Overview

- Announcements
- C History
- Intro to C
- Variables
- Arrays & Strings

Announcements

- HW 1: Due Sunday (1/24) at midnight
- Lab 1: this week

History of C

C was developed in parallel with UNIX
- Martin Richards wrote BCPL in mid 1960s
- Ken Thompson wrote B in 1970
- Dennis Ritchie designed most creative parts of C in 1972
- C is used to re-write UNIX in 1973
- Dennis Ritchie and Brian Kernighan wrote “The C Programming Language” in 1978
- C was standardized during 1983-1988 by ANSI

C and its predecessors were designed as system programming languages
- BCPL needs to be compiled on a DEC PDP-7 machine with 8K 18-bit words
- B was used to write utility programs on a DEC PDP-11 with 24KB memory running UNIX
- C was used to re-write that UNIX on the same machine

C has influenced many modern programming languages
- Java, C++, Obj-C, C#, PHP, ...
INTRO TO C

http://class.ece.iastate.edu/cpre288

1/19/2016

1

Compare C and Java/C++

• C is a procedural language
  – No classes or objects
  – “Function” is the building block
• C philosophy
  – As simple as possible
  – Uses a minimum set of language constructs
• “The C programming language”
  – Quick Overall Intro: Chapter 1 (pgs 5 – 34)
  – Chapter 2
• Course Webpage: Resources sections

Simplest Embedded Program

```
void main()
{
    while (1); // do forever...
}

• Most embedded programs run forever
```

http://class.ece.iastate.edu/cpre288

2

Hello World!

```
#include <stdio.h>

void main()
{
    printf("hello, world\n");
}
```

To build and run on a Linux/unix machine:

```
$ gcc -o helloworld helloworld.c
$ ./helloworld
hello, world
```

Some C Elements

```
; A semicolon marks the end of an expression; a C statement is an expression ended with a semicolon

{} Braces mark a code block

// or /* ... */ Comments
```

http://class.ece.iastate.edu/cpre288

3

Expression and Statement

Which of the follow are valid C statements?

```
a = a + b;
a;
a + b;
10 + 20;
a = (b = c);
```

http://class.ece.iastate.edu/cpre288

4
Expression and Statement

Syntax gotchas?

// sum up all elements in an array
for (i = 0, sum = 0; i < N; i++):
    sum += X[i];

// if flag is set, print a message
if (flag = 1)
    print ("flag has been set");

// enter an idle loop
while (1)
;

Java and C

• Syntactically very similar
  — Loops (for and while), if statements, switch statements.
• Some minor differences
  — Syntax for declaring arrays
  — Classes vs. structures
• Major differences
  — Java has a String data type. C does NOT, instead uses an array
    of characters (we call it a C-string)
  — Pointers: Java does not have this concept

Variables

• Variables are the primary mechanism for storing data to
  be processed by your program
• Naming rules are similar to Java
• Examples:
  — area, graph, distance, file1, file2, height, wheel_right
• The underscore is the only punctuation mark allowed
• Must start with a letter or underscore, no digit
• Case sensitive
  — MyVariable is different from myvariable

Reserved Words: Primitive Data Types

• char
• short
• int
• long
• double
• Float
• enum
• struct
• union
• typedef
• break
• case
• continue
• default
• do
• else
• for
• goto
• if
• return
• switch
• while
• auto
• const
• extern
• register
• signed
• static
• unsigned
• volatile
• sizeof
• void
Variables

- Like Java, a variable must be declared by specifying the variable's name and the type of information that it will hold.

```
data type variable name
int total;
int count, temp, result;
```

Multiple variables can be created in one declaration.

