

VARIABLES AND STORAGE

Today

- Variables
- Data types
- Data storage
- Binary numbers
- ASCII

Some of this recaps what we did last lecture, most of it is new.

Data types

- Programs = objects + methods
- Objects = data
 - There is more to be said here, but we won't say it until CIS 15.
- Data must be *stored*.
- All storage is numeric (0's and 1's)

Data storage

- 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*
- Example:

Program code:

```
int x;
```

Computer's memory:

x →

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	Minimum 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	$\approx -3.4E + 38, 7 \text{ sig. dig.}$	$\approx 3.4E + 38, 7 \text{ sig. dig.}$
double	64 bits	$\approx -1.7E + 308, 15 \text{ sig. dig.}$	$\approx 1.7E + 308, 15 \text{ sig. dig.}$

“sig. dig.” = significant digits

Assignment

- = is the assignment operator
- Example:

Program code:

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

or

```
int x = 19;
```

Computer's memory:

x → 19

Storage is binary

$x \rightarrow$ 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	0	0	0	0	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

this is 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:

$$1 = 2^0 = 1$$

$$10 = 2^1 = 2$$

$$100 = 2^2 = 4$$

$$1000 = 2^3 = 8$$

$$10000 = 2^4 = 16$$

...

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.

$$1010100_2 =$$

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

$$= 84_{10}$$

Base conversion: 10 to 2.

$$84_{10} =$$

$$\begin{array}{r|l} 84 / 2 & = 42 \text{ rem } 0 \\ 42 / 2 & = 21 \text{ rem } 0 \\ 21 / 2 & = 10 \text{ rem } 1 \\ 10 / 2 & = 5 \text{ rem } 0 \\ 5 / 2 & = 2 \text{ rem } 1 \\ 2 / 2 & = 1 \text{ rem } 0 \\ 1 / 2 & = 0 \text{ rem } 1 \end{array}$$

Read the remainders from the bottom up, and we get the binary number we want:

$$84_{10} = 1010100_2$$

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 digit with 3 binary digits
- Replace every 3 binary digits with one octal digit

- Thus 37_8 is 011111 in binary.
- And 101011 is 53_8
- Replace each hex digit with 4 binary digits.
- Replace every 4 binary digits with one hex digit
- Thus 37_{16} is 00110111 in binary.
- And 11110110 is $F6_{16}$

Back to storage

$x \rightarrow$ 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

ASCII.

- ASCII = American Standard Code for Information Interchange
- Characters are stored as numbers
- Standard table defines 128 characters
- Example:

```
char c = 'A' ;
```

'A' = $65_{10} = 01000001_2$

c →

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

Mathematical operators.

+	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

- We are always increasing and decreasing values by one, so there are shortcuts.

- Increment: ++

`i++;`

is the same as:

`i = i + 1;`

- Decrement: --

`i--;`

is the same as:

`i = i - 1;`

Assignment operators.

- There are shorthand ways of doing other combinations of arithmetic and assignment.

+=

`i += 3;` is the same as: `i = i + 3;`

-=

`i -= 3;` is the same as: `i = i - 3;`

***=**

`i *= 3;` is the same as: `i = i * 3;`

- Also:

`/=`

`i /= 3;` is the same as: `i = i / 3;`

`%=`

`i %= 3;` is the same as: `i = i % 3;`

Lab Tie-in

- In case you don't know what a Roomba is, here is one:



- It is a simple robot programmed for floor cleaning.
- We have some in our research lab that we program for other things as well.

Summary

- This lecture expanded on the idea of a variable.
- We considered how variables are represented in computer memory.
- We mentioned binary, octal and hexadecimal.
- With the idea of a variable under our belts, we recapped arithmetic and assignment.