CprE 288 – Introduction to Embedded Systems
(C for Embedded Systems: Variables, Arrays, and Strings)

Instructors:
Dr. Phillip Jones
Overview

• Announcements
• Some Basics
• Simplest Embedded C program
• Variables
• Arrays
• C-strings
Announcements

• HW 1: Due Sunday (5/22)
• Quiz 1: Friday (5/20) – First 10 minutes of class using Canvas
  – Material covered during first week of class
• Lab 1: LCD Banner
  – Lab attendance is mandatory
C IN EMBEDDED SYSTEMS
Methods for Representing Data

- **Bit**
  - 1 (True)
  - 0 (False)
- **Nibble** (less commonly used)
  - 4 bits
- **Byte**
  - 8 bits
- **N-byte Words**
  - 2-byte / 16-bit Word, 4-byte / 32-bit Word
Methods for Representing Data

• Three of the most common forms of notation
  – Decimal (base 10) 0123456789
  – Hexadecimal (base 16) 0123456789ABCDEF
  – Binary (base 2) 01

• Converting between forms
  – When converting binary to hexadecimal, every group of 4 bits (nibble) represents a hexadecimal digit
  – Examples:

    | Binary | Hexadecimal |
    |--------|-------------|
    | 0010   | 2           |
    | 0100   | 4           |
    | 1010   | A           |
Methods for Representing Data

• Memorize conversion between decimal (0-15), hex (0x0 – 0xF), and binary (0000 – 1111).
  
  – This will be an important survival skill in Embedded Systems, similar to knowing your time tables up to 10x10 is important if you want to survive engineering.

<table>
<thead>
<tr>
<th>Decimal (Dec)</th>
<th>Hexadecimal (Hex)</th>
<th>Binary (Bin)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x0</td>
<td>0b00000</td>
</tr>
<tr>
<td>1</td>
<td>0x1</td>
<td>0b0001</td>
</tr>
<tr>
<td>2</td>
<td>0x2</td>
<td>0b0010</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>14</td>
<td>0xE</td>
<td>0b1110</td>
</tr>
<tr>
<td>15</td>
<td>0xF</td>
<td>0b1111</td>
</tr>
</tbody>
</table>
Base Conversion (by hand)

• Base n to base 10

Problem: Convert base 2: 0b0100_1011, to base 10

Solution:

<table>
<thead>
<tr>
<th>$2^7$</th>
<th>$2^6$</th>
<th>$2^5$</th>
<th>$2^4$</th>
<th>$2^3$</th>
<th>$2^2$</th>
<th>$2^1$</th>
<th>$2^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>128’s</td>
<td>64’s</td>
<td>32’s</td>
<td>16’s</td>
<td>8’s</td>
<td>4’s</td>
<td>2’s</td>
<td>1’s</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$64 + 8 + 2 + 1 = 75$
Base Conversion (by hand)

- Base 10 to base \( n \)

Problem: Convert 175 to base 16

Solution:
Create a table of the columns in a base 16 number and subtract from the original number:

\[
\begin{array}{c|c}
16^1 & 16^0 \\
16's & 1's \\
A & A \\
\end{array}
\]

\[
175 - 160 = 15
\]

\[
\begin{array}{c|c}
16^1 & 16^0 \\
16's & 1's \\
A & F \\
\end{array}
\]
• Syntax in C
  – Computers understand binary
  – The following lines of code are all the same (the compiler does not care what base the programmer uses):

char x = 2 + 1;
char x = 0b10 + 1;
char x = 0x2 + 1;
char x = 0x02 + 0x01;
void main()
{
    while (1)
    {
        // do forever...
    }

• Embedded programs often run forever
Hello World!

```c
#include <stdio.h>

void main()
{
    printf("hello, world\n");
}
```
Microcontroller / System-on-Chip (SoC)

- Program Memory → CPU
- CPU → Data Memory
- Data Memory → Outside World

Interrupts

- NVIC

Devices

- ADC
  - CFG | DATA | STATUS
- UART
  - CFG | DATA | STATUS
- Timers
  - CFG | DATA | STATUS

GPIO

- GPIO_DATA
- Port X (8-bits): 7 6 5 4 3 2 1 0

http://class.ece.iastate.edu/cpre288
VARIABLES IN C
Variables

- Variables are the primary mechanism for storing data to be processed by your program

