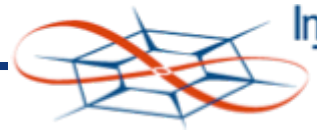
EECS 388
C Introduction

Gary J. Minden
August 29, 2016



Information and Telecommunication Technology Center

THE UNIVERSITY OF KANSAS

C

- Developed at AT&T Bell Laboratories in the early 1970s by Dennis Richie
- Intended as a systems programming language, that is used to write operating systems and embedded kernels
- Efficient, code is fast and compact
- Intended to be "somewhat" portable, but that is **not** always the case
- Popularized through the UNIX and Berkeley UNIX (BSD) operating systems initially and today through Linux
- C programming systems come with an "almost" standard set of library subroutines (libc.a)

C References

- There are many books on C
  - S. P. Harbison and G. L Steele, Jr., C: A Reference Manual, 3 ed., Prentice Hall, Englewood Cliffs, NJ, 1991, ISBN: 0-13-110933-2.
- Other References
  - The International Standard for C
  - D. Richie's C Reference Manual
  - The GNU libc reference
  - C Style Guides
  - Available at: http://www.ittc.ku.edu/~gminden/Embedded_Systems/Content/Datasheets.html

C Identifiers

- Identifiers name programming entities
  - Variables, constants, subroutines, data types, and data structures
- Identifiers must start with an underscore "_" or a letter [a...z] or [A...Z]
- After the initial underscore or letter, identifiers can contain underscores, letters, or digits
- Identifiers are case sensitivity
  - "abc" is distinct from "Abc"
- ANSI (American National Standards Institute) requires at least 31 leading characters to be significant
  - Many implementations support more significant characters
- Longer identifiers tend to increase program clarity and mixing upper and lower case and use of underscores leads to more readable identifiers.

C Identifier Examples

- Examples of C Identifiers are:

  Task_1_Init
  SysTickCount
  Task_2_State

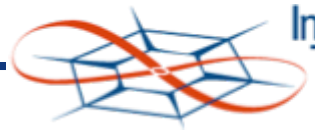
- Identifiers are automatically created in the compiler symbol table when you define variables, constants, subroutines, data types, and data structures
- There may be additional restrictions or allowances on identifiers when interfacing to system software
  - E.g. Identifier significant length or additional characters, respectively

C Primary Data Types

- C enables defining new data types in terms of existing data types
- For most of our work, we will use pre-defined data types.
- The common data types we will use are:

int	**int32_t**	**uint32_t**
short	**int16_t**	**uint16_t**
char	**int8_t**	**uint8_t**
	bool	

- **int32_t** represents a 32-bit number
  - If treated as a two's complement number, values are between 2^-31 and 2^31 - 1
  - If treated as a unsigned number, values are between 0 and 2^32-1
- **char** represents a 8-bit, integer value
  - Also used for single characters
- The ARM processors we use does not support the floating point type **float**



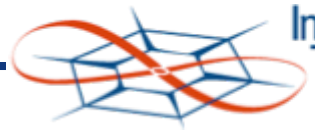
THE UNIVERSITY OF KANSAS

C Primary Data Types

- **enum** is a set of values
  - enum States { On, Off, TriState } GateState;
  - Usually treated as **int32_t**
- **struct** is a collection of typed values
  - struct S { int Red; int Green; int Blue } Color;
  - Color.Red;
- **typedef --** a new type is being defined
  - typedef struct S { int Red; int Green; int Blue } S;
  - struct S Color1, Color2;
- **union --** multiple views of the same set of bits
  - union U { int A; char Word[4]; } Foo;
  - Foo.A; Foo.Word[0]

Type Qualifiers (Old way)

- **unsigned**
  - To declare a variable unsigned, use the **unsigned** modifier
  - unsigned int A, B, C;
- **long**
  - To declare a variable as long (generally 32-bits), use the **long** modifier
  - This is not needed in our programs, integers are 32-bits by default
  - You will see it in some example code because it is good practice to be explicit
- **short**
  - To declare a 16-bit integer, use the **short** modifier
  - unsigned short int A, B, C;
- **volatile**
  - Values are altered outside the control of the program
  - volatile uint32_t SysTickCount = 0;



