Introduction to C
Geared toward programmers

Sean Ogden

Slide heritage: Alin Dobra → Niranjan Nagarajan → Owen Arden → Robert Escriva → Zhiyuan Teo → Ayush Dubey

Cornell CS 4411, August 30, 2013
Outline

1. Administrative Information
2. Why C?
3. Language Basics
   - Pointers and Memory Management
4. Program Structure and Style
5. Concluding Remarks
Course Staff: same as CS 4410
(cs4410staff@systems.cs.cornell.edu)

Office Hours
- On course webpage
- Immediately after lecture

Lectures will rotate between TAs

Keep checking CS 4411 webpage for announcements!
6 projects, 90% of your grade

- per-project weight not yet decided

- 10% of your grade from flexible components
  - participation in class and discussions
  - helping TAs identify errors
  - ... and lots of other opportunities!

- Strictly 2 members per project team
  - member swaps are allowed between projects but inform course staff first
  - also inform us if you or your teammate plans to drop the course!
Outline

1. Administrative Information
2. Why C?
3. Language Basics
   - Pointers and Memory Management
4. Program Structure and Style
5. Concluding Remarks
Predictability

- No unintended memory allocations.
- Control of layout of structures in memory.
- Simple syntax makes determining behavior easy.
Portability

- Most operating systems have a C compiler available...
- ... (partially) because most operating systems are written in C.

Linux, {Free,Open,Net,Dragonfly}BSD, Minix, XNU
Introduction to C

Why C?

Power

- Direct manipulation of control flow.
- Complex datastructure creation.
- Assembly is just a step away.
But great power can corrupt... *

* which will it be, the stack or the heap?
Goals for today

- Provide a quick primer for using/learning C.
- Warn you of pitfalls related to C programming.
- Practice reading/writing C.
Outline

1 Administrative Information
2 Why C?
3 Language Basics
   - Pointers and Memory Management
4 Program Structure and Style
5 Concluding Remarks
#include <stdio.h>

int main(int argc, char* argv[]) 
{
    printf("Hello World!\n");
}
Try it out

- Install Microsoft SDK
- Set environment variables (path, include) to incorporate bin and include folders of Microsoft SDK
- **Create** `hello.c`.
- In a command prompt run:
  - `cl.exe hello.c`
- **Now run the** `hello.exe` **binary.**
- If `cl.exe` complains about missing DLLs or library include paths, run `vcvars32.bat`

† or fetch it from the course webpage
The Programmer’s Responsibilities

- Memory maintenance (no automatic garbage collection here).
- Error handling (no exceptions here).
Language Basics

- Control flow
- Data types
- Operators
- Preprocessor
Control Flow

- if (x) { } else { }
- for (pre(); loop_invariant(); post()) { }
- while (x) { }
- do { } while (x);
- switch (x) { case 0: break; default: act(); }
Operators

- Arithmetic: +, −, *, /, %
- Relational: <, >, <=, >=, ==, !=
- Logical: &&, ||, !, ?:  
- Bitwise: &, |, ^, !, <<, >>
Primitives

- **Integer types:**
  - `char`: character, always one byte.
  - `int`, `short`, `long`: integers of different sizes.
  - Optional prefixes `signed` or `unsigned`.

- **Floating point types:**
  - `float`: single-precision floating point.
  - `double`: double-precision floating point.

- **No booleans:**
  - `0 ⇒ false`
  - `≠ 0 ⇒ true`
Sample Primitive Declarations

```c
char a = 'A';
char b = 65;
char c = 0x41;

int i = -2343234;

unsigned int ui = 4294967295;
unsigned int uj = ((long) 1 << 32) - 1;
unsigned int uk = -1;

float pi = 3.14;
double long_pi = 0.31415e+1;
```


```c
enum months
{
    JANUARY,
    FEBRUARY,
    MARCH
};

enum days
{
    SUNDAY,
    TUESDAY = 2,
    WEDNESDAY
};
```

- Values are consecutive integers starting from zero.
- Explicitly assigning a value causes the sequence to continue from that value (e.g., `WEDNESDAY == 3`).
Preprocessor

- **Inclusion of other files:**
  ```
  #include <header.h>
  ```

- **Definition of constants:**
  ```
  #define HOURS_PER_DAY 24
  ```

- **Creation of macros:**
  ```
  #define CELSIUS(F) ((F - 32) * 5/9)
  ```
Pointers hold memory addresses