Variables

- A variable can be given an initial value in the declaration.
- If no initial value is given, do not assume the default value is 0.

```
int sum = 0;
int base = 32, max = 149;
int k, i;
for (i = 0; i < 10; i++) {
    k = k + 1;
}
```

Primitive Types and Sizes

<table>
<thead>
<tr>
<th>Name</th>
<th>Number of Bytes sizeof()</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>signed char</td>
<td>1</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>unsigned char</td>
<td>1</td>
<td>0 to 255</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>-32,768 to 32,767</td>
</tr>
<tr>
<td>unsigned short</td>
<td>2</td>
<td>0 to 65,535</td>
</tr>
<tr>
<td>int</td>
<td>Varies by platform</td>
<td>Varies by platform</td>
</tr>
<tr>
<td>int (on Atmega 128)</td>
<td>2</td>
<td>-32,768 to 32,767</td>
</tr>
<tr>
<td>(pointer)</td>
<td>Varies by platform</td>
<td>Varies by platform</td>
</tr>
<tr>
<td>(pointer on Atmega 128)</td>
<td>2</td>
<td>Address Space</td>
</tr>
</tbody>
</table>

- Primitive types in C: char, short, int, long, float, double.
- Default modifier on primitive types is signed (not unsigned).

Variables: Size

```
char sum_char = 0;
int sum_int = 0;
long sum_long = 0;
```

- `sum_char` value is a 8-bit value:
  - Binary: 0b0000 0000
  - Hex: 0x0
- `sum_int` value is a 16-bit value:
  - Binary: 0b0000 0000 0000 0000
  - Hex: 0x0000
- `sum_long` value is a 32-bit value:
  - Binary: 0b0000 0000 0000 0000 0000 0000 0000 0000
  - Hex: 0x0000 0000

Primitive Types and Sizes

<table>
<thead>
<tr>
<th>Name</th>
<th>Number of Bytes sizeof()</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>4</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>signed long</td>
<td>4</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>unsigned long</td>
<td>4</td>
<td>0 to 4,294,967,295</td>
</tr>
<tr>
<td>long long</td>
<td>8</td>
<td>-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>±1.175e-38 to ±3.402e38</td>
</tr>
<tr>
<td>double</td>
<td>Varies by platform</td>
<td>Varies by platform</td>
</tr>
<tr>
<td>double (on Atmega 128)</td>
<td>4</td>
<td>±1.175e-38 to ±3.402e38</td>
</tr>
</tbody>
</table>

- double is an alias to float on the Atmega 128
- Primitive types in C: char, short, int, long, float, double.
- Default modifier on primitive types is signed (not unsigned).

Variables: Size

```
unsigned char my_number = 255;
unsigned char my_number_too_big = 257;
```

- my_number in:
  - Binary: 0b1111 1111
  - Decimal: 255
- my_number_too_big in:
  - Binary: 0b1 0000 0001
  - Decimal:
**Variables: Size**

```c
unsigned char my_number = 255;
unsigned char my_number_too_big = 257;
```

- **my_number in:**
  - Binary: 0b1111 1111
  - Decimal: 255

- **my_number_too_big in:**
  - Binary: 0b1 0000 0001 // Need 9-bits, too big for a unsigned char.
  - Decimal: // the C compiler will truncate to 8-bits

**Simple Program**

```c
void main()
{
    int num_apples, num_oranges = 0;
    int num_fruits = 0;

    num_apples = 5;
    num_oranges = 4;
    num_fruits = num_apples + num_oranges;
}
```

**Arrays in C**

- Sequence of a specific variable type stored in memory
- **Zero-indexed** (starts at zero rather than one)
- Define an array as
  ```c
  Type VariableName [ArraySize];
  Example: int my_array[100]
  ```
- Last element is found at N-1 location
- Curly brackets can be used to initialize the array

**Arrays in C**

- Sequence of a specific variable type stored in memory
- **Zero-indexed** (starts at zero rather than one)
- Define an array as
  ```c
  Type VariableName [ArraySize];
  Example: int my_array[100]
  ```
- Last element is found at N-1 location
- Curly brackets can be used to initialize the array
Arrays in C

• Examples:

```c
// allocates and initializes 3 chars
char myarray1[3] = {2, 9, 4};

// allocates memory for 4 char's
char myarray2[4];

// allocates memory for 2 int's
int myarray3[2];
```

• You do not have to specify the size if the array is being initialized during the declaration

```c
char myarray1[] = {2, 9, 4};
```

• When defining an array, the array name is the address in memory for the first element of the array

  • `myarray3 == ??`

```c
Myarrays[]
```

• Be careful of boundaries in C
  – No guard to prevent you from accessing beyond array end
  – Write beyond array = Potential for disaster

