APS105
Winter 2012

Jonathan Deber
jdeber -at- cs -dot- toronto -dot- edu

Lecture 4
January 23, 2012

Today

- Math
- Characters
- Constants

- Office Hours begin today @ 1:00 pm (BA 2270)
- Labs and Tutorials begin this week

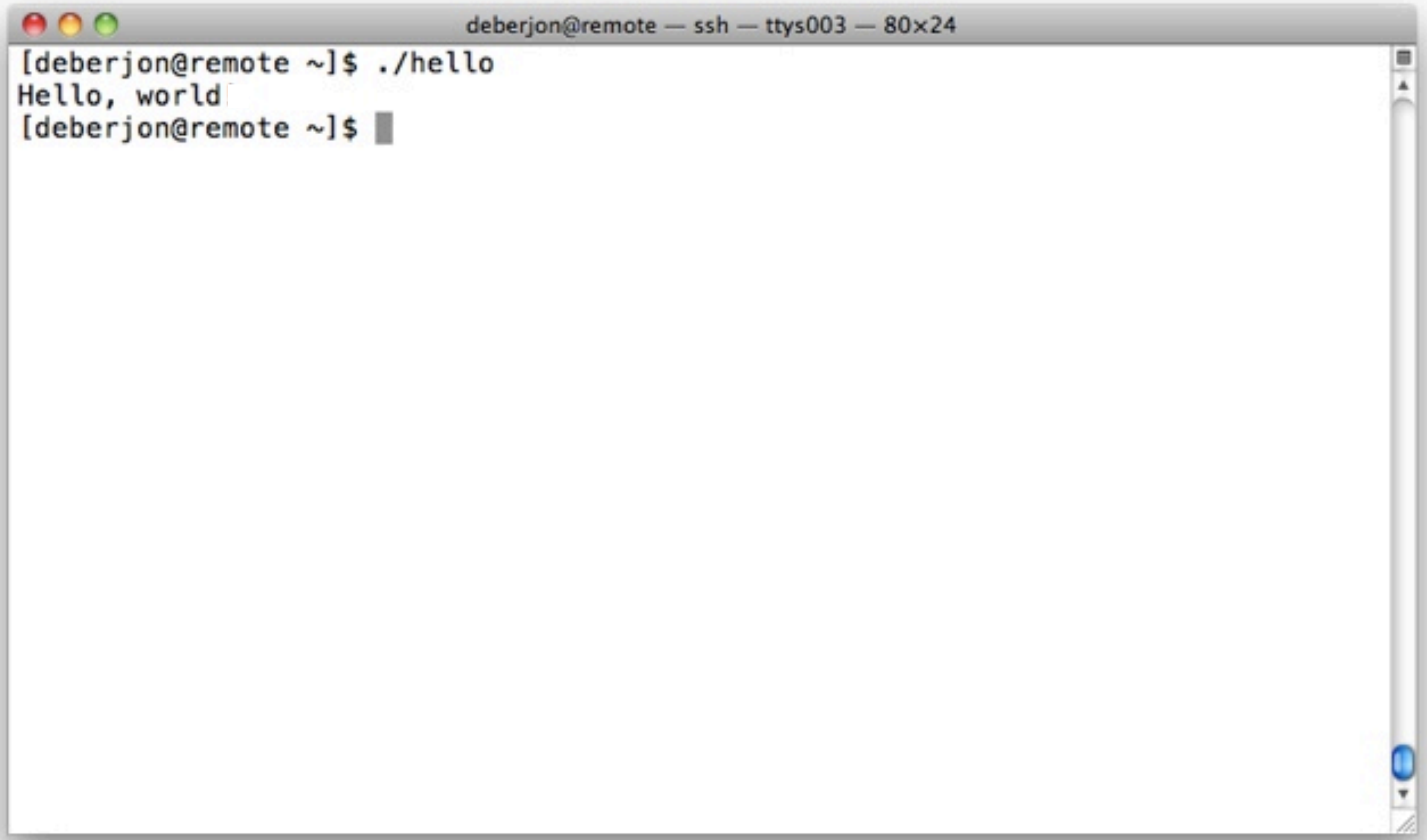
`#include`

- To use `printf`, need to tell compiler where to find it
- `#include <stdio.h>` at beginning of file
- Tells C to use the stdio (standard I/O) library

The Correct Program

- No such thing as *the* correct program

- There are correct/incorrect answers
- There are better/worse programs



```
[deberjon@remote ~]$ ./hello
Hello, world
[deberjon@remote ~]$ 
```

Source: http://us.battle.net/sc2/

Math

Addition and Subtraction

- Mostly what you expect

```
int i = 2;                  double d = 2.0;
int j = 4;                  double e = 4.5;

int k = i + j;              double f = d + e;
k = i - j;                  f = d - e;
```

- Exception is overflow/underflow

Multiplication

- Mostly what you expect (except we use *)

```
int i = 2;                double d = 2.0;
int j = 4;                double e = 4.5;

int k = i * j;            double f = d * e;
```

- Exception is overflow/underflow

Division

- Uses /, since there is no ÷ key
- double
  - What you expect
- int
  - floor division
  - % (modulo or mod)

```
double d = 5.0;
double e = 2.0;
                         2.5
double f = d / e;

int i = 5;
int j = 2;

int k = i / j;  2
int l = i % j;  1
```

Division

$$x = \frac{x}{y} y$$

```
x  is not  (x  /  y)  *  y

x  is (x  /  y)  *  y  +  (x  %  y)
```

