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

Instructors:

Dr. Phillip Jones

Overview

- Announcements
- Simplest Embedded C program
- Variables
- Arrays
- C-strings

Announcements

- HW 1 due Sunday 9/2
- **Quiz 1 (15 min):** Thursday 8/30 at beginning of class: in Canvas (**Let me know if you need a paper version, before day of quiz**)
  - Can use one side of one page of notes (**must be on paper**)
    - Will be collected as part of class participation grade
  - Covers Lecture material (8/21 – 8/28)
  - Covers material from WW1 Lab.

C IN EMBEDDED SYSTEMS

Compare C and Java/C++

- C is a procedural language
  - No classes or objects
  - "Function" is the building block
- C structure: Uses a minimum set of language constructs
- "The C programming language" (Library has web version)
  - Quick Overall Intro: Chapter 1 (pgs 5 – 34)
  - Chapter 2
- Course Webpage: Resources sections
  - "The C Book": http://publications.gbdirect.co.uk/c_book/

Simplest Embedded Program

```
void main()
{
  while (1)
  {

  }  // do forever…
}
```

- Embedded programs often run forever

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
```

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)
{
   printf (“flag has been set”);
}

// enter an idle loop
while (1);
```

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

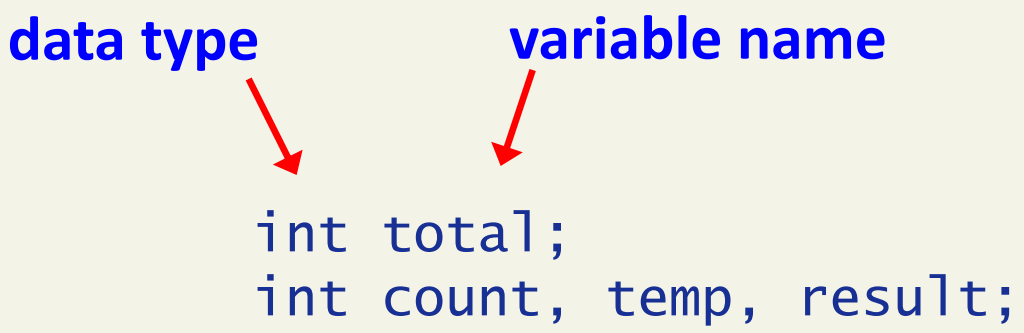
- auto
- const
- extern
- register
- signed
- static
- unsigned
- volatile

- 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



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

Name	Number of Bytes sizeof()	Range
char	1	0 to 255 or -128 to 127 (Depends on Compiler settings)
signed char	1	-128 to 127
unsigned char	1	0 to 255
short	2	-32,768 to 32,767
unsigned short	2	0 to 65,535
int	Varies by platform	Varies by platform
int (on TM4C123)	4	-2,147,483,648 to 2,147,483,647
unsigned int (on TM4C123)	4	0 to 4,294,967,295
(pointer)	Varies by platform	Varies by platform
(pointer on TM4C123)	4	Address Space

- 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

Name	Number of Bytes sizeof()	Range
long	4	-2,147,483,648 to 2,147,483,647
signed long	4	-2,147,483,648 to 2,147,483,647
unsigned long	4	0 to 4,294,967,295
long long	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4	±1.175e-38 to ±3.402e38
double	Varies by platform	
double (on TM4C123)	8	±2.3E-308 to ±1.7E+308

- 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

```
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

```
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

```
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

```
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];

  Size: i.e. Number of elements

  data type

  variable name

- Last element is found at *N-1* location
- Curly brackets can be used to initialize the array