- Examples:
  - area, graph, distance, file1, file2, height, wheel_right

- Must not be a reserved keyword (next slide)

- Good practice: use descriptive variable names
  - Good names: height, input_file, area
  - Bad names: h, if, a

- Rule of thumb: Always code as though the person maintaining your code knows where you sleep... and has anger management issues.
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

- Variables must be *declared* by specifying the variable's *name* and the *type* of information that it will hold

```c
int total;
int count, temp, result;
```

*Multiple variables can be created in one declaration*
A variable can be given an initial value in the declaration

```c
int sum = 0;
int base = 32, max = 149;
```

If no initial value is given, do not assume the default value is 0

```c
int k, i;

for (i = 0; i < 10; i++)
{
    k = k + 1;
}

printf(“%d”, k);
```
## Primitive Types and Sizes

<table>
<thead>
<tr>
<th>Name</th>
<th>Number of Bytes</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td>0 to 255 or -128 to 127 (Depends on Compiler settings)</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 TM4C123)</td>
<td>4</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>unsigned int (on TM4C123)</td>
<td>4</td>
<td>0 to 4,294,967,295</td>
</tr>
<tr>
<td>(pointer)</td>
<td>Varies by platform</td>
<td>Varies by platform</td>
</tr>
<tr>
<td>(pointer on TM4C123)</td>
<td>4</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)
- Note: char does not have a standard default, depends on Compiler settings
### Primitive Types and Sizes

<table>
<thead>
<tr>
<th>Name</th>
<th>Number of Bytes</th>
<th><code>sizeof()</code></th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>4</td>
<td></td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>signed long</td>
<td>4</td>
<td></td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>unsigned long</td>
<td>4</td>
<td></td>
<td>0 to 4,294,967,295</td>
</tr>
<tr>
<td>long long</td>
<td>8</td>
<td></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></td>
<td>±1.175e-38 to ±3.402e38</td>
</tr>
<tr>
<td>double</td>
<td>Varies by platform</td>
<td></td>
<td></td>
</tr>
<tr>
<td>double (on TM4C123)</td>
<td>8</td>
<td></td>
<td>±2.3E-308 to ±1.7E+308</td>
</tr>
</tbody>
</table>

- Primitive types in C: char, short, int, long, float, double
- Default modifier on primitive types is **signed** (not unsigned)
- Note: char does not have a standard default, depends on Compiler settings
Variables: Size

```c
char     sum_char    = 0;
short    sum_short   = 0;
int      sum_int     = 0;
```

- **sum_char** value is a 8-bit value:
  - Binary: 0b0000 0000
  - Hex: 0x00

- **sum_short** value is a 16-bit value:
  - Binary: 0b0000 0000 0000 0000
  - Hex: 0x0000

- **sum_int** value is a 32-bit value:
  - 0b0000 0000 0000 0000 0000 0000 0000 0000
  - Hex: 0x0000 0000
### 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
  - Decimal: 

[Link to course page](http://class.ece.iastate.edu/cpre288)
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.
    // the C compiler will truncate to 8-bits
  - Decimal:
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: 0b0000 0001
  – Decimal: 1
ARRAYS IN C
Arrays in C

- Sequence of a specific variable type stored in memory
- **Zero-indexed** (starts at zero rather than one)
- Define an array as
  
  ```
  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
  ```
  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

http://class.ece.iastate.edu/cpre288
Arrays in C

• Examples:

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

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

// allocates memory for 2 short's
short myarray3[2];
```
Arrays in C

- Examples:

```c
char  myarray1[3] = {2, 9, 4};
char  myarray2[4];
short myarray3[2];
```

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>FF00</th>
<th>FF01</th>
<th>FF02</th>
<th>FF03</th>
<th>FF04</th>
<th>FF05</th>
<th>FF06</th>
<th>FF07</th>
<th>FF08</th>
<th>FF09</th>
<th>FF0A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0x02</td>
<td>0x09</td>
<td>0x04</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Array</td>
<td>myarray1</td>
<td></td>
<td></td>
<td>myarray2</td>
<td></td>
<td></td>
<td>myarray3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Arrays in C

• You do not have to specify the size if the array is being initialized during the declaration
  
  ```
  char myarray1[] = {2, 9, 4};
  char myarray2[4];
  short myarray3[2];
  ```

• When defining an array, the array name is the address in memory for the first element of the array
  
  • `myarray3 == ??`

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>FF00</th>
<th>FF01</th>
<th>FF02</th>
<th>FF03</th>
<th>FF04</th>
<th>FF05</th>
<th>FF06</th>
<th>FF07</th>
<th>FF08</th>
<th>FF09</th>
<th>FF0A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0x02</td>
<td>0x09</td>
<td>0x04</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Array</td>
<td>myarray1</td>
<td>myarray2</td>
<td>myarray3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Arrays in C

- You do not have to specify the size if the array is being initialized during the declaration
  
  ```c
  char myarray1[] = {2, 9, 4};
  char myarray2[4];
  short myarray3[2];
  ```

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

  - `myarray3 == 0xFF07`

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>FF00</th>
<th>FF01</th>
<th>FF02</th>
<th>FF03</th>
<th>FF04</th>
<th>FF05</th>
<th>FF06</th>
<th>FF07</th>
<th>FF08</th>
<th>FF09</th>
<th>FF0A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0x02</td>
<td>0x09</td>
<td>0x04</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Array</td>
<td>myarray1</td>
<td>myarray2</td>
<td>myarray3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Arrays

• Be careful of boundaries in C
  – No guard to prevent you from accessing beyond array end
  – Write beyond array => Potential for disaster
• No built-in mechanism for copying arrays
Arrays in C

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

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>FF00</th>
<th>FF01</th>
<th>FF02</th>
<th>FF03</th>
<th>FF04</th>
<th>FF05</th>
<th>FF06</th>
<th>FF07</th>
<th>FF08</th>
<th>FF09</th>
<th>FF0A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0x02</td>
<td>0x09</td>
<td>0x04</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Array</td>
<td>myarray1</td>
<td>myarray2</td>
<td>myarray3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Arrays in C

• Examples:
  
  ```c
  char  myarray1[3] = {2, 9, 4};
  char  myarray2[4];
  short myarray3[2];
  ```

  ```plaintext
  myarray1[0]  // First element of myarray1
  ```
Arrays in C

- Examples:

```c
char myarray1[3] = {2, 9, 4};
char myarray2[4];
short myarray3[2];
```

```plaintext
myarray1[2] // Last element of myarray1
```

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>FF00</th>
<th>FF01</th>
<th>FF02</th>
<th>FF03</th>
<th>FF04</th>
<th>FF05</th>
<th>FF06</th>
<th>FF07</th>
<th>FF08</th>
<th>FF09</th>
<th>FF0A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0x02</td>
<td>0x09</td>
<td>0x04</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Array</td>
<td>myarray1</td>
<td>myarray2</td>
<td>myarray3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

http://class.ece.iastate.edu/cpre288
Arrays in C

• Examples:

```c
char myarray1[3] = {2, 9, 4};
char myarray2[4];
short myarray3[2];
```

myarray1[3]  // Passed end of myarray1!!!
// Overwrote myarray2!!
Arrays in C

• Examples:
  
  ```c
  char  myarray1[3] = {2, 9, 4};
  char  myarray2[4];
  short myarray3[2];
  ```

  `myarray1[8] = 0x32;` (update the memory map)
Arrays in C

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

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

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>FF00</th>
<th>FF01</th>
<th>FF02</th>
<th>FF03</th>
<th>FF04</th>
<th>FF05</th>
<th>FF06</th>
<th>FF07</th>
<th>FF08</th>
<th>FF09</th>
<th>FF0A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0x02</td>
<td>0x09</td>
<td>0x04</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>0x32</td>
<td>?</td>
</tr>
<tr>
<td>Array</td>
<td>myarray1</td>
<td>myarray2</td>
<td>myarray3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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” !!!

for (int i = 0; i < 20; i++)
{
    TestArray1[i] = TestArray2[i];  // This copies
}
```
STRINGS IN C
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:

```
"This is a string literal."
"123 Main Street"
"X"
```
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:

  char* str1 = "123";  // pointer, discuss next week
  char str2[] = "123";
  char str3[4] = {'1', '2', '3', 0};
Character Strings in C