THE UNIVERSITY OF KANSAS

Numbers, Characters, and Strings

- Constant integers can be represented in decimal radix or hexadecimal radix
  - Decimal radix is just a sequence of digits, e.g. 5463
  - Hexadecimal radix is preceded by "0x" and followed by the digits 0...9 and letters a...f (or A...F), e.g. 0xFF03 would be a 16-bit integer
- Characters (**char**) are 8-bit integers
  - Character constants are enclosed in single quotes, e.g. 'A', '4', and '\n'
  - The backslash enables representing control characters: '\n' represents a new-line character (named Linefeed in ASCII), '\0' represents the Null (value 0) character which terminates strings.
- A sequence (array) of charterers is a string
  - String constants are defined with double-quotes, e.g. "TickCount: %d"
  - Strings are terminated by a Null character, '\0' or 0x00

Arrays

- **int32_t**, **int16_t**, and **char** can be organized into a sequence or vector of values
- Example syntax is:

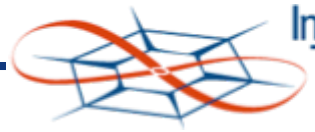
```
int32_t DataInput[ k ];
char InputString[ 256 ];
```

- Multi-dimensional arrays can be allocated

```
int32_t Image[ 16 ] [16 ];
```

  represents a 16x16 image of 32-bit integers

- The first element in an array has index 0. The last element has index <size-of-array>-1



THE UNIVERSITY OF KANSAS

Initializing Variables

- Variables can be initialized at compile time. The syntax is:

  declaration = expression;

- Where declaration is a variable definition and expression evaluates to a value. Examples are:

  int32_t A = 54;
  char theChar = 'A';

- Initial array values are in braces:

  int32_t Table[5] = { 0, 2, 4, 8, 16 };
  char Encode[4] = { 'z', 'y', 'x', 'w' };

Storage Classes

- **extern --** the variable or function is visible outside the program module (file)
- **register --** the variable is used a lot and is a hint to the compiler the allocate in a hardware CPU register, if possible
- **static --** the variable or function is not visible outside the program module

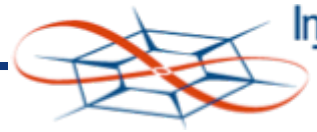
Assignment Statements

```
X = A + B + C * ( D / E );
StartTime = GetCurrentTime();
```

Conditional Statements

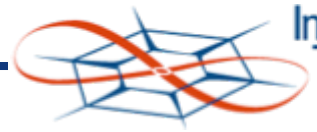
```
if ( A == B ) {
  StartTime = GetCurrentTime();
};

if ( A <= B ) {
  Foo = 24;
} else {
  Foo = 32;
};
```



Conditional Operators

- `Relational operators: ==, != <>, <, <=, =>, >`
- `Bitwise operators: &, |, ^`
- `Logical operators: &&, ||, !`



Iterative Statements (for)

- for (E1; E2; E3) { ... };
  - E1 evaluated first
  - E2 evaluated at the beginning of the loop
  - E3 evaluated at the end of the loop

```
for ( i = 0; i < K; i++ ) {
Sum = Sum + Data[ i ];
};
```

Iterative Statements (while, do)

- Simple loops
  - **while** tests at the beginning
  - **do** tests at the end

```
while (1) {
   Task1_Execute();
   Task2_Execute();
}
```

Selection Statements (switch)

```
switch ( GateState ) {
   case On:
      Foo = 23;
      break;
   case Off:
      Foo = 45;
      break;
   default:
      Foo = 67;
};
```

Functions

```
int Sub1( int A, int B, char S ) {
   ...
   ...
   return ( B + A );
}

X = Sub1( 4, 5, 'B' );
```