MC140: lecture #4

today's topic:

data storage
binary arithmetic

reading a value.

if code contains:

```
int x;
printf( "please enter a number: " );
scanf( "%d",&x );
```

and run looks like this:

```
please enter a number: 97
```

then memory will look like this:

the scanf function.

- **scanf**
  - input statement = "read formatted data"
  - stdio library function
  - format:

    scanf("%d",&x);
  - "%d" = input is an integer
  - &x = store input in memory location named x

storage.

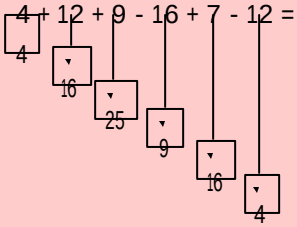
- what does

  "memory location named x"

  mean?
- the computer *stores* data so it can be used later
- stores = remembers
- for example:
  - STORE / STO command on a calculator

storage, 2:
calculator example.

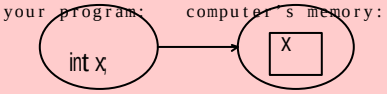
4 + 12 + 9 - 16 + 7 - 12 = ?



storage, 3:
variables.

- think of the computer's memory as a bunch of boxes
- inside each box, there is a number
- you give each box a name

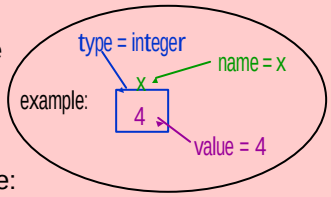
⇒ defining a *variable*



variables, 1.

- variables have
  - type
  - name
  - value
- first data type:
  - int = integer

  ⇒ *whole numbers, including 0 and negative numbers*



variables, 2: naming.

- naming rules in C:
  - names may contain letters and/or numbers
  - names may also contain "_"
  - length between 1 and 245 (max for LCC)
  - cannot use C keywords
  - cannot begin with a number
  - *C is case-sensitive!!!*

variables, 3: data types.

- int = integer
- float = real numbers
- double = really big/precise real numbers
- char = character

data storage.

- data is stored in *bytes* = words
- 1 byte = 8 bits
- 1 bit = 1 switch/relay
  - ON or OFF
  - OPEN or CLOSED
  - value = 1 or 0

data representation.

- 1 byte = 8 bits: 10010100
- because each bit can have one of two values, numbers are stored using the *binary* or *base 2* number system

what is a *base*?

- we normally use base 10
  - right-most digit is 1's digit
  - then (moving left) is 10's digit
  - then 100's digit
  - etc.
- example: 362
  - 2 1's
  - 6 10's
  - 3 100's

  ⇒ (2 * 1) + (6 * 10) + (3 * 100)

base 10.

- $1 = 10^0$
  $10 = 10^1$
  $100 = 10^2$
  etc.
- so:
  $(2 * 1) + (6 * 10) + (3 * 100)$
  $= (2 * 10^0) + (6 * 10^1) + (3 * 10^2)$
- valid digits: 0 .. 9

base 2.

- $1 = 2^0$
  $10 = 2^1$
  $100 = 2^2$
  etc.
- valid digits: 0 .. 1
- examples:
  10010100
  0001
  1111

base conversion: base 2 → base 10.

- what are these numbers in base 10?

(1) start from right-most digit
- that is 2^0-th place
- next is 2^1-th place
- next is 2^2-th place
- etc.

(2) multiply value of digit by place value

(3) add them all up

base conversion, 2 → 10: example.

$10010100/_2$ =
$(0*2^0)+$
$(0*2^1)+$
$(1*2^2)+$
$(0*2^3)+$
$(1*2^4)+$
$(0*2^5)+$
$(0*2^6)+$
$(1*2^7)$

$10010100/_2$ =
(0)+
(0)+
(4)+
(0)+
(16)+
(0)+
(0)+
(128) = $148/_{10}$

base conversion: base 10 → base 2.

- how do we get base 2?

(1) divide base 10 number by 2, repeatedly, until there is nothing left

(2) remember all the remainders

(3) list the remainders backwards and you've got the base 2 equivalent

base conversion, 10 → 2: example.

