Introduction to C

Introduction and Hello World

Instructor: Yin Lou

01/24/2011

Administrivia

- Instructor: Yin Lou, 4144 Upson
- Email: yinlou@cs.cornell.edu
- Lectures: MWF 12:20-1:10pm, HLS 306
- Office Hours: TBD
- `http://www.cs.cornell.edu/courses/cs2022/2011sp/`
- CMS: `https://cms.csuglab.cornell.edu`

Administrivia

- Instructor: Yin Lou, 4144 Upson
- Email: yinlou@cs.cornell.edu
- Lectures: MWF 12:20-1:10pm, HLS 306
- Office Hours: TBD
- `http://www.cs.cornell.edu/courses/cs2022/2011sp/`
- CMS: `https://cms.csuglab.cornell.edu`
  - Email me your netid if you are not in CMS!

Goals

- C syntax
- Standard libraries
- Programming for robustness and speed
- Understanding compiler

Topics

Lect01:	Introduction and Hello World
Lect02:	Control Flow
Lect03:	Functions and Make
Lect04:	Pointers and Arrays
Lect05:	Complex Types
Lect06:	Memory Model
Lect07:	Performance
Lect08:	Preprocessor
Lect09:	Standard Input and Output
Lect10:	File and Variable-length Argument Lists
Lect11:	Threads and Synchronization
Lect12:	To Be a Master Programmer

Environment

- OS: GNU/Linux
- Editor: Vim
- Compiler: gcc
- Debugger: gdb

Introduction to Vim

- $ vim hello.c
- i - Enter **editing mode**
- <esc> - Enter **normal mode**

More Normal Mode Commands

- :w - **W**rite/Save
- :q - **Q**uit
- :q! - **Q**uit without saving
- :wq - **W**rite and **Q**uite

Introduction to Vim

More Commands

- Search
  - / + <pattern>
  - n - next search match
  - N - previous search match
- Goto
  - : + <line no>
  - gg - Goto first line
  - G - Goto last line
- Delete Line
  - dd

Google “vim cheat sheet” for more!

Structure of a C Program

Overall Program

<some pre-processor directives>

<global declarations>
<global variables>

<functions>

Structure of a C Program

Overall Program

<some pre-processor directives>

<global declarations>
<global variables>

<functions>

Functions

<function header>
<local declarations>

<statements>

hello.c: Hello World

```c
#include <stdio.h>

int main()
{
    printf("Hello World\n");
    return 0;
}
```

Compiling and Running

- $ gcc hello.c -o hello
- $ ./hello
  Hello World

What Happens?

- $ gcc hello.c -o hello
  - Compile “hello.c” to machine code named “hello”
  - “-o” specifies the output file name. (Notice it’s case-sensitive.)
- $ ./hello
  - Execute program “hello”
  - “./” is necessay!

What Happens?

- $ gcc hello.c -o hello
  - Compile “hello.c” to machine code named “hello”
  - “-o” specifies the output file name. (Notice it’s case-sensitive.)
- $ ./hello
  - Execute program “hello”
  - “./” is necessay!

hello.c

```c
#include <stdio.h> // "printf" is declared in this header file.

int main() // Main point of execution.
{
    printf("Hello World\n"); // Output "Hello World" to console.
    return 0; // Tell OS the program terminates normally.
}
```

vars.c: Variables

```
#include <stdio.h>

int main()
{
    int a, b, c;

    a = 10;
    b = 20;
    c = a * b;
    printf("a = %d b = %d c = %d\n", a, b, c);
    return 0;
}
```

vars.c: Variables

```
#include <stdio.h>

int main()
{
    int a, b, c;

    a = 10;
    b = 20;
    c = a * b;
    printf("a = %d b = %d c = %d\n", a, b, c);
    return 0;
}
```

```
a = 10 b = 20 c = 200
```

cmdarg.c: Command Line Args

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
    int n, m;

    n = atoi(argv[1]);
    m = atoi(argv[2]);
    printf("Argument 1: %d\nArgument 2: %d\n", n, m);
    return 0;
}
```

cmdarg.c: Command Line Args

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
    int n, m;

    n = atoi(argv[1]);
    m = atoi(argv[2]);
    printf("Argument 1: %d\nArgument 2: %d\n", n, m);
    return 0;
}
```

```
$ ./cmdarg 10 20
Argument 1: 10
Argument 2: 20
```

More on printf

- printf(format_string, val1, val2);

More on printf

- printf(format_string, val1, val2);
  - format_string can include placeholders that specify how the arguments val1, val2, etc. should be formatted
  - %c : format as a character
  - %d : format as an integer
  - %f : format as a floating-point number
  - %% : print a % character

More on printf

- printf(format_string, val1, val2);
  - format_string can include placeholders that specify how the arguments val1, val2, etc. should be formatted
  - %c : format as a character
  - %d : format as an integer
  - %f : format as a floating-point number
  - %% : print a % character

Examples

```
float f = 0.95;
printf("f = %f%%\n", f * 100);
```

More on printf

- printf(format_string, val1, val2);
    - format_string can include placeholders that specify how the arguments val1, val2, etc. should be formatted
    - %c : format as a character
    - %d : format as an integer
    - %f : format as a floating-point number
    - %% : print a % character

Examples

```
float f = 0.95;
printf("f = %f%%\n", f * 100);
```

```
f = 95.000000%
```

More on printf

- Placeholders can also specify widths and precisions
  - %10d : add spaces to take up at least 10 characters
  - %010d : add zeros to take up at least 10 characters
  - %.2f : print only 2 digits after decimal point
  - %5.2f : print 1 decimal digit, add spaces to take up 5 chars

More on printf

- Placeholders can also specify widths and precisions
  - %10d : add spaces to take up at least 10 characters
  - %010d : add zeros to take up at least 10 characters
  - %.2f : print only 2 digits after decimal point
  - %5.2f : print 1 decimal digit, add spaces to take up 5 chars

Examples

```
float f = 0.95;
printf("f = %.2f%%\n", f * 100);
// f = 95.00%
printf("f = %10.2f%%\n", f * 100);
// f =      95.00%
```

Warning about printf

- printf is powerful, but potentially dangerous

What does this code output?

```
int i = 90;
float f = 3;
printf("f = %f i = %d\n", f);
printf("f = %f\n", f, i);
printf("i = %d f = %f\n", f, i);
```