OUTPUT, VARIABLES AND ASSIGNMENT

Today

- Our first C++ program
- Output
- The software development cycle
- Variables
- Data types
- Data storage
- Binary numbers
- ASCII
- Assignment and mathematical operators

Our first C++ program

"hello world"

- Typical first program in any language
- Output only (no input)

The application source code

file name = hello.cpp

```
/*-------------------------------------------------------
   hello.cpp, 30jan07/parsons

   This class demonstrates output from a C++ application.
-------------------------------------------------------*/
#include <iostream>
using namespace std;

int main()
{
  cout << "this is my c++ world\n";
  cout << "hello from inside of it!\n";
}
```

Output

- *Methods*

  `cout`

- *Arguments*
  - Also called *parameters*
  - Those things that follow `cout`
  - << followed by a *string*, i.e., text in double quotes (")
  - Escape sequences: \n, \t
- Example

```
cout << "Are macs better than PCs?\n";
```

Things to notice

- C++ is CASE sensitive
- Punctuation is really important!
- *Whitespace* doesn't matter for compilation
- *BUT* whitespace DOES matter for readability and your grade!
- File name is same as class name

Let's try it: the software development cycle

1. Open up a *text editor* or an *IDE*
2. Type in the *source code* and save it as a *text file*
3. *Compile* the source code,
   using the *g++* command or a menu option on the IDE
4. *Execute* the program, from the command line or from within the IDE

Data types

- Programs = objects + methods
- Objects = data
- Data must be *stored*
- All storage is numeric (0's and 1's)

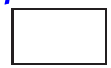
Data storage

- Think of the computer's memory as a bunch of boxes
- Inside each box, there is a number
- You give each box a name
  $\Rightarrow$ defining a *variable*
- Example:

Program code:

```
int x;
```

Computer's memory:

x $\rightarrow$ 

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

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	1	0	0	1	1

this is base 2!

$19_{10} = 10011_2$

Remember bases?

Base 10:

$362 = (2 * 1) + (6 * 10) + (3 * 100)$

$= (2 * 10^0) + (6 * 10^1) + (3 * 10^2)$

Base 2:

$1 = 2^0 = 1$

$10 = 2^1 = 2$

$100 = 2^2 = 4$

$1000 = 2^3 = 8$

$10000 = 2^4 = 16$

...

so

$10011_2 = (1 * 2^0) + (1 * 2^1) + (0 * 2^2) + (0 * 2^3) + (1 * 2^4)$

$= (1 * 1) + (1 * 2) + (0 * 4) + (0 * 8) + (1 * 16)$

$= 19_{10}$

Base conversion: 2 to 10.

$1010100_2 =$

	=		=	
$(0 * 2^0)$		$(0 * 1)$		0
$+ (0 * 2^1)$		$+ (0 * 2)$		$+ 0$
$+ (1 * 2^2)$		$+ (1 * 4)$		$+ 4$
$+ (0 * 2^3)$		$+ (0 * 8)$		$+ 0$
$+ (1 * 2^4)$		$+ (1 * 16)$		$+ 16$
$+ (0 * 2^5)$		$+ (0 * 32)$		$+ 0$
$+ (1 * 2^6)$		$+ (1 * 64)$		$+ 64$

$= 84_{10}$

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 $\rightarrow$ 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 $\rightarrow$

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

- Increment: ++
  `i++;`
  is the same as:
  `i = i + 1;`

- Decrement: −−
  `i--;`
  is the same as:
  `i = i - 1;`

Assignment operators.

`+=`

`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;`

`/=`

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

`%=`

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

Summary

- This lecture looked at some of the fundamentals of C++.
- we looked again at the "hello world" program, this time disecting it, and discussing what the components are.
- We then went on to look at variables, and the basic types of information that one can store in a C++ program.
- We considered how this data is *really* stored on the computer, and
- Finally we considered some simple mathematical operations.