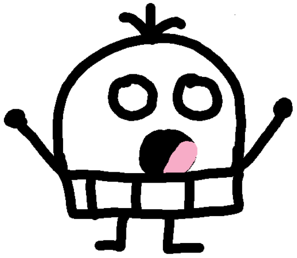
UNSW SYDNEY
Australia's Global University

COMP1911 - Computing 1A

10. Characters and Strings



In this lecture we will cover:

- What is a Character
- ASCII Encoding
- Reading and writing Characters
- What is a String
- Manipulating Strings

The char Type

- The C type char stores small integers.
- It is 8 bits (almost always).
- char guaranteed able to represent integers 0 .. +127.
- char mostly used to store ASCII character codes.
- Don't use char for individual variables, only arrays
- Only use char for characters.
- Even if a numeric variable is only use for the values 0..9, use the type int for the variable.

ASCII Encoding

- ASCII (American Standard Code for Information Interchange)
- Specifies mapping of 128 characters to integers 0..127.
- The characters encoded include:
  - upper and lower case English letters: A-Z and a-z
  - digits: 0-9
  - common punctuation symbols
  - special non-printing characters: e.g. newline and space.
- You don't have to memorize ASCII codes
- Single quotes give you the ASCII code for a character:

```
printf("%d", 'a'); // prints 97
printf("%d", 'A'); // prints 65
printf("%d", '0'); // prints 48
printf("%d", ' ' + '\n'); // prints 42 (32 + 10)
```

- Don't put ASCII codes in your program - use single quotes instead.

Decimal - Binary - Octal - Hex – ASCII Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	“	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	‘	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

ASCII Encoding

- The standard ASCII character set uses just 7 bits for each character.
- The eighth bit became an optional parity bit.
- There larger sets use 8 bits, which gives them 128 additional characters.
- The extra characters are used to represent non-English characters, graphics symbols, and mathematical symbols.

Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex
Ç	128	0x80	á	160	0xa0	└	192	0xc0	α	224	0xe0
ü	129	0x81	í	161	0xa1	┴	193	0xc1	β	225	0xe1
é	130	0x82	ó	162	0xa2	┬	194	0xc2	Γ	226	0xe2
â	131	0x83	ú	163	0xa3	├	195	0xc3	π	227	0xe3
ä	132	0x84	ñ	164	0xa4	─	196	0xc4	Σ	228	0xe4
à	133	0x85	Ñ	165	0xa5	┼	197	0xc5	σ	229	0xe5
å	134	0x86	ª	166	0xa6	╞	198	0xc6	µ	230	0xe6
ç	135	0x87	º	167	0xa7	╟	199	0xc7	τ	231	0xe7
ê	136	0x88	¿	168	0xa8	╚	200	0xc8	Φ	232	0xe8
ë	137	0x89	⌐	169	0xa9	╔	201	0xc9	Θ	233	0xe9
è	138	0x8a	¬	170	0xaa	╩	202	0xca	Ω	234	0xea
ï	139	0x8b	½	171	0xab	╦	203	0xcb	δ	235	0xeb
î	140	0x8c	¼	172	0xac	╠	204	0xcc	∞	236	0xec
ì	141	0x8d	¡	173	0xad	═	205	0xcd	φ	237	0xed
Ä	142	0x8e	«	174	0xae	╬	206	0xce	ε	238	0xee
Å	143	0x8f	»	175	0xaf	╧	207	0xcf	∩	239	0xef
É	144	0x90	░	176	0xb0	╨	208	0xd0	≡	240	0xf0
æ	145	0x91	▒	177	0xb1	╤	209	0xd1	±	241	0xf1
Æ	146	0x92	▓	178	0xb2	╥	210	0xd2	≥	242	0xf2
ô	147	0x93	│	179	0xb3	╙	211	0xd3	≤	243	0xf3
ö	148	0x94	┤	180	0xb4	╘	212	0xd4	⌠	244	0xf4
ò	149	0x95	╡	181	0xb5	╒	213	0xd5	⌡	245	0xf5
û	150	0x96	╢	182	0xb6	╓	214	0xd6	÷	246	0xf6
ù	151	0x97	╖	183	0xb7	╫	215	0xd7	≈	247	0xf7
ÿ	152	0x98	╕	184	0xb8	╪	216	0xd8	°	248	0xf8
Ö	153	0x99	╣	185	0xb9	┘	217	0xd9	·	249	0xf9
Ü	154	0x9a	║	186	0xba	┌	218	0xda	•	250	0xfa
¢	155	0x9b	╗	187	0xbb	█	219	0xdb	√	251	0xfb
£	156	0x9c	╝	188	0xbc	▄	220	0xdc	^{n}	252	0xfc
¥	157	0x9d	╜	189	0xbd	▌	221	0xdd	^{2}	253	0xfd
Pts	158	0x9e	╛	190	0xbe	▐	222	0xde	■	254	0xfe
ƒ	159	0x9f	┐	191	0xbf	▀	223	0xdf		255	0xff

Unicode

- Unicode is a superset of ASCII.
- The numbers 0–127 have the same meaning in ASCII.
- Unicode defines 2^{21} characters.

