C tutorial

Embedded Systems
Electrical and Computer Engineering,
Cornell University
ece3140-staff@csl.cornell.edu

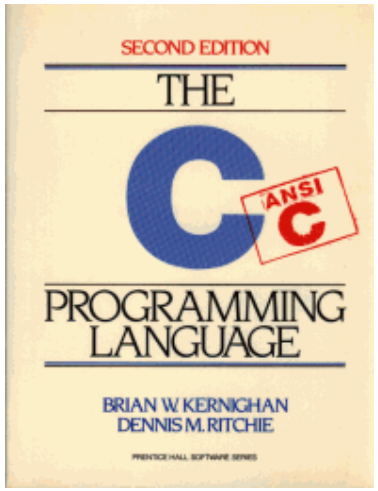
Goal of this tutorial

- Introduce basic C language concepts
- NOT to teach you all there is to know about C

Additional resources

- **The C Programming Language**
  by Brian W. Kernighan and Dennis M. Ritchie,
  a.k.a. "K & R"

- **C for Java programmers**
  www.cs.cornell.edu/courses/cs316/2006fa/cforjava.php



Contents

- **Data types**
- Operators
- Control flow
- Structure of a C program
- Arrays & Structs

C data types & ARMv6-M

Type	Size (bits)	Description
char	8	ASCII character
short	16	integer value
int	32	integer value
long	32	integer value
float(double)	32(64)	Not available in all –M CPUs

No boolean type

Data type modifiers

- unsigned/signed
  - Applies to integer data types
  - Unsigned represents the variable as a binary value
  - Signed represents the variable using 2's complement
- const/volatile
  - Applies to all data types
  - const: variable will not change its value
  - volatile: variable may be changed unpredictably outside of program control

Data types examples

- signed int x;
  - Maximum value of x?
  - Minimum value of x?
- unsigned char y;
  - Maximum value of y?
  - Minimum value of y?

Data types examples

- signed int x;
  - Maximum value of x? $2^{31}-1$
  - Minimum value of x? -2^{31}
- unsigned char y;
  - Maximum value of y? 255
  - Minimum value of y? 0

Contents

- Data types
- **Operators**
- Control flow
- Structure of a C program
- Arrays & Structs

Operators

Arithmetic	+, −, *, /, % +=, −=, *=, /=, %=
Relational	>, >=, <, <=, ==, !=
Logical	&&, ||
Increment/decrement	++, −−
Bitwise	&, |, ^, <<, >>, ~
Ternary	$expr_1$? $expr_2$: $expr_3$

Manipulating bits with masks

- Set the 3rd bit of a variable
- Clear the 1st and 2nd bits of a variable
- Toggle the 4th bit of a variable

Manipulating bits with masks

- Set the 3rd bit of a variable

```
var |= 0x04;

var |= (1 << 2);
```

- Clear the 1st and 2nd bits of a variable

```
var &= ~0x03;
var &= ~((1 << 0) | (1 << 1));
```

- Toggle the 4th bit of a variable

```
var ^= 0x08;
var ^= (1 << 3);
```

Contents

- Data types
- Operators
- **Control flow**
- Structure of a C program
- Arrays & Structs

Control flow: if

```
if (expression) {
    block of statements
}
```

Control flow: if-else

```
if (expression) {
    block of statements
} else {
    block of statements
}
```

Control flow: if-else if-else

```
if (expression) {
    block of statements
} else if (expression) {
    block of statements
} else {
    block of statements
}
```

Control flow: switch

```
switch (variable) {
  case value1:
    statements
    break;
  case value2:
    statements
    break;
  default:
    statements
    break;
}
```

Control flow: while

```
while (expression) {
        block of statements
}
```

- Loop statement
- Executes the block of statements while “expression” is true.
- The “expression” is checked “before” the statements

Control flow: for

```
for (expr1; expr2; expr3) {
    block of statements
}
```

- Loop statement
- “expr1” initializes the loop;
- “expr2” tests if loop execution should continue
- “expr3” is executed after each loop iteration

Control flow: for examples

```
for (i = n; i > 0; i--) {
    block of statements
}
```

```
for (;;) {
    block of statements
}
```

Control flow: do-while

```
do {
      block of statements
} while (expression);
```

- Loop statement
- Similar to the while loop, but the test expression is checked at the end

Control flow: break/continue

- **break** - immediately exit from the innermost enclosing loop or switch
- **continue** - move on to the next iteration of the innermost enclosing *for*, *while*, or *do* loop