Arrays in C

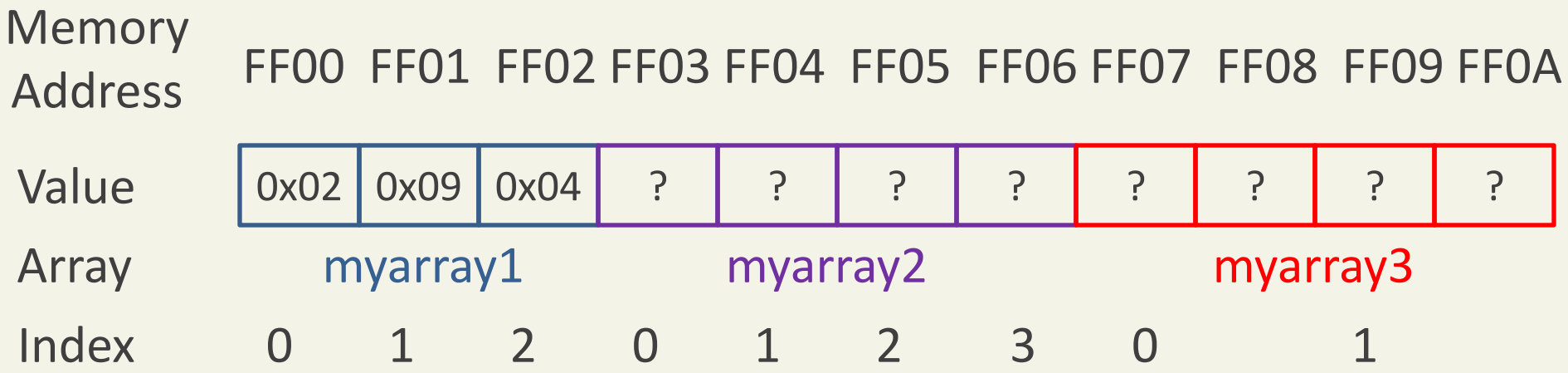
- Examples:

```
// 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 short's
short myarray3[2];
```

Arrays in C

- Examples:

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

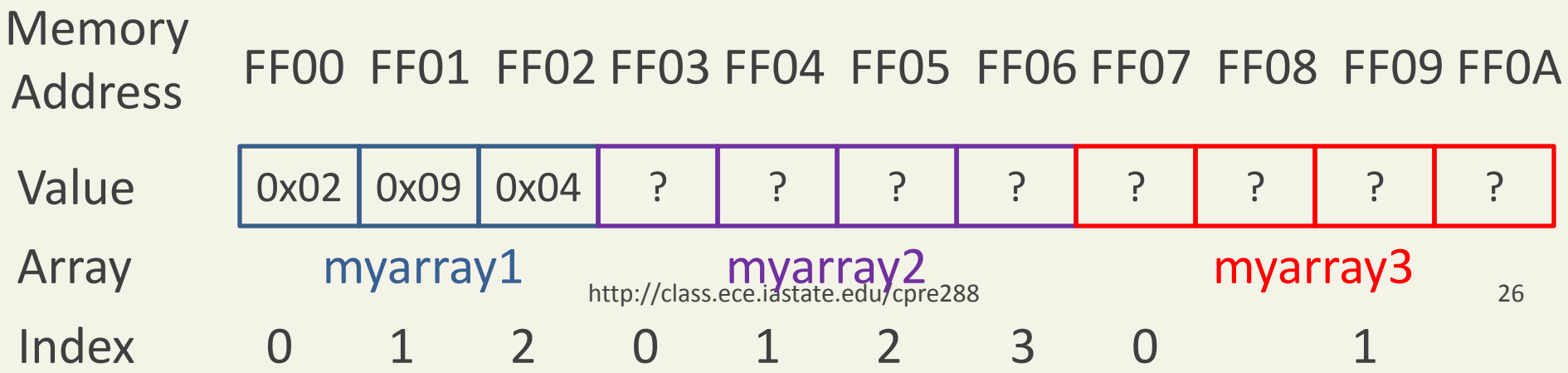


Arrays in C

- 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 == ??`

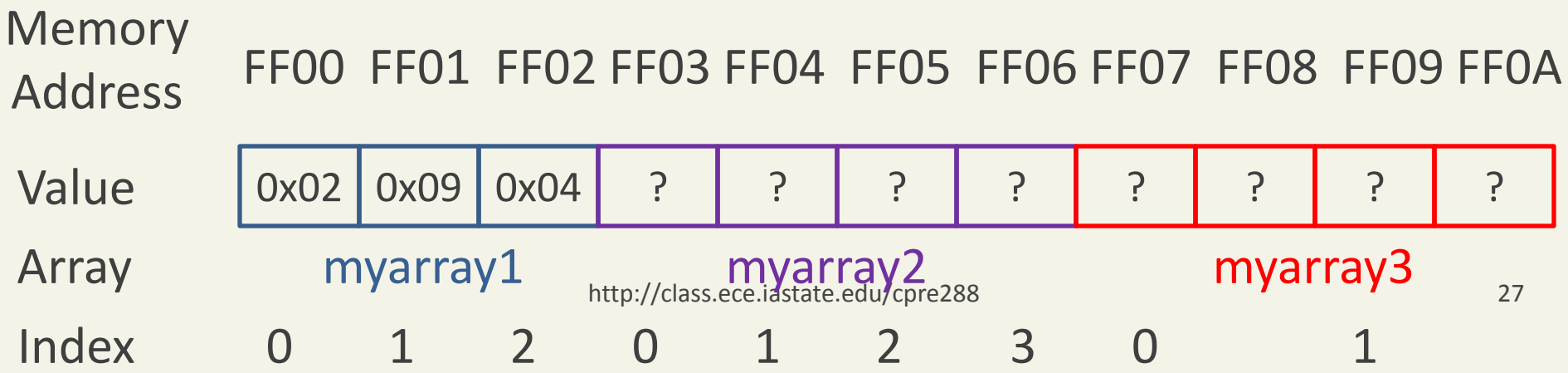


Arrays in C

- 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 == 0xFF07`



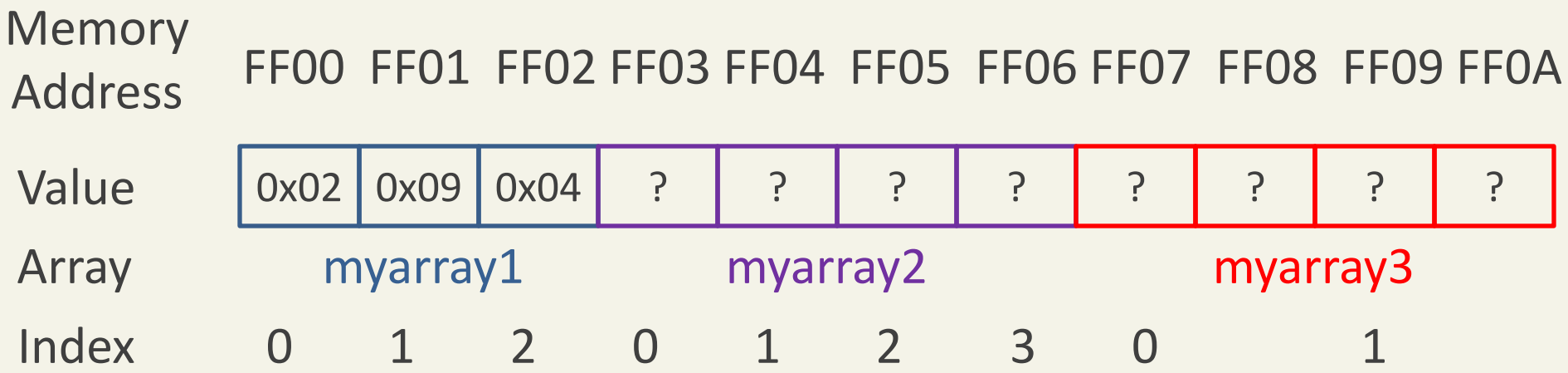
Arrays

- 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:

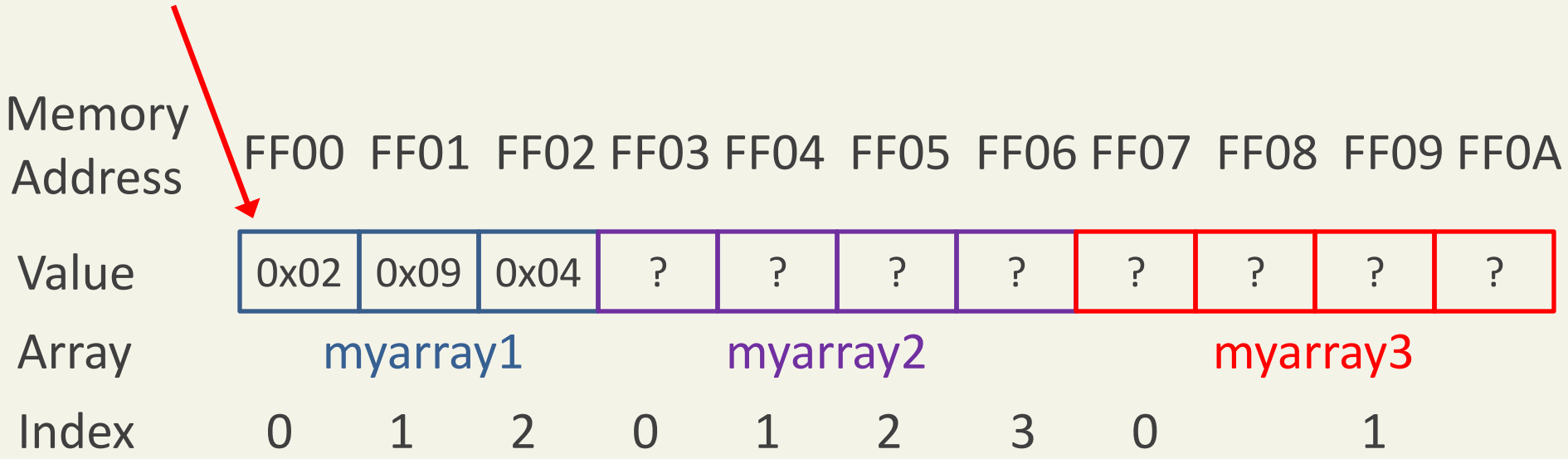
```
char  myarray1[3] = {2, 9, 4};
char  myarray2[4];
short myarray3[2];
```



Arrays in C

- Examples:

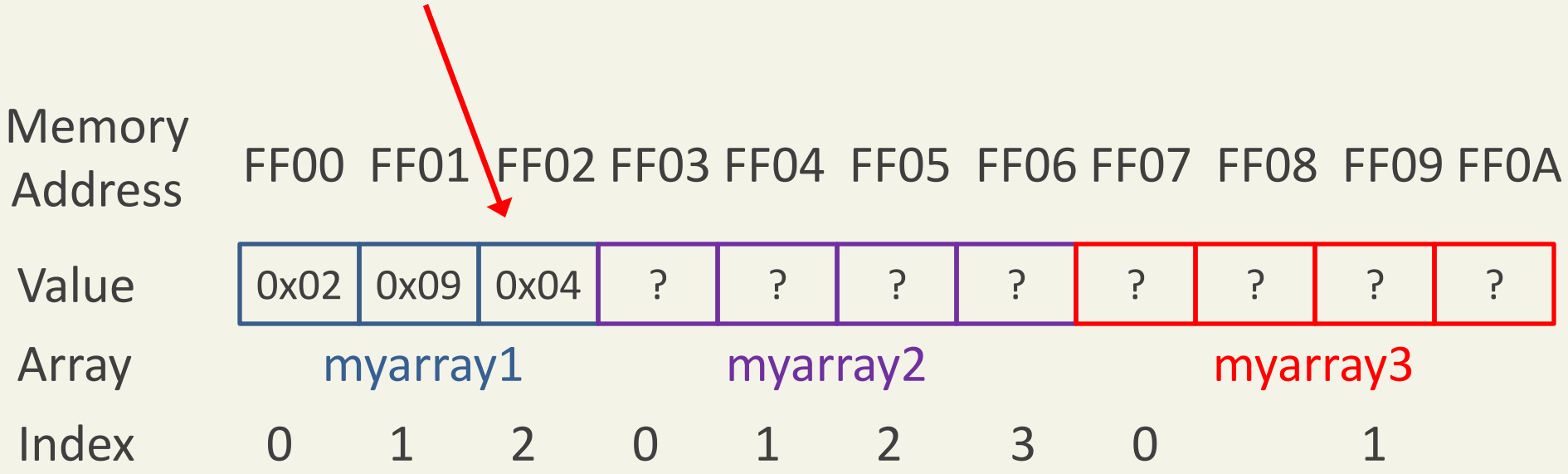
```
char  myarray1[3] = {2, 9, 4};
char  myarray2[4];
short myarray3[2];

myarray1[0] // First element of myarray1
```