- An int: `int foo;`
- Pointer to an int: `int* ptr_foo;`
- Pointer to a pointer to an int: `int** ptr_foo;`
Pointer operations

- Obtain the address of a variable: `&`
- Dereference a memory address: `*`
Pointers and Memory Management

Pointer Example

```c
int x;
int* ptr; /* ptr points to an undefined location */

ptr = &x; /* ptr now points to integer x */
*ptr = 3; /* integer pointed to by ptr is now 3 */
```
Pass by Value

swap_value() only changes n1 and n2 within the local scope.

```c
void swap_value(int n1, int n2)
{
    int temp;
    temp = n1;
    n1 = n2;
    n2 = temp;
}
```
swap_reference() changes the values at the memory locations pointed to by \texttt{p1} and \texttt{p2}.

```c
void swap_reference(int* p1, int* p2)
{
    int temp;
    temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
```
Function Pointers

```c
int inc(int i) { return i + 1; }
int dec(int i) { return i - 1; }
int apply(int (*f)(int), int i)
{
    return f(i);
}
int main(int argc, char* argv[])
{
    printf("++: \%i\n", apply(inc, 10));
    printf("--: \%i\n", apply(dec, 10));

    return 0;
}
```
Heap Variables

- Allocate with `malloc()` and deallocate with `free()`.
  ```c
  void* malloc(int)
  void free(void*)
  ```
- It is up to you to manage memory.
- Never calling `free()` leaks memory.
- Calling `free()` more than once will crash your program. ‡

‡ Known as a “double free” bug, and is sometimes exploitible.
Malloc/Free Example

```c
int* ptr; /* pointer to an int */

/* allocate space to hold an int */
ptr = (int*) malloc(sizeof(int));

/* check if successful */
if (ptr == NULL) exit(1);

*ptr = 4; /* store value 4 in the int */
printf("ptr: %p %d\n", ptr, *ptr);

/* deallocate memory */
free(ptr);
```
Warning

- Dereferencing an un-initialized pointer can crash your program (or worse)!
- Consider initializing a pointer to NULL and checking before dereferencing.
- Some functions return NULL on error.
  - Pay attention to the function specification!
  - Check return values!
### Arrays

```c
for (i = 0; i < 10; ++i)
{
    A[i] = i * i;
}
```

### Strings

```c
char name[] = "CS4410";
name[5] = '1';
```

### Functions to operate on strings in `string.h`

- `strncpy`, `strncmp`, `strncat`, `strstr`, `strchr`
Outline

1. Administrative Information
2. Why C?
3. Language Basics
   - Pointers and Memory Management
4. Program Structure and Style
5. Concluding Remarks
Structures

```c
struct list_elem
{
    int data;
    struct list_elem* next;
};

int main(int argc, char* argv[])
{
    struct list_elem le = { 10, NULL };  
    le.data = 20;

    return 0;
}
```
Typedefs

- Create an *alias* for a type.
- **Syntax:** `typedef type alias`
- **Use it like any primitive:** `list_elem_t le;`

```c
typedef struct list_elem
{
    int data;
    struct list_elem* next;
} list_elem_t;
```
Code Style

- Write comments where appropriate!
  - Bad comment:
    /* print the value of x */
  - Good comment:
    /* take one process off the ready queue */

- Create logically names constants and variables.
  - Bad naming:
    struct lock mylock, lock1
  - Good naming:
    struct lock process_queue_lock

- Split large functions into smaller functions.
Outline

1. Administrative Information
2. Why C?
3. Language Basics
   - Pointers and Memory Management
4. Program Structure and Style
5. Concluding Remarks
C is a great language (especially for systems work).

Save yourself some trouble:
- Initialize variables before use.
- Don’t return pointers to stack-based variables.
- Allocate and deallocate memory properly.
- Check return values.
- Use version control! – keep track of changes, merging changes from all developers (check out SVN, CVS, Git etc.)
Some advice

- Start projects early!
- If you feel a problem is unclear or underdefined, seek clarification or make reasonable design assumptions.
- Don’t put off questions if you have any.
- Don’t think of this course as ‘course staff vs students’, we are all here to learn, TAs included.
Introduction to C

Sean Ogden
sean @ cs
August 30, 2013