• 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[3] = { 104, 105, 0 };
Char str[3] = {0x68, 0x69, 0x0};
```
<table>
<thead>
<tr>
<th>Binary</th>
<th>Oct Dec</th>
<th>Hex</th>
<th>Glyph</th>
</tr>
</thead>
<tbody>
<tr>
<td>010 0000</td>
<td>040</td>
<td>20</td>
<td>$^{sp}$</td>
</tr>
<tr>
<td>010 0001</td>
<td>041</td>
<td>21</td>
<td>!</td>
</tr>
<tr>
<td>010 0010</td>
<td>042</td>
<td>22</td>
<td>&quot;</td>
</tr>
<tr>
<td>010 0011</td>
<td>043</td>
<td>23</td>
<td>#</td>
</tr>
<tr>
<td>010 0100</td>
<td>044</td>
<td>24</td>
<td>$</td>
</tr>
<tr>
<td>010 0101</td>
<td>045</td>
<td>25</td>
<td>%</td>
</tr>
<tr>
<td>010 0110</td>
<td>046</td>
<td>26</td>
<td>&amp;</td>
</tr>
<tr>
<td>010 0111</td>
<td>047</td>
<td>27</td>
<td>'</td>
</tr>
<tr>
<td>010 1000</td>
<td>050</td>
<td>28</td>
<td>(</td>
</tr>
<tr>
<td>010 1001</td>
<td>051</td>
<td>29</td>
<td>)</td>
</tr>
<tr>
<td>010 1010</td>
<td>052</td>
<td>2A</td>
<td>+</td>
</tr>
<tr>
<td>010 1011</td>
<td>053</td>
<td>2B</td>
<td>-</td>
</tr>
<tr>
<td>010 1100</td>
<td>054</td>
<td>2C</td>
<td>+</td>
</tr>
<tr>
<td>010 1101</td>
<td>055</td>
<td>2D</td>
<td>-</td>
</tr>
<tr>
<td>011 0000</td>
<td>060</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>011 0001</td>
<td>061</td>
<td>31</td>
<td>1</td>
</tr>
<tr>
<td>011 0010</td>
<td>062</td>
<td>32</td>
<td>2</td>
</tr>
<tr>
<td>011 0011</td>
<td>063</td>
<td>33</td>
<td>3</td>
</tr>
<tr>
<td>011 0100</td>
<td>064</td>
<td>34</td>
<td>4</td>
</tr>
<tr>
<td>011 0101</td>
<td>065</td>
<td>35</td>
<td>5</td>
</tr>
<tr>
<td>011 0110</td>
<td>066</td>
<td>36</td>
<td>6</td>
</tr>
<tr>
<td>011 0111</td>
<td>067</td>
<td>37</td>
<td>7</td>
</tr>
<tr>
<td>011 1000</td>
<td>070</td>
<td>38</td>
<td>8</td>
</tr>
<tr>
<td>011 1001</td>
<td>071</td>
<td>39</td>
<td>9</td>
</tr>
<tr>
<td>011 1010</td>
<td>072</td>
<td>3A</td>
<td>:</td>
</tr>
<tr>
<td>011 1011</td>
<td>073</td>
<td>3B</td>
<td>;</td>
</tr>
<tr>
<td>011 1100</td>
<td>074</td>
<td>3C</td>
<td>&lt;</td>
</tr>
<tr>
<td>011 1101</td>
<td>075</td>
<td>3D</td>
<td>=</td>
</tr>
<tr>
<td>011 1110</td>
<td>076</td>
<td>3E</td>
<td>&gt;</td>
</tr>
<tr>
<td>011 1111</td>
<td>077</td>
<td>3F</td>
<td>?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Binary</th>
<th>Oct Dec</th>
<th>Hex</th>
<th>Glyph</th>
</tr>
</thead>
<tbody>
<tr>
<td>110 0000</td>
<td>140</td>
<td>60</td>
<td>`</td>
</tr>
<tr>
<td>110 0001</td>
<td>141</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>110 0010</td>
<td>142</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>110 0011</td>
<td>143</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>110 0100</td>
<td>144</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>110 0101</td>
<td>145</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>110 0110</td>
<td>146</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>110 0111</td>
<td>147</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>110 1000</td>
<td>150</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>110 1001</td>
<td>151</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>110 1010</td>
<td>152</td>
<td>6A</td>
<td>j</td>
</tr>
<tr>
<td>110 1011</td>
<td>153</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>110 1100</td>
<td>154</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>110 1101</td>
<td>155</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>110 1110</td>
<td>156</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>110 1111</td>
<td>157</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>111 0000</td>
<td>160</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>111 0001</td>
<td>161</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>111 0010</td>
<td>162</td>
<td>72</td>
<td>r</td>
</tr>
<tr>
<td>111 0011</td>
<td>163</td>
<td>73</td>
<td>s</td>
</tr>
<tr>
<td>111 0100</td>
<td>164</td>
<td>74</td>
<td>t</td>
</tr>
<tr>
<td>111 0101</td>
<td>165</td>
<td>75</td>
<td>u</td>
</tr>
<tr>
<td>111 0110</td>
<td>166</td>
<td>76</td>
<td>v</td>
</tr>
<tr>
<td>111 0111</td>
<td>167</td>
<td>77</td>
<td>w</td>
</tr>
<tr>
<td>111 1000</td>
<td>170</td>
<td>78</td>
<td>x</td>
</tr>
<tr>
<td>111 1001</td>
<td>171</td>
<td>79</td>
<td>y</td>
</tr>
<tr>
<td>111 1010</td>
<td>172</td>
<td>7A</td>
<td>z</td>
</tr>
<tr>
<td>111 1011</td>
<td>173</td>
<td>7B</td>
<td>{</td>
</tr>
<tr>
<td>111 1100</td>
<td>174</td>
<td>7C</td>
<td></td>
</tr>
<tr>
<td>111 1101</td>
<td>175</td>
<td>7D</td>
<td>}</td>
</tr>
<tr>
<td>111 1110</td>
<td>176</td>
<td>7E</td>
<td>~</td>
</tr>
</tbody>
</table>
Character Strings in C

• Examples:

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

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>DF00</th>
<th>DF01</th>
<th>DF02</th>
<th>DF03</th>
<th>DF04</th>
<th>DF05</th>
<th>DF06</th>
<th>DF07</th>
<th>DF08</th>
<th>DF09</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>‘H’</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Array</td>
<td>myword1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>myword2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

http://class.ece.iastate.edu/cpre288
## Character Strings in C

### Examples:

```c
char myword1[6] = "Hello";  // declare and initialize
char myword2[4]   = "288";  // declare and initialize
```

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>DF00</th>
<th>DF01</th>
<th>DF02</th>
<th>DF03</th>
<th>DF04</th>
<th>DF05</th>
<th>DF06</th>
<th>DF07</th>
<th>DF08</th>
<th>DF09</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>‘H’</td>
<td>‘e’</td>
<td>‘l’</td>
<td>‘l’</td>
<td>‘o’</td>
<td>‘\0’</td>
<td>‘2’</td>
<td>‘8’</td>
<td>‘8’</td>
<td>‘\0’</td>
</tr>
<tr>
<td>Array</td>
<td>myword1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Source: [http://class.ece.iastate.edu/cpre288](http://class.ece.iastate.edu/cpre288)
Character Strings in C

• Examples:

```c
char myword1[6] = "Helloo";  // declare and initialize
char myword2[4] = "288";    // declare and initialize
```

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>DF00</th>
<th>DF01</th>
<th>DF02</th>
<th>DF03</th>
<th>DF04</th>
<th>DF05</th>
<th>DF06</th>
<th>DF07</th>
<th>DF08</th>
<th>DF09</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>‘H’</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Array</td>
<td>myword1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

http://class.ece.iastate.edu/cpre288
Character Strings in C

• Examples:

```c
char myword1[6] = "Helloo"; // 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</td>
<td>0</td>
<td>0</td>
<td>NUL</td>
<td>^@</td>
<td>\0</td>
<td>Null character</td>
</tr>
<tr>
<td>000 0111</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>BEL</td>
<td>^G</td>
<td>\a</td>
<td>Bell</td>
</tr>
<tr>
<td>000 1000</td>
<td>10</td>
<td>8</td>
<td>8</td>
<td>BS</td>
<td>^H</td>
<td>\b</td>
<td>Backspace</td>
</tr>
<tr>
<td>000 1001</td>
<td>11</td>
<td>9</td>
<td>9</td>
<td>HT</td>
<td>^I</td>
<td>\t</td>
<td>Horizontal Tab</td>
</tr>
<tr>
<td>000 1010</td>
<td>12</td>
<td>10</td>
<td>0A</td>
<td>LF</td>
<td>^J</td>
<td>\n</td>
<td>Line feed</td>
</tr>
<tr>
<td>000 1011</td>
<td>13</td>
<td>11</td>
<td>0B</td>
<td>VT</td>
<td>^K</td>
<td>\v</td>
<td>Vertical Tab</td>
</tr>
<tr>
<td>000 1100</td>
<td>14</td>
<td>12</td>
<td>0C</td>
<td>FF</td>
<td>^L</td>
<td>\f</td>
<td>Form feed</td>
</tr>
<tr>
<td>000 1101</td>
<td>15</td>
<td>13</td>
<td>0D</td>
<td>CR</td>
<td>^M</td>
<td>\r</td>
<td>Carriage return</td>
</tr>
<tr>
<td>001 1011</td>
<td>33</td>
<td>27</td>
<td>1B</td>
<td>ESC</td>
<td>^[</td>
<td>\e</td>
<td>Escape</td>
</tr>
<tr>
<td>010 0111</td>
<td>47</td>
<td>39</td>
<td>27</td>
<td>'</td>
<td></td>
<td>'</td>
<td>Single Quote</td>
</tr>
<tr>
<td>010 0010</td>
<td>42</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
<td></td>
<td>&quot;</td>
<td>Double Quote</td>
</tr>
<tr>
<td>101 1100</td>
<td>134</td>
<td>92</td>
<td>5C</td>
<td>\</td>
<td></td>
<td>\</td>
<td>Backslash</td>
</tr>
</tbody>
</table>
Formatting Strings

• `printf`, `sprintf`, `fprintf` = standard library functions for printing data into char arrays

• Must include `stdio.h` in order to use these functions
  ```
  #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”, first result is:
    ```
    http://www.cplusplus.com/reference/clibrary/cstdio/sprintf/
    ```

• TAs will review basic string manipulation functions in Lab
Formatting Strings: Example % formats

- See: Table 7-1 of the “The C Programming Language”
  - Also can be found in many places on the Internet

<table>
<thead>
<tr>
<th>Character</th>
<th>Argument type; Printed As</th>
</tr>
</thead>
<tbody>
<tr>
<td>d, i</td>
<td>int; decimal number</td>
</tr>
<tr>
<td>o</td>
<td>int; unsigned octal number (without a leading zero)</td>
</tr>
<tr>
<td>x, X</td>
<td>int; unsigned hexadecimal number (without a leading 0x or 0X), using abcdef or ABCDEF for 10, ..., 15.</td>
</tr>
<tr>
<td>u</td>
<td>int; unsigned decimal number</td>
</tr>
<tr>
<td>c</td>
<td>int; single character</td>
</tr>
<tr>
<td>s</td>
<td>char *; print characters from the string until a ‘\0’ or the number of characters given by the precision.</td>
</tr>
<tr>
<td>f</td>
<td>double; [-]m.ddd, where the number of d’s is given by the precision (default 6).</td>
</tr>
<tr>
<td>e, E</td>
<td>double; [-]m.dddde+/-xx or [-]m.ddddeE+/-xx, where the number of d’s is given by the precision (default 6).</td>
</tr>
<tr>
<td>g, G</td>
<td>double; use %e or %E if the exponent is less than -4 or greater than or equal to the precision; otherwise use %f. Trailing zeros and a trailing decimal point are not printed.</td>
</tr>
<tr>
<td>p</td>
<td>void *; pointer (implementation-dependent representation).</td>
</tr>
<tr>
<td>%</td>
<td>no argument is converted; print a %</td>
</tr>
</tbody>
</table>
int age = 18;
int course = 288;
char message[] = “Hello World”;

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
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`