• What exactly is an array?
  – Not a specific type
  – Pointer to a block of memory
  – No built-in mechanism for copying arrays

Examples:

```
char myarray1[3] = {2, 9, 4};
char myarray2[4];
int myarray3[2];
```
Arrays in C

- Examples:
  char myarray1[3] = {2, 9, 4};
  char myarray2[4];
  int myarray3[2];

  myarray1[0] // First element of myarray1

  Memory Address
  FF00 FF01 FF02 FF03 FF04 FF05 FF06 FF07 FF08 FF09 FF0A
  Value 0x02 0x09 0x04 ? ? ? ? ? ? ? ?
  Array myarray1 myarray2 myarray3
  Index 0 1 2 0 1 2 3 0 1

  myarray1[3] // Passed end of myarray1!!!
  // Overwrote myarray2!!

  Memory Address
  FF00 FF01 FF02 FF03 FF04 FF05 FF06 FF07 FF08 FF09 FF0A
  Value 0x02 0x09 0x04 ? ? ? ? ? ? ? ?
  Array myarray1 myarray2 myarray3
  Index 0 1 2 0 1 2 3 0 1

Arrays in C

- Examples:
  char myarray1[3] = {2, 9, 4};
  char myarray2[4];
  int myarray3[2];

  myarray1[2] // Last element of myarray1

  Memory Address
  FF00 FF01 FF02 FF03 FF04 FF05 FF06 FF07 FF08 FF09 FF0A
  Value 0x02 0x09 0x04 ? ? ? ? ? ? ? ?
  Array myarray1 myarray2 myarray3
  Index 0 1 2 0 1 2 3 0 1

Arrays in C

- Examples:
  char myarray1[3] = {2, 9, 4};
  char myarray2[4];
  int myarray3[2];

  myarray1[8] = 0x32; (update the memory map)

  Memory Address
  FF00 FF01 FF02 FF03 FF04 FF05 FF06 FF07 FF08 FF09 FF0A
  Value 0x02 0x09 0x04 ? ? ? ? ? ? ? ?
  Array myarray1 myarray2 myarray3
  Index 0 1 2 0 1 2 3 0 1

Arrays

Array Copy Example

```c
int TestArray1[20]; // An array of 20 integers
int TestArray2[20]; // An array of 20 integers

TestArray1 = TestArray2; // This does not "copy" !!!
```

```c
for (int i = 0; i < 20; i++)
  { TestArray1[i] = TestArray2[i]; // This copies }
```
Arrays in C

- Looping through an array

```c
int myarray[5] = {1, 2, 3, 4, 5};
int x;
for(int i=0; i < 5; i++) {
    x = myarray[i];
    // do something with x
}
```

Character Strings in C

- There are no Strings in C like in Java (there are no classes)
- Strings are represented as char arrays
- char is a primitive data type
  - stores 8 bits of data, not necessarily a character
  - can be used to store small numbers
- A string of characters can be represented as a string literal by putting double quotes around the text:
  - Examples:
    ```c
    "This is a string literal."
    "123 Main Street"
    "X"
    ```

- Do not use statements like: `if (str2 == str3)` to test equality
  - Again: str1, str2, and str3 are the address of the first char in each array.
  - Use a function like strcmp to test if char arrays are equivalent

```c
char str1[] = "123";
char str2[] = "123";
if (strcmp(str1, str2) == 0) {
    // str1 matches str2
}
```

Character Strings in C

- The end of a string (char array) is signified by a null byte
  - Null bytes is a byte with a value of 0
  - String literals (i.e. "some text") have an automatic null byte included
- str1, str2, and str3 below each consume 4 bytes of memory and are equivalent in value:
  ```c
  char* str1 = "123"; // pointer, discuss next week
  char str2[] = "123";
  char str3[4] = {'1', '2', '3', 0};
  ```

- Each character is encoded in 8 bits using ASCII:
  - The following statements are equivalent:

```c
char str[] = "hi";
char str[3] = {'h', 'i', '\0'};
char str[] = {104, 105, 0};
char str[3] = {0x68, 0x69, 0x0};
```
Character Strings in C

- Examples:
  char myword1[6] = "Hello";  // declare and initialize
  char myword2[4] = "288";  // declare and initialize

Note: myword1[6] does not give room for the NULL byte.

