cis1.5 spring2009 lecture I.3

today we are going to talk about **remembering information**

- storing data
- variables
- intrinsic data types
- assigning values
- storing numbers
- bases (2, 8, 10, 16)
- storing characters
- doing math
- outputting variables

storing data

- think of the computer's memory as a bunch of boxes
- inside each box, there is a number
- you give each box a name
  $\Rightarrow$ defining a *variable*
- example:

program code:

```
int x;
```

computer's memory:

x $\rightarrow \boxed{}$

variables

- variables have:
  - name
  - type
  - value
- naming rules:
  - names may contain letters and/or numbers
  - but cannot begin with a number
  - names may also contain underscore (_)
  - can be of any length
  - cannot use C++ keywords (also called *identifiers*)
  - C++ is *case-sensitive!!*

intrinsic data types

type	size	minimim value	maximum value
bool	1 bit	0	1
byte	8 bits	$-128 = -2^7$	$127 = 2^7 - 1$
char	8 bits	$-128 = -2^7$	$127 = 2^7 - 1$
short	16 bits	$-32,768 = -2^{15}$	$32,767 = -2^{15} - 1$
int	32 (or 16) bits	-2^{31} (2^{15})	$2^{31} - 1$ $(2^{15} - 1)$
long	32 bits	-2^{31}	$2^{31} - 1$
float	32 bits	$\approx -3.4E + 38$, 7 sig. dig.	$\approx 3.4E + 38$, 7 sig. dig.
double	64 bits	$\approx -1.7E + 308$, 15 sig. dig.	$\approx 1.7E + 308$, 15 sig. dig.

"sig. dig." = significant digits

assigning values

- = is the assignment operator
- example:

program code:

```
int x; // declaration
x = 19; // assignment
```

or

```
int x = 19;
```

computer's memory:

x $\rightarrow$ $\boxed{19}$

storing numbers

x $\rightarrow$ $\boxed{19}$

is really stored like this:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1

this is binary, or base 2!

$19_{10} = 10011_2$

remember bases?

base 10:

$$\begin{aligned} 362 &= (2 * 1) + (6 * 10) + (3 * 100) \\ &= (2 * 10^0) + (6 * 10^1) + (3 * 10^2) \end{aligned}$$

base 2:

$$\begin{aligned} 1 &= 2^0 &= 1 \\ 10 &= 2^1 &= 2 \\ 100 &= 2^2 &= 4 \\ 1000 &= 2^3 &= 8 \\ 10000 &= 2^4 &= 16 \\ \ldots \end{aligned}$$

so

$$\begin{aligned} 10011_2 &= (1 * 2^0) + (1 * 2^1) + (0 * 2^2) + (0 * 2^3) + (1 * 2^4) \\ &= (1 * 1) + (1 * 2) + (0 * 4) + (0 * 8) + (1 * 16) \\ &= 19_{10} \end{aligned}$$

base conversion: 2 to 10

$$1010100_2 =$$

$$\begin{array}{r|r|r} & = & = \\ (0 * 2^0) & (0 * 1) & 0 \\ +\ (0 * 2^1) & +\ (0 * 2) & +\ 0 \\ +\ (1 * 2^2) & +\ (1 * 4) & +\ 4 \\ +\ (0 * 2^3) & +\ (0 * 8) & +\ 0 \\ +\ (1 * 2^4) & +\ (1 * 16) & +\ 16 \\ +\ (0 * 2^5) & +\ (0 * 32) & +\ 0 \\ +\ (1 * 2^6) & +\ (1 * 64) & +\ 64 \end{array}$$

$$= 84_{10}$$

base conversion: 10 to 2

$84_{10} =$

84 / 2	=	42	rem 0
42 / 2	=	21	rem 0
21 / 2	=	10	rem 1
10 / 2	=	5	rem 0
5 / 2	=	2	rem 1
2 / 2	=	1	rem 0
1 / 2	=	0	rem 1

$\Rightarrow 1010100_2$

two tricks

base 8 (octal):

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

base 16 (hexadecimal, "hex"):

0000	0	1000	8
0001	1	1001	9
0010	2	1010	A (10)
0011	3	1011	B (11)
0100	4	1100	C (12)
0101	5	1101	D (13)
0110	6	1110	E (14)
0111	7	1111	F (15)

- replace each octal (or hex) digit with the 3 (or 4) digit binary
- replace every 3 (or 4) binary digits with one octal (or hex) digit

back to storage

x → $\boxed{19}$

is really stored like this:

31	30	...	7	6	5	4	3	2	1	0
0	0	...	0	0	0	1	0	0	1	1

- bits are numbered, from right to left, starting with 0
- highest (rightmost, "most significant") bit is *sign* bit

storing characters: ASCII

- ASCII = American Standard Code for Information Interchange
- characters are stored as numbers
- standard table defines 128 characters
- for example, when you define:
  `char c = 'A';` the data is stored as a number:
  'A' $= 65_{10} = 01000001_2$
  like this:
  c →

7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	1

- sometimes it is handy to *convert* between integers and characters explicitly
- for example:

```
char c = 'A';
int i;
i = (int)c;
```

in which case, the value of `i` will be 65.

doing math

$+$	unary plus
$-$	unary minus
$+$	addition
$-$	subtraction
$*$	multiplication
$/$	division
$\%$	modulo

example:

```
int x, y;
x = -5;
y = x * 7;
y = y + 3;
x = x * -2;
y = x / 19;
```

what are x and y equal to?

modulo means "remainder after integer division"

increment and decrement operators

- increment operator: $++$
  meaning: add one and assign
  example: `i++;`
  is the same as: `i = i + 1;`
- decrement operator: $--$
  meaning: subtract one and assign
  example: `i--;`
  is the same as: `i = i - 1;`

assignment operators

operator	*meaning*	*example*
+=	add and assign	`i += 3;` is the same as: `i = i + 3;`
-=	subtract and assign	`i -= 3;` is the same as: `i = i - 3;`
*=	multiply and assign	`i *= 3;` is the same as: `i = i * 3;`
/=	divide and assign	`i /= 3;` is the same as: `i = i / 3;`
%=	modulo and assign	`i %= 3;` is the same as: `i = i % 3;`

outputting variables

- you can output the value of a variable using `cout`
- for example:

```
int i;
i = 7;
cout << "the value of i is " << i << endl;
```

- notice the use of the `endl` function. this is a built-in C++ function that produces a newline. it is the same as `"\n"`. so we could have written the following, which would do the same thing as the last line, above:

```
cout << "the value of i is " << i << "\n";
```

- you can output character variables in the same way:

```
char c;
c = 'E';
cout << "the value of c is " << c << endl;
```