Manipulating Characters

The ASCII codes for the digits, the upper case letters and lower case letters are contiguous.
This allows some simple programming patterns:

```
// check for lowercase
if (c >= 'a' && c <= 'z') {
    ...
```

```
// check is a digit
if (c >= '0' && c <= '9') {
    // convert ASCII code to corresponding integer
    numeric_value = c - '0';
}
```

How to use one line of code to convert lower case to uppercase?

```
char c;
c = 'a';
```

Total Results: 0

Reading a Character - getchar

C provides library functions for reading and writing characters

- getchar reads a byte from standard input.
- getchar returns an int
- getchar returns a special value (EOF usually -1) if it can not read a byte.
- Otherwise getchar returns an integer (0..255) inclusive.
- If standard input is a terminal or text file this likely be an ASCII code.
- Beware input often buffered until entire line can be read.

```c
int c;
printf("Please enter a character: ");
c = getchar();
printf("The ASCII code is %d\n", c);
```

Output a Character - putchar

C provides library functions for reading and writing characters

- printf is a generic printing function
- putchar print a single character to the screen
- putchar returns the character written on the stdout as an unsigned char. It also returns EOF when some error occurs.
- putchar is much faster

```
c = getchar();
printf("%c", c);
cReturn = putchar(c);
putchar('\n');
printf("%d\n", cReturn);
```

Reading a Character - getchar

Consider the following code:

```c
int c1,c2;

printf("Please enter first character:\n");
c1 = getchar();
printf("Please enter second character:\n");
c2 = getchar();
printf("First %d\nSecond: %d\n", c1, c2);
```

The newline character from pressing Enter will be the second character read.

Reading a Character - getchar

How can we fix the program?

```
int c1, c2;

printf("Please enter first character:\n");
c1 = getchar();
getchar(); // reads and discards a character
printf("Please enter second character:\n");
c2 = getchar();
printf("First: %c\nSecond: %c\n", c1, c2);
```

End of Input

- Input functions such as scanf or getchar can fail because no input is available, e.g., if input is coming from a file and the end of the file is reached.
- On UNIX-like systems (Linux/OSX) typing Ctrl + D signals to the operating system no more input from the terminal.
- Windows has no equivalent - some Windows programs interpret Ctrl + Z similarly.
- getchar returns a special value to indicate there is no input was available.
- This non-ASCII value is #defined as EOF in stdio.h.
- On most systems EOF == -1. Note getchar otherwise returns (0..255) or (0..127) if input is ASCII

Reading Characters to End of Input

Programming pattern for reading characters to the end of input:

```
int ch;

ch = getchar();
while (ch != EOF) {
    printf("'%c' read, ASCII code is %d\n", ch, ch);
    ch = getchar();
}
```

For comparison the programming pattern for reading integers to end of input:

```
int num;
// scanf returns the number of items read
while (scanf("%d", &num) == 1) {
    printf("you entered the number: %d\n", num);
}
```

Useful C Library Functions for Characters

The C library includes some useful functions which operate on characters.
Several of the more useful listed below.

```c
#include <ctype.h>

int toupper(int c); // convert c to upper case
int tolower(int c); // convert c to lower case
int isalpha(int c); // test if c is a letter
int isdigit(int c); // test if c is a digit
int islower(int c); // test if c is lower case letter
int isupper(int c); // test if c is upper case letter
```

Strings

- A string in computer science is a sequence of characters.
- In C strings are an array of **char** containing ASCII codes.
- These array of char have an extra element containing a 0
- The extra 0 can also be written '\0' and may be called a null character or null-terminator.
- This is convenient because programs don't have to track the length of the string.