```c
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:**

```c
int class_num = 288;
char my_array[20];
char another_array[10] = "Goodbye"
sprintf(my_array, "Hello CPRE %d \n", class_num);
// my_array now contains: Hello CPRE 288
printf("%s", another_array); // prints Goodbye
```
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[10];
char another_array[10] = “Goodbye”
sprintf(my_array, “Hello CPRE %d”, class_num);
printf(“%s”, another_array); //??
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
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
• Predict the value of message after each line:
char str1[] = “hello”;
char str2[] = “world”;
char message[100];

`printf(message, “The meaning of life is %d.”, 42);`
The meaning of life is 42.

`printf(message, “The meaning of life is %s.”, str1);`
The meaning of life is hello.

`printf(message, “%s %s”, str1, str2);`
hello world

`printf(message, “%s %s”, str1+1, str2+3 );`
ello ld
STRUCTs in C
struct

- Chapter 6.1 - 6.4 (C programming Language)
- Chapter 6.9: Bit-fields (C programming Language)
The struct type allows a programmer to define a compound data type.

The size of a struct is the size of its components added together.

```c
struct RGB
{
    char red;
    char green;
    char blue;
};

struct RGB my_color;
my_color.blue = 255;

struct RGB *my_color_ptr = &my_color;

(*my_color_ptr).blue = 255;
my_color_ptr->blue = 255;  // equivalent to previous line
```
struct student
{
    char name[30];
    int ISUID;
};

struct student student_records[100];

// Set student ISUID at index 10
student_records[10].ISUID = 5678;
struct

struct sensor
{
    float distance;
    unsigned char bumpLeft;
    unsigned char bumpRight;
};

struct sensor my_sensor;
struct sensor *my_ptr = &my_sensor;

float my_distance;

// Access Distance
my_distance = my_sensor.distance;
my_distance = my_ptr->distance
union

- Chapter 6.8: C programming language
Union: Merge multiple components

```c
union u_tag {
    int ival;    // size 4 bytes
    short sval;  // size 2 bytes
    float fval;  // size 4 bytes
};
```

The size of a union variable is the size of its maximum component.
Union: Merge multiple components

```c
union u_tag {
    int ival;     // size 4 bytes
    short sval;  // size 2 bytes
    float fval;  // size 4 bytes
};
```

The size of a union variable is the size of its maximum component.

This example the size is 4, since the largest component is 4 bytes.
Use of union inside of a struct

```c
struct {
    char *name;
    int flags;
    short s_type;
    union {
        short val;
        float fval;
        char cval;
    } u;
} symtab;
```

How large is the struct symtab?
Use of union inside of a struct

```c
struct {
    char *name; 4
    int flags; 4
    short s_type; 2
    union {
        short val; 2
        float fval; 4 //largest member of union u
        char cval; 1
    } u; //largest member defines a union's size
} symtab;
```

Just sum the size of each struct member. symtab size is: 4+4+2+4 = 14 bytes
typedef examples

typedef unsigned char uint8_t;

typedef struct RGB{
    uint8_t red;
    uint8_t green;
    uint8_t blue;
} RGB_t;

RGB_t my_color;
my_color.blue = 255;
typedef examples

typedef unsigned char uint8_t;

typedef struct RGB{
    uint8_t red;
    uint8_t green;
    uint8_t blue;
} RGB_t;

// Array of RGB_t’s
RGB_t my_color[10]; // An array of 10 RGB_t
my_color[5].blue = 255; // set blue of 6^{th} RGB_t