16.070 Introduction to Computers and Programming

March 14 **Recitation 6** **Spring 2002**

Topics:

- Quick review of PS4 issues
- Fundamental Data types
- ASCII / Arithmetic Conversion
- Number Systems / Logical Operation
- Data Representation
- Sampling

Common Programming Mistakes

At office hours and while grading a number of mistakes have surfaced repeatedly.
Here are a few:

1. Pointer variables should be initialized to the constant NULL.

2. The address a pointer points to is incremented by pointer arithmetic according to the size of the variable type that the pointer is declared as.

3. Constants (#define and const) should be all uppercase.

3. Function prototypes specified in homework should be used without modification.

Fundamental Data Types

The fact is that data types are platform specific. IBM, SGI, Sun and other manufacturers all adhere to their own standards regarding the amount of memory assigned for different data types. ANSI-C does provide a minimum standard, indicating **at least** how many bytes should be assigned to each standard data type.

Variable Type	Keyword	Bytes Required	Range
Character	`char`	1	-2^{7} to $2^{7}-1$
Integer	`int`	2	-2^{15} to $2^{15}-1$
Short Integer	`short`	2	-2^{15} to $2^{15}-1$
Long Integer	`long`	4	-2^{31} to $2^{31}-1$
Unsigned Character	`unsigned char`	1	0 to $2^{8}-1$
Unsigned Integer	`unsigned int`	2	0 to $2^{16}-1$
Unsigned Short Integer	`unsigned short`	2	0 to $2^{16}-1$
Unsigned Long Integer	`unsigned long`	4	0 to $2^{32}-1$
Single-Precision Floating-Point*	`float`	4	-10^{38} to 10^{38}
Double-Precision Floating-Point**	`double`	8	-10^{308} to 10^{308}

* Approximate precision to 7 digits
** Approximate precision to 19 digits

Table 1 ANSI-C minimum memory requirements for standard data types

How do we know how much memory is allotted for each data type on your specific platform? Programming languages have a *sizeof()*, or equivalent, statement. It is always good style to use *sizeof()* in C when you rely on the amount of memory used by your program. This statement returns the size of any data type e.g.

```
int a = 0;
a=sizeof(int);
printf("integers have size: %d bytes",a);

Output:
integers have size: 4 bytes
Press any key to continue
```

You will notice that integers on the PC platform, running MS Windows, consume more memory than the minimum ANSI specification of 2 bytes.

ASCII Character Table

You may have noticed that the above table specifies a variable of type *char* to have a range from 0 to 255. How can a character correspond to a number?
The **ASCII** ("American Standard Code for Information Interchange") codes are used to represent characters as one byte integers. The first 128 of them (0 → 127) are the standard ASCII characters, while the next 128 (128 → 255) are the extended ASCII characters (symbols, accented letters, Greek letters, etc...).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Table 2 Hex-indexed ASCII table

Some examples...

Character	Decimal	Hex	Octal	Binary
space	32	20	40	0010 0000
!	33	21	41	0010 0001
"	34	22	42	0010 0010
#	35	23	43	0010 0011
$	36	24	44	0010 0100
A	65	41	101	0100 0001
B	66	42	102	0100 0010

Control Codes

NUL (null) SOH (start of heading) STX (start of text) ETX (end of text) EOT (end of transmission) - Not the same as ETB ENQ (enquiry) ACK (acknowledge) BEL (bell) - Caused teletype machines to ring a bell. Causes a beep in many common terminals and terminal emulation programs. BS (backspace) - Moves the cursor (or print head) move backwards (left) one space. TAB (horizontal tab) - Moves the cursor (or print head) right to the next tab stop. The spacing of tab stops is dependent on the output device, but is often either 8 or 10. LF (NL line feed, new line) - Moves the cursor (or print head) to a new line. On Unix systems, moves to a new line AND all the way to the left. VT (vertical tab) FF (form feed) - Advances paper to the top of the next page (if the output device is a printer).	CR (carriage return) - Moves the cursor all the way to the left, but does not advance to the next line. SO (shift out) - Switches output device to alternate character set. SI (shift in) - Switches output device back to default character set. DLE (data link escape) DC1 (device control 1) DC2 (device control 2) DC3 (device control 3) DC4 (device control 4) NAK (negative acknowledge) SYN (synchronous idle) ETB (end of transmission block) - Not the same as EOT CAN (cancel) EM (end of medium) SUB (substitute) ESC (escape) FS (file separator) GS (group separator) RS (record separator) US (unit separator)

Type Casting

Sometimes we don't like the standard rules of arithmetic conversion to be applied to us. In such cases we may decide to *cast* variables or results of arithmetic into specific types. Lets look at an example:

```
double Velocity, Time_Elapsed;
Distance = Velocity*Time_Elapsed;
```

We would usually need to define *Distance* as being of data type *double*. A compiler like Visual C would even compile the code if *Distance* was defined as *int*, but it would issue a warning message. This is not necessarily true for all compilers! Other C compilers, such as Interactive C, which we will use when programming the Handy Boards, are more strict about types. Many compilers don't even hold with the arithmetic conversion rules that you have learned. The safest bet is to make use of a process called *type casting*. The correct way to cast a variable of one data type into another would be:

```
/* variable declaration */
int Distance = 0;
double Velocity = 10.0;
double Time_Elapsed = 100.0;

/* Type casting */
Distance = (int)(Velocity*Time_Elapsed);
```