Escape Sequences

- What if we wanted to print the quote character?
- The following line would confuse the compiler because it would interpret the second quote as the end of the string:
  ```c
  char str[] = "I said "Hello" to you.";
  ```
- An escape sequence is a series of characters that represents a special character
- An escape sequence begins with a backslash character (\)
  ```c
  char str[] = "I said \"Hello\" to you.";
  ```
### Escape Sequences

<table>
<thead>
<tr>
<th>Binary</th>
<th>Oct</th>
<th>Dec</th>
<th>Hex</th>
<th>Abbr</th>
<th>Carrot</th>
<th>Escape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000 0000</td>
<td>0 0 0 0</td>
<td>NUL</td>
<td>@</td>
<td>\0</td>
<td>Null character</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000 0111</td>
<td>7 7 7 BE</td>
<td>BEL</td>
<td>G</td>
<td>\a</td>
<td>Bell</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000 1000</td>
<td>8 8 8 BS</td>
<td>BS</td>
<td>H</td>
<td>\b</td>
<td>Backspace</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000 1001</td>
<td>8 8 8 HT</td>
<td>HT</td>
<td>I</td>
<td>\t</td>
<td>Horizontal Tab</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000 1010</td>
<td>8 8 8 LF</td>
<td>LF</td>
<td>J</td>
<td>\n</td>
<td>Line feed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000 1111</td>
<td>15 15 15 VT</td>
<td>VT</td>
<td>K</td>
<td>\v</td>
<td>Vertical Tab</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001 1000</td>
<td>16 16 16 FF</td>
<td>FF</td>
<td>L</td>
<td>\f</td>
<td>Form feed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010 0000</td>
<td>12 12 12 DC</td>
<td>DC</td>
<td>M</td>
<td>\r</td>
<td>Carriage return</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010 0001</td>
<td>12 12 12 ESC</td>
<td>ESC</td>
<td>N</td>
<td>\e</td>
<td>Escape</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010 0010</td>
<td>12 12 12 \</td>
<td>\</td>
<td>\</td>
<td>Single Quote</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>011 1111</td>
<td>31 31 31 &quot;</td>
<td>&quot;</td>
<td>0D</td>
<td>%</td>
<td>Double Quote</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Multiline String Literals

- The compiler will concatenate string literals that are only separated by white space.
- The following are equivalent expressions:
  - `char str[] = "hello world";`
  - `char str[] = "hello " "world";`
  - `char str[] = "hello world";`
- If you need to concatenate string variables, use a function from the standard library like `strcat` by including `<string.h>` or `sprintf` by including `<stdio.h>`.

### Formatting Strings

- `printf`, `sprintf`, `fprintf` = standard library functions for printing data into char arrays
- Must include `stdio.h` in order to use these function
  ```
  #include <stdio.h>
  ```
- These functions have an argument called a formatter string that accepts % escaped variables
- Review the documentation on functionality of `sprintf`
  - [Google “sprintf”](http://www.cplusplus.com/reference/clibrary/cstdio/sprintf/), first result is:
- TAs will review basic string manipulation functions in Lab

### Formatting Strings: Example % formats

```c
int age = 18;
int course = 288;
char message[] = "Hello World";
char short_msg[5] = {'H', 'I'};

printf("My age is %d", age);
// gives: My age is 18

printf("Say %s my age is %d", message, age);
// gives: Say Hello World my age is 18

printf("Hi is spelled %c %c, in class %d", short_msg[0], short_msg[1], course);
// gives: Hi is spelled H I, in class 288
```

### Formatting Strings: Example % formats

- See: Table 7-1 of the “The C Programming Language”
  - Also can be found in many places on the Internet
- [http://class.ece.iastate.edu/cpre288](http://class.ece.iastate.edu/cpre288)

---

**STRING MANIPULATION**
String Manipulation Functions
• int sprintf(char * str, const char * format, ...);
• int strlen(const char * str);
• int strncmp(const char * str1, const char * str2, size_t num);

