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

Instructors:
Dr. Phillip Jones
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
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, ...
- TIOBE Index:
INTRO TO C
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

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

- Most embedded programs run forever
#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
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
Which of the follow are valid C statements?

```c
a = a + b;
a;
a + b;
10 + 20;
a = (b = c);
;
```
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 IN C
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*
Variables

• 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
• Exception: names of iterators in loops
  – Common names for iterators: i, j, k, x, y, z
• 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

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

```java
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

```c
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</th>
<th>sizeof()</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td></td>
<td>-128 to 127</td>
</tr>
<tr>
<td>signed char</td>
<td>1</td>
<td></td>
<td>-128 to 127</td>
</tr>
<tr>
<td>unsigned char</td>
<td>1</td>
<td></td>
<td>0 to 255</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td></td>
<td>-32,768 to 32,767</td>
</tr>
<tr>
<td>unsigned short</td>
<td>2</td>
<td></td>
<td>0 to 65,535</td>
</tr>
<tr>
<td>int</td>
<td>Varies by platform</td>
<td></td>
<td>Varies by platform</td>
</tr>
<tr>
<td>int (on ATmega 128)</td>
<td>2</td>
<td></td>
<td>-32,768 to 32,767</td>
</tr>
<tr>
<td>(pointer)</td>
<td>Varies by platform</td>
<td></td>
<td>Varies by platform</td>
</tr>
<tr>
<td>(pointer on ATmega 128)</td>
<td>2</td>
<td></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)
## Primitive Types and Sizes

### Table: 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>4</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>signed long</td>
<td>4</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>4</td>
<td>0 to 4,294,967,295</td>
</tr>
<tr>
<td>long long</td>
<td>8</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>4</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 ATmega 128)</td>
<td>4</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

cchar  sum_char  = 0;
int    sum_int   = 0;
long   sum_long  = 0;

- **sum_char** value is a 8-bit value:
  - Binary: 0b0000 0000
  - Hex: 0x00
- **sum_int** value is a 16-bit value:
  - Binary: 0b0000 0000 0000 0000
  - Hex: 0x0000
- **sum_long** value is a 32-bit value:
  - 0b0000 0000 0000 0000 0000 0000 0000 0000
  - Hex: 0x0000 0000
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.
  // the C compiler will truncate to 8-bits
  - 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: 0b0000 0001
  - Decimal: 1
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
  
  \text{Type VariableName [ArraySize];}
  
  Example: \text{int my_array[100]}

  \textbf{Size: i.e. Number of elements}

  \textbf{data type} \hspace{1cm} \textbf{variable name}

• Last element is found at $N-1$ location
• Curly brackets can be used to initialize the array
Arrays in C

• Examples:

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

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

// allocates memory for 2 ints's
int myarray3[2];
## Arrays in C

**Examples:**

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

• 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 == ??`

<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};
```

• 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**

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

- Examples:
  
  ```c
  char myarray1[3] = {2, 9, 4};
  char myarray2[4];
  int 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 1 2</td>
<td>0 1 2</td>
<td>3 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Arrays in C

- Examples:

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

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

• Examples:

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

```
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>
Arrays in C

• Examples:

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

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

• Examples:

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

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

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

- **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
}
```
• 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</th>
<th>Dec</th>
<th>Hex</th>
<th>Glyph</th>
</tr>
</thead>
<tbody>
<tr>
<td>010 0000</td>
<td>040</td>
<td>32</td>
<td>20</td>
<td>^p</td>
</tr>
<tr>
<td>010 0001</td>
<td>041</td>
<td>33</td>
<td>21</td>
<td>!</td>
</tr>
<tr>
<td>010 0010</td>
<td>042</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
</tr>
<tr>
<td>010 0011</td>
<td>043</td>
<td>35</td>
<td>23</td>
<td>#</td>
</tr>
<tr>
<td>010 0100</td>
<td>044</td>
<td>36</td>
<td>24</td>
<td>$</td>
</tr>
<tr>
<td>010 0101</td>
<td>045</td>
<td>37</td>
<td>25</td>
<td>%</td>
</tr>
<tr>
<td>010 0110</td>
<td>046</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
</tr>
<tr>
<td>010 0111</td>
<td>047</td>
<td>39</td>
<td>27</td>
<td>'</td>
</tr>
<tr>
<td>010 1000</td>
<td>050</td>
<td>40</td>
<td>28</td>
<td>(</td>
</tr>
<tr>
<td>010 1001</td>
<td>051</td>
<td>41</td>
<td>29</td>
<td>)</td>
</tr>
<tr>
<td>010 1010</td>
<td>052</td>
<td>42</td>
<td>2A</td>
<td>*</td>
</tr>
<tr>
<td>010 1011</td>
<td>053</td>
<td>43</td>
<td>2B</td>
<td>+</td>
</tr>
<tr>
<td>010 1100</td>
<td>054</td>
<td>44</td>
<td>2C</td>
<td>.</td>
</tr>
<tr>
<td>010 1101</td>
<td>055</td>
<td>45</td>
<td>2D</td>
<td>-</td>
</tr>
<tr>
<td>010 1110</td>
<td>056</td>
<td>46</td>
<td>2E</td>
<td>/</td>
</tr>
<tr>
<td>010 1111</td>
<td>057</td>
<td>47</td>
<td>2F</td>
<td></td>
</tr>
</tbody>
</table>
Character Strings in C

• Examples:

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

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>
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.

<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>‘o’</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>
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:

```
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 (`\`)

```
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;$&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>
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 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:
- 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.dddddd, where the number of d’s is given by the precision (default 6).</td>
</tr>
<tr>
<td>e, E</td>
<td>double; [-]m.ddddddE+/-xx or [-]m.ddddddE+/−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
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);`
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 \n”, class_num);
// my_array now contains: Hello CPRE 288
printf(%s, another_array); // prints Goodbye
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); //??
String Manipulation Functions: strlen

```c
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
Class Activity

• Predict the value of *message* after each line:

```c
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: 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

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

Welcome to CPRE 288
What’s the desired program behavior?
First display “Welcome to ” and wait
Then display “elcome to C” and wait
Then display “lcome to CP” 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
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
Programming Example

Some details to take care

“display 10 chars from the starting pos.”

“shift the starting pos. to the next position”
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 screen becomes clear, with 0.5 second delay
  – Repeat this procedure
Lab 1 Programming Exercise

First, have a function to print the banner for one time

```c
void print_banner(char *msg, int start, int end);
```

This makes the rest of programming easier
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
int main()
{
    while (1)
    {
        for (…) // Phase 1
            ...
        for (…) // Phase 2
            ...
        for (…) // Phase 3
            ...
    }
}