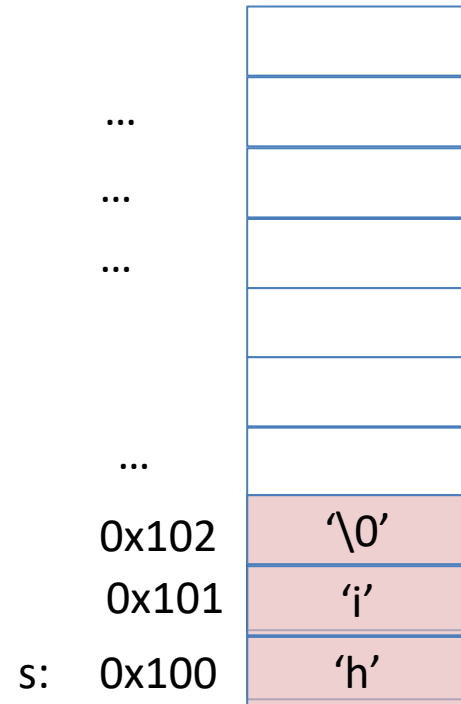
Method-2:

```
char s[3]="hi";
```

same as:

```
char s[3]=['h','i','\0'];
```

```
s[0]='h';
```



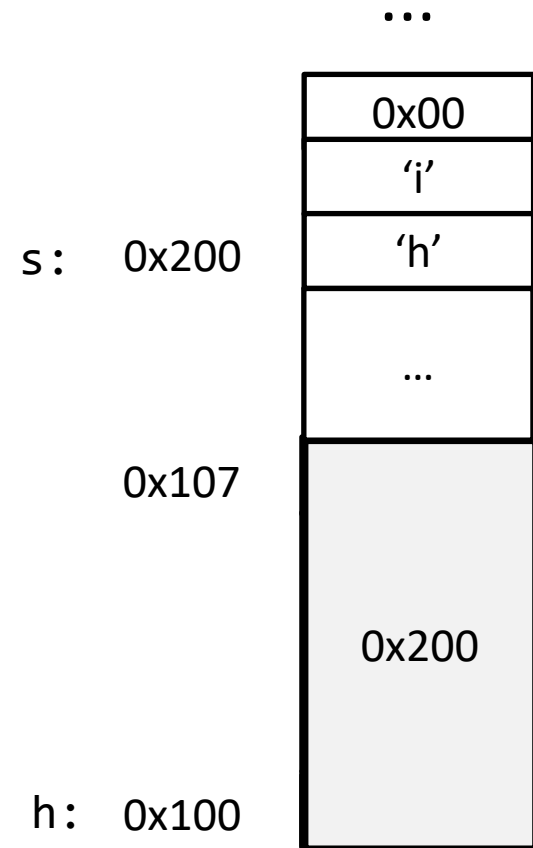
- Character array is a local variable (assuming it's defined within a function).

# Copying string?

does this make a copy of “hi”?

```
char s[3] = "hi";  
char *h;  
h = s;  
h[0] = 'H';
```

```
printf("s=%s h=%s\n", s, h);
```



# Copying string?

does this make a copy of “hi”?

```
char s[3] = “hi”;
```

```
char h[3];
```

```
h = s;
```

```
h[0] = ‘H’;
```

```
printf(“s=%s h=%s\n”, s, h);
```



# Copying string

```
void strcpy(char *dst, char *src)
{

}
```

```
int main()
{
    char s[3] = "hi";
    char h[3];
    strcpy(h, s);
    h[0] = 'H';

    printf("s=%s h=%s\n", s, h);
}
```

# Copying string

```
void strcpy(char *dst, char *src) {  
    int i = 0;  
    while (src[i] != '\0') {  
        dst[i] = src[i];  
        i++;  
    }  
}
```

strcpy is included in C std library.

```
int main() {  
    char s[3] = {'h', 'i', '\0'};  
    char h[3];  
    strcpy(h, s);  
    h[0] = 'H';  
  
    printf("s=%s h=%s\n", s, h);  
}
```