Arrays in C

- Examples:

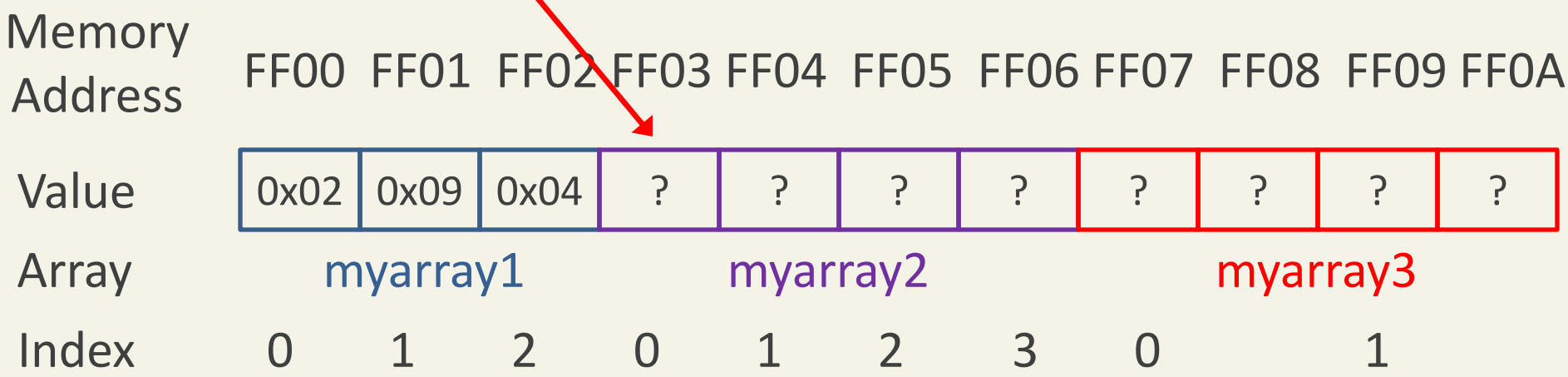
```
char  myarray1[3] = {2, 9, 4};
char  myarray2[4];
short myarray3[2];

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



Arrays in C

- Examples:

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

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

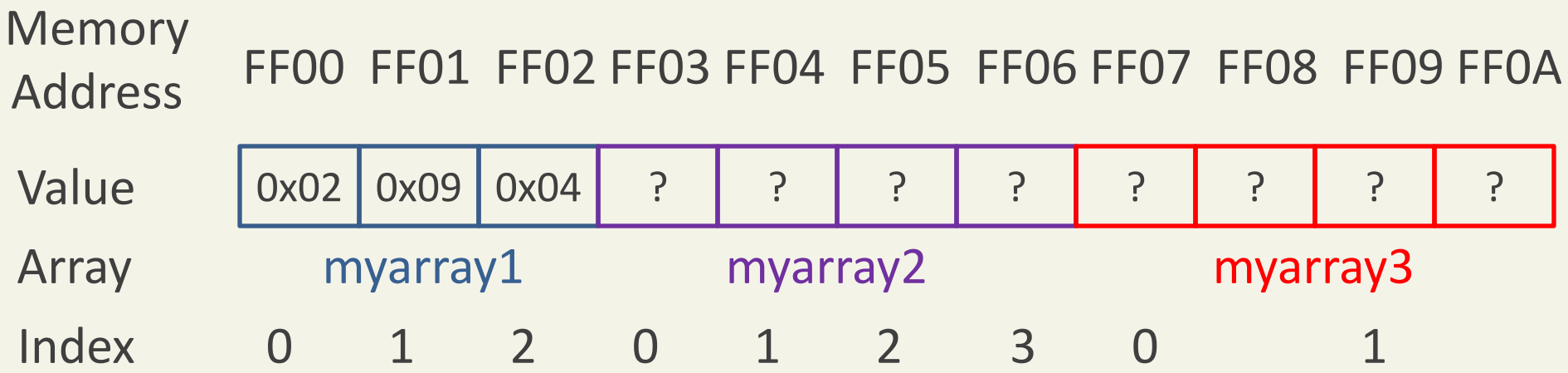


Arrays in C

- Examples:

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

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

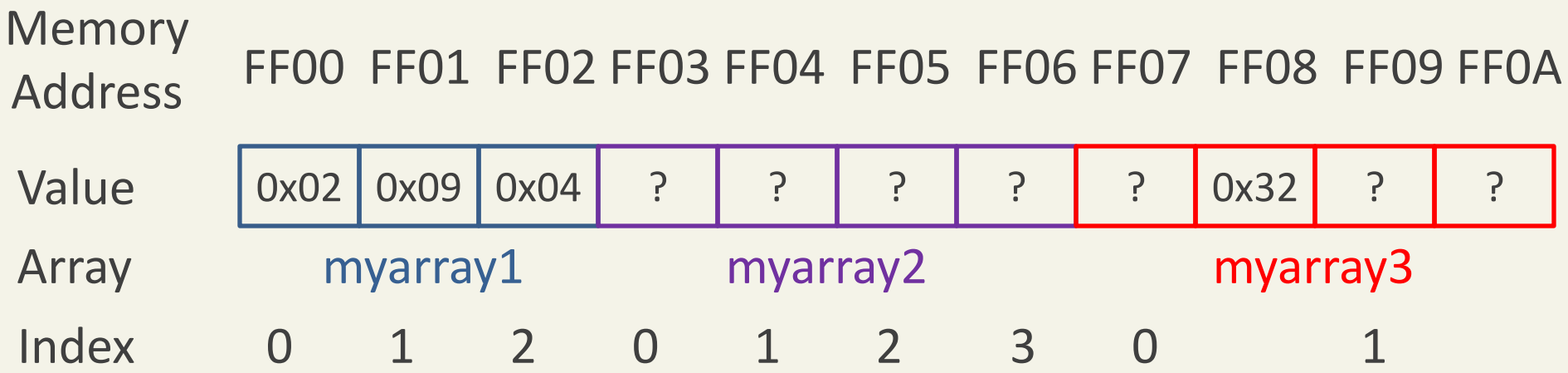


Arrays in C

- Examples:

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

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



Arrays

Array Copy Example