Strings

Because working with strings is so common, C provides some convenient syntax.
Instead of writing:

```
char hello[]  = {'h', 'e', 'l', 'l', 'o', '\0'};
```

You can write

```
char hello[] = "hello";
```

Note hello will have 6 elements.

fgets - Read a Line

- **fgets(array, array_size, stream)** reads a line of text
    1. **array** - char array in which to store the line
    2. **array_size** - the size of the array
    3. **stream** - where to read the line from, e.g. stdin
- **fgets** will not store more than array size characters in array
- Never use similar C function **gets** which can overflow the array and major source of security exploits
- **fgets** always stores a '\0' terminating character in the array.
- **fgets** stores a '\n' in the array if it reads entire line often need to overwrite this newline character:

```
int i = strlen(line);
if (i > 0 && line[i - 1] == '\n') {
    line[i - 1] = '\0';
}
```

Reading an Entire Input Line

You might use fgets as follows:

```c
#define MAX_LINE_LENGTH 1024
...
char line[MAX_LINE_LENGTH];
printf ("Enter a line: ");
// fgets returns NULL if it can't read any characters
if (fgets(line, MAX_LINE_LENGTH, stdin) != NULL {
        fputs(line, stdout);
        // or
        printf("%s" ,line); // same as fputs
}
```

Reading Lines to End of Input

Programming pattern for reading lines to end of input:

```c
// fgets returns NULL if it can't read any characters

while (fgets(line, MAX_LINE, stdin) != NULL) {
    printf("you entered the line: %s", line);
}
```

string.h

```c
#include <string.h>

// string length (not including '\0')
int strlen(char *s);

// string copy
char *strcpy(char *dest, char *src);
char *strncpy(char *dest, char *src, int n);

// string concatenation/append
char *strcat(char *dest, char *src);
char *strncat(char *dest, char *src, int n);
```

string.h

```c
#include <string.h>

// string compare
int strcmp(char *s1, char *s2);
int strncmp(char *s1, char *s2, int n);
int strcasecmp(char *s1, char *s2);
int strncasecmp(char *s1, char *s2, int n);

// character search
char *strchr(char *s, int c);
char *strrchr(char *s, int c);
```

Arrays of Strings

We can define an array of strings as a 2d array of characters.
As well as manipulating each individual cell (characters), we can sometimes manipulate whole arrays(strings).

```c
char animals[3][20] = {"cat", "dog", "bird"};

// using 1 index gives us a whole array
// (in this case a whole string)
// using 2 indexes gives one element in the 2d array
// (in this case a character);
printf("%s %c \n", animals[1], animals[2][1]);
```

What is the output?

```
char animals[3][20] = {"cat",
"dog", "bird"};
printf("%s %c", animals[1],
animals[2][1]);
```

Total Results: 0

Arrays of Strings

We can define an array of strings as a 2d array of characters. As well as manipulating each individual cell (characters), we can sometimes manipulate whole arrays(strings).

```
char animals[3][20] = {"cat", "dog", "bird"};

// using 1 index gives us a whole array
// (in this case a whole string)
// using 2 indexes gives one element in the 2d array
// (in this case a character);
printf("%s %c \n", animals[1], animals[2][1]);
```

Note this will print dog i

Command-line Arguments

Command-line arguments are 0 or more strings specified when program is run.

For example, if you run this command in a terminal:

```
dcc count.c -o count
```

dcc will be given 3 command-line arguments: **"count.c" "-o“ "count"**

main needs a different prototype if you want to access command-line arguments

```
int main(int argc, char *argv[]) { ...
```

Accessing Command-line Arguments

argc stores the number of command-line arguments + 1

argc == 1 if no command-line arguments

argv stores program name + command-line arguments

argv[0] always contains the program name

argv[1] argv[2] ... command-line arguments if supplied

```
#include <stdio.h>
int main(int argc, char *argv[]) {
    int i = 1;
    printf("My name is %s\n", argv[0]);
    while (i < argc) {
        printf("Argument %d is: %s\n", i, argv[i]);
        i = i + 1;
    }
}
```

Converting Command-line Arguments

stdlib.h defines useful functions to convert strings.

atoi converts string to int

atof converts string to double

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
    int i, sum = 0;
    i = 1;
    while (i < argc) {
        sum = sum + atoi(argv[i]);
        i = i + 1;
    }
    printf("sum of command-line arguments=%d\n", sum);
}
```

Array Representation

A C array has a very simple underlying representation, it is stored in a contiguous (unbroken) memory block and a pointer is kept to the beginning of the block.

```
char s[] = "Hi!";
printf("s:\t%p\t*s:\t%c\n\n", s, *s);
printf("&s[0]:\t%p\ts[0]:\t%c\n", &s[0], s[0]);
printf("&s[1]:\t%p\ts[1]:\t%c\n", &s[1], s[1]);
printf("&s[2]:\t%p\ts[2]:\t%c\n", &s[2], s[2]);
printf("&s[3]:\t%p\ts[3]:\t%c\n", &s[3], s[3]);
```

Array variables act as pointers to the beginning of the arrays!

Questions