LECTURE 6: NUMBER SYSTEMS, OCTAL/HEX DUMPS, VISITYPE.C

NUMBER SYSTEMS

We are accustomed to the base-10 number system (decimal).
Computers use the base-2 number system (binary).

problem	binary numbers are not easy for people to read conversion between base-2 and base-10 is not easy
solution	convert binary numbers to: octal (base-8) or hexadecimal (base-16)
	these conversions are easier because 8 is 2^3 and 16 is 2^4. This means that:
	Three binary digits can represent an octal digit: — 000 through 111 / 0 through 7
	Four binary digits can represent a hex digit: — 0000 through 1111 / 0 through 15

Example:		
It's not easy to look at the binary number and get 1708...		
decimal	1708	
binary	011010101100	
...but you can go to octal or hex with a few quick steps:		
octal	011010101100	start with binary
	011 010 101 100	group in sets of 3
	03 02 05 04	convert sets
	03254	
hex	011010101100	start with binary
	0110 1010 1100	group in sets of 4
	0x6 0xa 0xc	convert sets
	0x6ac	

For computers to process our letters, digits, punctuation, etc., we need a binary code for each such "character."

ASCII	American Standard Code for Information Interchange
	An encoding that provides these codes.

Standard 8-bit bytes and 16-bit words are not integer multiples of 3 bits, but are multiples of 4 bits, favoring use of hex.

HEX DUMP

od	octal dump utility
	man od to view od's options
	x option dumps in hex

hex dump of hello.c

```
$ od -tx1 -cw8 hello.c
0000000  23  69  6e  63  6c  75  64  65
          #   i   n   c   l   u   d   e
0000010  3c  73  74  64  69  6f  2e  68
          <   s   t   d   i   o   .   h
0000020  3e  0a  0a  69  6e  74  20  6d
          >  \n  \n   i   n   t       m
0000030  61  69  6e  20  28  76  6f  69
          a   i   n       (   v   o   i
0000040  64  29  20  7b  0a  20  20  70
          d   )       {  \n              p
0000050  72  69  6e  74  66  28  22  48
          r   i   n   t   f   (   "   H
0000060  65  6c  6c  6f  2c  20  77  6f
          e   l   l   o   ,       w   o
0000070  72  6c  64  21  5c  6e  22  29
          r   l   d   !   \   n   "   )
0000100  3b  0a  20  20  72  65  74  75
          ;  \n              r   e   t   u
0000110  72  6e  20  30  3b  0a  7d  0a
          r   n       0   ;  \n   }  \n
0000120  0a
         \n
0000121
```

The first column is the offset from the beginning of the file

VISITYPE.C

Recall:	a string is an array of chars.	
	a string literal ends in \0.	
	a character is a small integer.	
For this program we want to convert an input character value to a printable ASCII string and send that to output.		
We need:	a table of 4-byte character strings	
	why 4 bytes?	largest control character abbreviation is 3 characters
		we need to end with \0
We can use an array for this.		

ARRAY INITIALIZATION

The general form for declaring an array is:
`type arrayName[arraySize];`

```
int numArray[5]; /* declares an array of 5 ints */
```

We can also declare and initialize at the same time:

```
int numArray[5] = {0, 1, 2, 3, 4};
```

and if we leave out arraySize, an array just large enough to hold the initialization is created:

```
int numArray[] = {0, 1, 2 , 3, 4}; /* same as above */
```

We can create strings (arrays of chars) using the double quote notation:

```
char str[] = "hello"; /* array of 6 chars: h e l l o \0 */
```

ASCII CODE CONVERSION ARRAY

We'll create an array containing the ASCII characters in the order of their character value. In other words, since the null control character is the 0th character in the ASCII table, it will be the first element in our array.

```
char asciiname[] = "NUL\0"  "SOH\0"  "STX\0"  "ETX\0"
  [all the other characters represented by ASCII codes]
  "□□|\0"  "□□}\0"  "□□~\0"  "DEL\0";
```

There are 128 ASCII codes.	We will have 128 4-byte character strings.
Why 4 bytes?	Each char uses 1 byte (8 bits).
	The longest control character abbreviations (e.g., ESC) are 3 characters long.
	We need \0 to make a string.

This will create a long string of chars:
"NUL\0SOH\0STX\0ETX\0 [...] □□|\0□□}\0□□~\0DEL\0"

string	"NUL\0SOH\0STX\0 [...] }\0 ~\0DEL\0"
char index	012 34567 89... (recall that \0 is a single character)

This array will have 4 * 128 = 512 char elements.

CHARACTER CONSTANTS

C character constants (K&R, page 193)

newline	NL(LF)	\n		backslash	\	\\
horiz. tab	HT	\t		question mark	?	\?
vert. tab	VT	\v		single quote	'	\'
backspace	BS	\b		double quote	"	\"
carriage ret.	CR	\r		octal number	ooo	\ooo
formfeed	FF	\f		hex number	hh	\xhh
audible alert	BEL	\a				

Some characters can't be placed in the quoted initialization string directly, we need to use the escape character.

"□□"\0"	Why won't these work?
"□□\\0"	

ACCESSING THE STRINGS IN asciiname[]

The printf conversion specifier for a string (%s) requires a pointer to a string.

We have created an array of 512 chars, where each set of 4 elements is a string.

If we have a string index i with value 0-127:	Array index 4*i indicates the start of each 4-char string.

We use the address operator (&) to get the pointer:

```
&asciiname[4 * i]
```