$148/_{10}$ =
148 / 2 = 74 rem 0
74 / 2 = 37 rem 0
37 / 2 = 18 rem 1
18 / 2 = 9 rem 0
9 / 2 = 4 rem 1
4 / 2 = 2 rem 0
2 / 2 = 1 rem 0
1 / 2 = 0 rem 1

= $10010100/_2$

back to storage…

- int = integer
  - one byte (aka short) = 8 bits
  - two bytes = 16 bits
- float = real number (aka floating point)
  - four bytes = 32 bits
- double = real number, larger, more precision
  - eight bytes = 64 bits
- char = character
  - one byte = 8 bits

so… a few questions:

- short or 1-byte or 8-bit integers can have a maximum value of…?
- int or 2-byte or 16-bit integers can have a maximum value of…?
- how is the maximum represented?

some things to KNOW.

- base 10 = decimal
- base 8 = octal (0..7)
  - 3 bits
- base 16 = hexadecimal (0..F), aka hex
  - 4 bits
- know how to convert binary ↔ hex
  - 4-bit trick…

more questions…

- how are floating point numbers stored? doubles?
- i.e., how are fractions stored?
- how are negative numbers stored?
- how are characters stored?
- today: characters
- future: answers to other questions ☺

characters.

- 1 byte = 8 bits
- numeric values from 0 to 127
- used as indexes into a table

the ASCII table

- ASCII = American Standard Code for Information Interchange
- contains numbers, letters (upper and lower case), escape sequences, punctuation marks…

ASCII.

Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
000	0	00	NUL '\0'	100	64	40	@
001	1	01	SOH	101	65	41	A
002	2	02	STX	102	66	42	B
003	3	03	ETX	103	67	43	C
004	4	04	EOT	104	68	44	D
005	5	05	ENQ	105	69	45	E
006	6	06	ACK	106	70	46	F
007	7	07	BEL '\a'	107	71	47	G
010	8	08	BS '\b'	110	72	48	H
011	9	09	HT '\t'	111	73	49	I
012	10	0A	LF '\n'	112	74	4A	J
013	11	0B	VT '\v'	113	75	4B	K
014	12	0C	FF '\f'	114	76	4C	L
015	13	0D	CR '\r'	115	77	4D	M
016	14	0E	SO	116	78	4E	N
017	15	0F	SI	117	79	4F	O
020	16	10	DLE	120	80	50	P
021	17	11	DC1	121	81	51	Q
022	18	12	DC2	122	82	52	R
023	19	13	DC3	123	83	53	S
024	20	14	DC4	124	84	54	T
025	21	15	NAK	125	85	55	U
026	22	16	SYN	126	86	56	V
027	23	17	ETB	127	87	57	W
030	24	18	CAN	130	88	58	X
031	25	19	EM	131	89	59	Y
032	26	1A	SUB	132	90	5A	Z
033	27	1B	ESC	133	91	5B	[
034	28	1C	FS	134	92	5C	\ '\\'
035	29	1D	GS	135	93	5D	]
036	30	1E	RS	136	94	5E	^

Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
037	31	1F	US	137	95	5F	_
040	32	20	SPACE	140	96	60	`
041	33	21	!	141	97	61	a
042	34	22	"	142	98	62	b
043	35	23	#	143	99	63	c
044	36	24	$	144	100	64	d
045	37	25	%	145	101	65	e
046	38	26	&	146	102	66	f
047	39	27	'	147	103	67	g
050	40	28	(	150	104	68	h
051	41	29	)	151	105	69	i
052	42	2A	*	152	106	6A	j
053	43	2B	+	153	107	6B	k
054	44	2C	,	154	108	6C	l
055	45	2D	-	155	109	6D	m
056	46	2E	.	156	110	6E	n
057	47	2F	/	157	111	6F	o
060	48	30	0	160	112	70	p
061	49	31	1	161	113	71	q
062	50	32	2	162	114	72	r
063	51	33	3	163	115	73	s
064	52	34	4	164	116	74	t
065	53	35	5	165	117	75	u
066	54	36	6	166	118	76	v
067	55	37	7	167	119	77	w
070	56	38	8	170	120	78	x
071	57	39	9	171	121	79	y
072	58	3A	:	172	122	7A	z
073	59	3B	;	173	123	7B	{
074	60	3C	<	174	124	7C	|
075	61	3D	=	175	125	7D	}
076	62	3E	>	176	126	7E	~
077	63	3F	?	177	127	7F	DEL

reading.

- material covered today:
  - DD: ch 2.3 - 2.4