# The atoi function

```
// atoi returns the integer
// corresponding to the string of digits
int atoi(char *s)
{

}

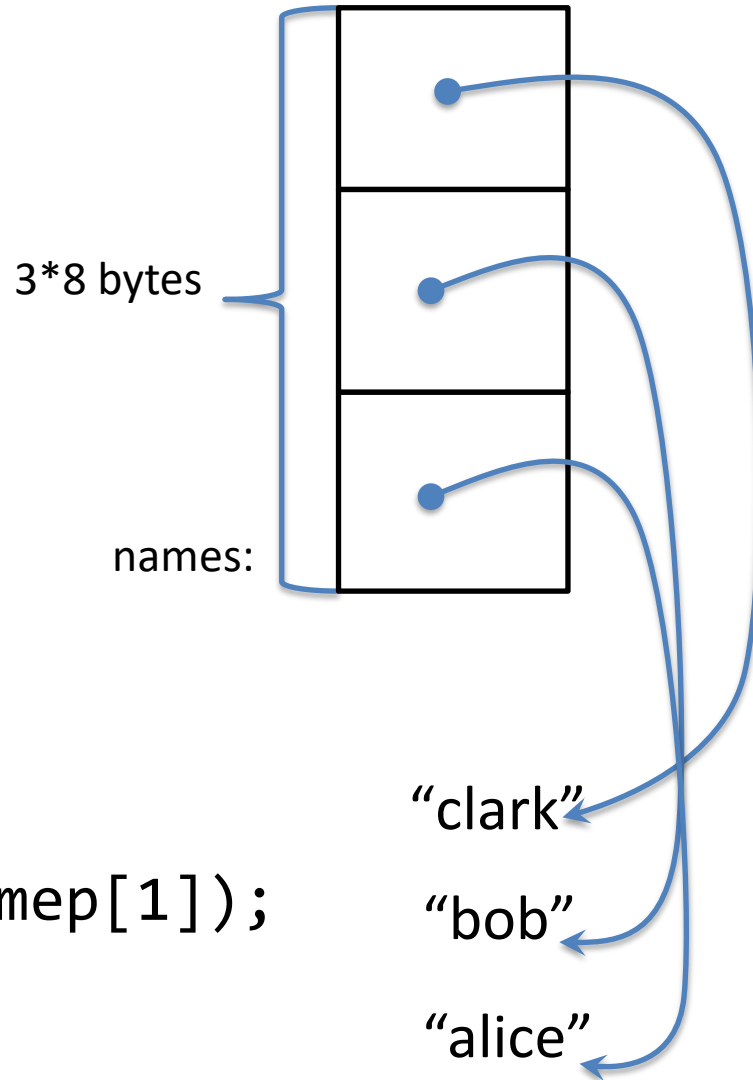
int main()
{
    char *s= "123";
    printf("integer is %d\n", atoi(s));
}
```

# Array of pointers

```
char* names[3] = {  
    "alice",  
    "bob",  
    "clark"  
};
```

```
char **namep;  
namep = names;
```

```
printf("name is %s", namep[1]);
```



# The most commonly used array of pointers: argv

```
int main(int argc, char **argv)
{
    for (int i = 0; i < argc; i++) {
        printf("%s\n", argv[i]);
    }
}
```

```
$ ./a.out 1 2 3
./a.out 1 2 3
```



argv[0] is the name of the executable

# Summary

- Pointers and arrays
  - Pointers are addresses

```
*(p+i) ↔ a[i]
```

- ASCII characters

```
assert('0' != 0)  
assert('\0' == 0)
```

- C string

- an array of characters terminated by `'\0'`
- Programmers are responsible for storing `'\0'` at the end

```
char s[10] = "hello";  
s[1] = '\0';  
printf("s=%s", s);
```