String Manipulation Functions: sprintf
int sprintf(char * str, const char * format, ...);
Param1: location to store the string (e.g. character array)
Param2: formatted string to store in the array
Param3-n: formatting variables that appear in the formatted string.
Example:
int class_num = 288;
char my_array[20];
char another_array[10] = “Goodbye”
sprintf(my_array, “Hello CPRE %d
”, class_num);
// my_array now contains: Hello CPRE 288
printf(“%s, another_array); // prints Goodbye

String Manipulation Functions: strlen
int strlen(const char * str);
Param1: location of a string (e.g. character array name)
Return value: returns the length of the string (not counting NULL byte).
Example:
char my_array[20] = “Hello CPRE288”;
int my_len = 0;
my_len = strlen(my_array);
// my_len now has a value of 13

String Manipulation Functions: strcmp
int strcmp(const char * str1, const char * str2);
Param1: location of a string
Param2: location of a string
Return value: if equal then 0, if the first position that does not match
is greater in str1 then +, else -. 
Example:
char my_array1[20] = “apple”;
char my_array2[20] = “pair”;
int my_compare = 0;
my_compare = strcmp(my_array1, my_array2);
// ‘a’ has a lower value than ‘p’, so my_compare will be negative

Class Activity
• Predict the value of message after each line:
  char str1 = “hello”;
  char str2 = “world”;
  char message[100];
sprintf(message, “The meaning of life is %d.”, 42);
The meaning of life is 42.
sprintf(message, “The meaning of life is %s.”, str1);
The meaning of life is hello.
sprintf(message, “%s %s”, str1, str2);
hello world
sprintf(message, “%s %s”, str1+1, str2+3 );
ello ld
LAB 1 QUICK OVERVIEW

Lab 1: Introduction to the Platform

Purpose: Introduction to the AVR Studio 6 and VORTEX Platform
- AVR Studio 6: The integrated development environment (IDE) for Atmel AVR platforms
- VORTEX: An integrated hardware platform of iRobot Create and Cerebot II microcontroller board

AVR Studio 6
An IDE from Atmel for AVR platforms
- Source code editing
- Compiling building
- Download binary to boards
- Debug
- Simulation

Lab 1
Lab 1: Introduction to the AVR Studio 6
- Part 1 "Hello, world"
  - Build, download, and execute
- Part 2 Simulated Environment
- Part 3 Rotating Banner
  - The message has 34 characters and the LCD can only show 20 characters per line at a time

Programming Example
How to display a rotating banner?
A smaller example: 10-char. display, 19-char. message
The screen Message
Shift for one character every one second

Programming Example
What's the desired program behavior?
First display "Welcome to " and wait
Then display "e"come to C" and wait
Then display "come to CPr" and wait
Then display "come to CPR" and wait and so on
Programming Example

Give a general but precise description
First show characters 0-9 and wait
Then show characters 1-10 and wait
Then show characters 2-11 and wait
Then show characters 3-12 and wait
and so on

Programming Example

Describe program’s behavior
set starting position at 0
loop forever
clear the screen
display 10 chars from the starting pos.
shift the starting pos. to the next position
wait for one second
end loop

Lab 1 Programming Exercise

Part 3. Rotating Banner
Show “Microcontrollers are loads of fun!” in a rotating style
- The message has 34 characters and the LCD line has 20
- Shift in first 20 characters one by one, with 0.3 second delay
- Start to rotate and continue till the last character is shown, with 0.5 second delay
- Continue rotating until the last character is shown, with 0.5 second delay
- Repeat this procedure

Lab 1 Programming Exercise

Idea 1: A forever loop of three phases
Phase 1: Shift in the first 20 characters
Phase 2: Rotate until the last character is displayed
Phase 3: Rotate until the last character is shifted out

Lab 1 Programming Exercise

First, have a function to print the banner for one time
void print_banner(char *msg, int start, int end);

This makes the rest of programming easier

Lab 1 Programming Exercise

Some details to take care
“display 10 chars from the starting pos.”

“shift the starting pos. to the next position”
Lab 1 Programming Exercise

```c
int main()
{
    while (1)
    {
        for (...) // Phase 1
        for (...) // Phase 2
        ... // Phase 3
    }
}
```

http://class.ece.iastate.edu/cpre288