Contents

- Data types
- Operators
- Control flow
- **Structure of a C program**
- Arrays & Structs

Structure of a C program

<*pre-processor directives*>

<*global declarations*>

<*function definitions*>

- Every C program must have a `main()` function, which is where execution begins

C preprocessor

- A program called as the first part of the compilation
- File inclusion

  #include <foo.h>
- Macro definition

  #define foo 4
- Conditional inclusion

  # if

File inclusion

```
#include <string.h>
```

Include standard library interfaces

```
#include ”myheaderfile.h"
```

Include declarations defined in other files

Macro definition

- `#define` performs text substitution
- Define constant values

```
#define PI 3.1416
float x = PI;
```

- Create simple function macros

```
#define abs(x) (x < 0) ? -x : x
int w; int y = -10;
w = abs(y);
```

Conditional inclusion

- Used to conditionally include (or remove) pieces of code (`#if`, `#ifdef`, `#ifndef`)

```
#define DEBUG
#ifdef DEBUG
  block of statements
#else
     block of statements
#endif
```

Conditional inclusion

- Used to include files only once

```
#ifndef HEADER
# define HEADER
  /* contents of header */
#endif
```

C preprocessor examples

- What is the difference between these two lines?

```
#define PI 3.1416
const float pi = 3.1416;
```

- What is the value of `x`?

```
#define Y 5+2
int x = 3*Y;
```

Global declarations

- Variables must be defined before use `int x;`

```
int y = 30;
char lastname[25];
```

- Function declaration (prototype)
  - <return_type> function_name(<arguments>);`int foo(int, float);`

```
void bar(char);
```

Function definitions

<return_type> function_name(<arguments>) {
    function body
}

```
int max(int a, int b) {/* Function body */}

void main(void) {/* Function body */}
```

- Function must have return type or `void`
- Arguments and return are passed by **value**

Structure of a C program

```
#include “myheaderfile.h”

int x;

char y = 'B';

int foo(int x, int y);

void foo2(void);


int foo(int x, int y) {/*Must match prototype*/

  return x+y;

}
```

Contents

- Data types
- Operators
- Control flow
- Structure of a C program
- **Arrays & Structs**

Arrays

- Data structure consisting of a collection of elements that reside contiguous in memory

- To declare an array:

```
int x[3];
```

An integer array consisting of 3 elements

Arrays

- Can be initialized

```
int x[3]={1,2,3}
```

- If initialized, you do not need to supply size of array

```
int x[] = {1,2,3}
```

- Can be multidimensional

```
int m[2][3];
```

- Contents can be accessed with an index to the array

```
x[2]=3;
```

- The first element of an array is at index "0"

Arrays

Example:

```
int x[3]={9,10,1};
```

0	*1*	*2*
9	10	1

What is the value of x[3]?
What is the value of x[2]?

Arrays

Example:

```
int x[3]={9,10,1};
```

0	*1*	*2*
9	10	1

What is the value of x[3]? unknown
What is the value of x[2]? 1

char arrays

Strings in C are stored as arrays of `char`, terminated by nul character `'\0'`

```
  char greeting[] =
{'h','e','l','l','o','\0'};
```

C also provides a shorthand for this initialization:

```
    char greeting[] = "hello";
```

Multidimensional Arrays

C stores multidimensional arrays in "row-major" order

Example:

```
int matrix[2][3] = {{1,2,3}, {4,5,6}};
```

1	2	3
4	5	6

would be laid out in memory as:

```
1 2 3 4 5 6
```

Structs

- A collection of grouped variables
- Similar to objects in Java/C++

```
struct structName{
        block of local variables
};
struct structName myStruct;
```

Structs

- Example

```
struct coord{
        int x; int y;
};
struct coord mycoord={1,2};
mycoord.x = 3;          //x=3
mycoord = another_coord;
```

Unions

- Holds objects of different types and sizes, one at a time
- Type retrieved must be the type last stored
- Definition and access similar to struct

Example:

```
union myUnion{
    int ival;
    float fval;
};
```

```
union myUnion u;
u.ival = 6;

union mUnion *pu;
pu->fval = 3.14;
```

Typedefs

Used to create new datatype names

```
typedef int Length;
Length maxlen;
Length *len;

typedef struct coord {
    int x; int y;
} Coordinate;
Coordinate p1;
```

don't need `struct` keyword

C tutorial

Questions?

Embedded Systems
Electrical and Computer Engineering,
Cornell University
ece3140-staff@csl.cornell.edu