Introduction to C

Kerry Xing
Outline

We will cover aspects of C that are

● Identical to Java.
● Slightly different in Java.
● Completely different in Java.
/* This is a comment in C. */

// Double slash works in C99.
Syntax in Common (cont.)

```java
if (x > 5) {
    // x is greater than 5
} else {
    // x is at most 5
}
```
Syntax in Common (cont.)

for (int i = 0; i < 10; i++) {
    // something...
}

Syntax in Common (cont.)

while (x <= 10) {
    // do something...
}

do {
    // something ...
} while (y > 5);
```java
switch (harvard_grade / 10) {
    case 10:
    case 9:
    case 8:
    case 7:
        return "A+";
    default:
        return "A";
}
```
Syntax in Common (cont.)

```c
int x = 5;
int y = 8;

x++;  
x += 7;

y = (x * x) + 42;
```
Slight Differences in C

We will cover:

● Types
● Enum
● Functions
● Printing
● Structs
Types

- `int`, `char`, `short`, `long`, `long long`, `float`, `double`
- Integers can be used as booleans
- Signed/unsigned
Enums

class TShirtSize {
    SMALL = 1,
    MEDIUM = 2,
    LARGE = 3,
    XLARGE = 4,
    HOMER_SIMPSON = 9999
}
;
Functions

```c
int sum(int x, int y) {
    return x + y;
}
```
Printing

In Java:
System.out.println(String.format("%d is my age.\n", 21));

In C: (make sure you include <stdio.h>)
printf("%d is my age.\n", 21);
Structs

- Objects--
- No constructors, no functions

Example:

```c
struct Point {
    int x;
    int y;
};
```
Structs

Gotchas:
● Remember the semicolon when declaring.
● Use “struct” whenever referencing the type.

```c
struct Point {
    int x;
    int y;
};
struct Point p;
```
Completely Different in C

- Pointers
- Arrays
- Strings
- Memory Allocation
- Typedefs
- Preprocessor
- Compilation
- Function Pointers
Pointers

- Stores the memory address of some data.
- “Points” to data
Pointers

Creating a pointer variable:

```c
int x = 5;

int* a = &x;
int* b = &x;
int* c = &x;
```
Pointers

Small gotcha:

```c
int* a, b;
```

*a* is an integer pointer, but *b* is an integer!

More like:

```c
int *a, b;
```
Pointers

Dereferencing - retrieving the data from the pointer

Example:

```c
int x = 5;
int *y = &x;
*y = 6;
printf("%d\n", x); // Prints out 6.
```
Pointers

NULL pointers

- Address of 0.
- Sentinel value.

Example:

```c
int *x = 0;
int *y = NULL;
```
Pointers

Pointer arithmetic

Example:

```c
char *c;
...
while (*c != 0) {
    c++; // same as c = c + 1
}
```
Arrays

- Block of data.
- Arrays are pointers!

Example:

```c
int my_array[10];
int *p = my_array;

*(p + 2) and my_array[2] are the same!
```
Strings

- Strings are char arrays!
- Strings in the null character: `\0`

Example:

```c
char my_array[10] = "Hello";
// my_array[0] == 'H'
// my_array[4] == 'o'
// my_array[5] == '\0'
```
Memory Allocation

malloc - memory allocation
free - free allocated memory

Example:

```c
int *x = malloc(80);
// Allocates 80 bytes.
free(x);
// Frees the allocated 80 bytes.
```
Memory Allocation

Gotchas:
● Remember to free when you are done with allocated memory.
● Don’t free something you didn’t allocate.
● Don’t double free.

Bottom line:
Memory allocators need to be fast, so they don’t have time to do sanity checking for you!
Typedefs

Abstracting out types!

typedef char smallint;
typedef long long long bigint;

bigint x;
smallint y;
Preprocessor

Directives start with #.
Preprocessor runs before your compiler.

Examples:
#include <stdio.h>
#define SOCKS 2
#ifndef NDEBUG
#else
#endif

#include

Practically copy-pastes the code in another file into your file.
Angle brackets for system files.
Double quotes for your own files.

#include <stdio.h>
#include "my_file.h"
#define

Defines constants and macro functions. The preprocessor effectively replaces the names with their corresponding expressions.

Example:

```c
#define SOCKS 2
// int x = SOCKS; → int x = 2;
#define SUM(x, y) ((x) + (y))
// int a = SUM(2, 3); →
//     int a = ((2) + (3));
```
#define

Gotchas:

- Use parenthesis!
- Be careful with your inputs.


Why we use parenthesis in macros:

```
#define SQUARE(x) (x * x)
```

`SQUARE(5 + 7)` won’t give you 144. It is:

```
(5 + 7 * 5 + 7) = 47
```
#define

Why we have to be careful with inputs:

#define SQUARE(x) ((x) * (x))

int a = 5;
SQUARE(++)a is not 36. It is:

((++a) * (++a)) = (7 * 7) = 49
#ifdef, #ifndef, #else, #endif

Selectively includes code depending on what is defined.

Example:

```c
void assert(int x) {
  #ifndef NDEBUG
    if (!x)
      exit(1);
  #endif
}
```
Compilation

- Preprocessor phase
- Compile phase
- Link phase
Preprocessor Phase

Goes through the preprocessor directives (#)

- `#include`
- `#define`
- `#ifdef, #ifndef, #else, #endif`
Compile Phase

Each source file gets turned into an object file.

source.c → source.o

Generated code may be optimized by the compiler.
Linker Phase

- The object files only know that certain functions exist, but don’t know where they are.
- The linker “links” function calls and the actual function definitions.
Organizing Your Code

In general:

● Declare your functions in header files (.h).
● Declare any “public” structures and typedefs in header files (.h).
● Implement your functions in source files (.c).
Organizing Your Code (cont.)

Example:

my_math.h:
int square(int x);

my_math.c:
int square(int x) {
    return x * x;
}
Organizing Your Code (cont.)

If you have a small function, you may get performance gains by inlining: (be sure you use static!)

my_math.h

```c
static int square(int x) {
    return x * x;
}
```
Organizing Your Code (cont.)

Use header guards to make sure you don’t include the same header file twice.

Example:

hello.h

```c
#ifndef HELLO
#define HELLO
// code here.
#endif
```

Function Pointers

Pointer to a function!

Example:

```
int (*transformation)(int);
```
Function Pointers (cont.)

int triple(int x) {
    return 3 * x;
}

int quadruple(int x) {
    return 4 * x;
}
Function Pointers (cont.)

```c
int main() {
    int (*t1)(int) = &triple;
    int (*t2)(int) = &quadruple;

    printf("%d\n", t1(5)); // 15
    printf("%d\n", t2(5)); // 20
    return 0;
}
```
Other Resources

www.cprogramming.com
www.cplusplus.com