Division

- Negative numbers are tricky
- `int i = -5 / 2;`
- Mathematically, that’s -2.5
- But, we can round that to either -2 or -3
- C89 is implementation dependent
- C99 says truncate towards 0

Order of Operations

- What you expect (for basic math)
  - unary + and -, * and /, + and -
  - `-2 * 4 + 5` $\boxed{\texttt{-3}}$
- You can use parentheses, even when not required
  - `-2 * (4 + 5)` $\boxed{\texttt{-18}}$
  - `(-2 * 4) + 5` $\boxed{\texttt{-3}}$

Associativity

- `12 / 4 * 6`
- Is that `(12 / 4) * 6` or `12 / (4 * 6)`?
- Arithmetic operators are left associative

```
 4 * 3 + 4 / 6 * 4 + 9 / 3
(4 * 3) + 4 / 6 * 4 + 9 / 3
(4 * 3) + (4 / 6) * 4 + 9 / 3
(4 * 3) + ((4 / 6) * 4) + 9 / 3
(4 * 3) + ((4 / 6) * 4) + (9 / 3)
```

Short Forms

```
numItems = numItems + 1;
numItems += 1;
numItems++;
++numItems;
```

$$
\begin{aligned}
&\texttt{weight -= 3;} &\longleftrightarrow\quad &\texttt{weight = weight - 3;}\\
&\texttt{ratio /= 4.9;} &\longleftrightarrow\quad &\texttt{ratio = ratio / 4.9;}\\
&\texttt{adjusted *= 4;} &\longleftrightarrow\quad &\texttt{adjusted = adjusted * 4;}\\
&\texttt{numItems =+ 2;} &\longleftrightarrow\quad &\texttt{numItems = (+2);}
\end{aligned}
$$

Wrong

Mixed Types

- C will (mostly) convert for you
- Arithmetic conversion:
  - `int` → `double` as necessary
  - Little to no loss of data
- Assignment conversion:
  - `int` → `double` as before
  - `double` → `int` is truncated

```
int i = 2;
double d = 2.5;
double e = i + d;
```

$\boxed{\texttt{2.0 + 2.5}}$

```
double d = 3;
```

$\boxed{\texttt{3.0}}$

```
int i = 3.14159;
```

$\boxed{\texttt{3}}$

Converting Types

- `2/3 * 9` $\boxed{\texttt{0}}$
- `2.0/3 * 9` $\boxed{\texttt{6.0}}$

- `double d = 2/3 * 9;` $\boxed{\texttt{0.0}}$

Casting

- Manually convert an expression from one type to another
- `(double)2` → `2.0`

- `double d = 2/3 * 9;` $\boxed{\texttt{0.0}}$
- `double e = (double)2/3 * 9;` $\boxed{\texttt{6.0}}$
- `double f = 2/(double)3 * 9;` $\boxed{\texttt{6.0}}$
- `double g = 2/3 * (double)9;` $\boxed{\texttt{0.0}}$
- `double h = (double)(2/3 * 9);` $\boxed{\texttt{0.0}}$

Casting

```
double e = (double)2/3 * 9;
double f = 2.0/3.0 * 9.0;

int n = 10;
int sum = 105;
double allocation = sum / n;  10.0
allocation = (double)sum / n; 10.5
```

Characters

Characters

- Smallest unit of text
  - `'a'`
  - `'5'`
  - `'\n'`
- Type is `char`
- Strings are sequences of characters
  - `"Hi"` is `'H'` followed by `'i'`
  - `"a"` vs. `'a'`

Pronunciation:

```
char acter;
char broiled;
```

Characters

- Need to store (encode) them as `1`s and `0`s
- Could decide something like `a = 1`, `b = 2`, `c = 3`, etc.
- ASCII
  - 128 characters = 2^7 values = 7 bits
  - Stored in 8 bits = 1 byte
  - Some logic, some arbitrariness
  - Only represents what's on an English keyboard

ASCII

- Control characters (0-31)
- Digits (48-57)
- Upper case (65-90)
- Lower case (97-122)
- Basic symbols (everything else)

Characters

- Escape sequences
  - `'\n'` or `'\"'`
  - `'\0'` is character code `0`
  - “Null character”
- `'\0'` vs. `'0'`

printf

- Uses %c format specifier

```
char first = 'J';
printf("My name starts with: %c. \n", first);
```

```
My name starts with: J.
```

Treating chars as Numbers

```
char c = 'a';
c++;              'b'
c++;              'c'

c -= 32;          'C'
```

- Generally, not a great idea
- We’ll see better alternatives later

Constants

Constants

```
#define NUM_STUDENTS 51
#define NUM_TAS 4
#define NUM_PROFS 1
#define NUM_EXTRA 2
```

$\}$ Own line (at top)
No =
No ;

```
int numCopies = 59;
```

$\longleftarrow$ Magic Number

```
int numCopies = NUM_STUDENTS + NUM_TAS
              + (2 * NUM_PROFS) + NUM_EXTRA;

int numCopies = 51 + 4
              + (2 * 1) + 2;
```

Constants

```
#define NUM_STUDENTS 51
...
NUM_STUDENTS = 1;
          51 = 1;
```

Error

```
int NUM_TAS = 4;
...
NUM_TAS = 1;
```

Constants and Style

```
int numCopies = 59;
```
$\longleftarrow$ Magic Number

Bad Style

```
#define FOUR 4
#define TWO_POINT_ONE 2.1
```
Bad Style

```
#define THIRTY 31
```
Evil