

## 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, 28jan07/azhar

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

int main() {

    cout << "welcome!\n";
    cout << "hello from my c++ world\n";

}
```

## output.

- like filling in graph paper
- *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 << "is mac better than PC???\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. *ompile* 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  
⇒ defining a *variable*
- example:

*program code:*

```
int x;
```

*computer's memory:*

x →



### 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.

$$1010100_2 = \begin{array}{r|l} (0 * 2^0) & \\ + (0 * 2^1) & \\ + (1 * 2^2) & \\ + (0 * 2^3) & \\ + (1 * 2^4) & \\ + (0 * 2^5) & \\ + (1 * 2^6) & \end{array} = \begin{array}{r|l} (0 * 1) & \\ + (0 * 2) & \\ + (1 * 4) & \\ + (0 * 8) & \\ + (1 * 16) & \\ + (0 * 32) & \\ + (1 * 64) & \end{array} = \begin{array}{l} 0 \\ + 0 \\ + 4 \\ + 0 \\ + 16 \\ + 0 \\ + 64 \end{array}$$

$= 84_{10}$

### base conversion: 10 to 2.

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

$\Rightarrow 1010100_2$

### 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;`