```
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

- **Do not** use statements like: *if (str1 == str2)* 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

```
char str1[] = “123”;
char str2[] = “123”;

if (strcmp(str1, str2) == 0)
{
   // str1 matches str2
}
```

Character Strings in C

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

```
char str[] = “hi”;
char str[3] = { ‘h’, ‘i’, ‘\0’ };
char str[3] = { 104, 105, 0 };
Char str[3] = {0x68, 0x69, 0x0};
```

Binary	Oct	Dec	Hex	Glyph
010 0000	040	32	20	sp
010 0001	041	33	21	!
010 0010	042	34	22	"
010 0011	043	35	23	#
010 0100	044	36	24	$
010 0101	045	37	25	%
010 0110	046	38	26	&
010 0111	047	39	27	'
010 1000	050	40	28	(
010 1001	051	41	29	)
010 1010	052	42	2A	*
010 1011	053	43	2B	+
010 1100	054	44	2C	,
010 1101	055	45	2D	-
010 1110	056	46	2E	.
010 1111	057	47	2F	/
011 0000	060	48	30	0
011 0001	061	49	31	1
011 0010	062	50	32	2
011 0011	063	51	33	3
011 0100	064	52	34	4
011 0101	065	53	35	5
011 0110	066	54	36	6
011 0111	067	55	37	7
011 1000	070	56	38	8
011 1001	071	57	39	9
011 1010	072	58	3A	:
011 1011	073	59	3B	;
011 1100	074	60	3C	<
011 1101	075	61	3D	=
011 1110	076	62	3E	>
011 1111	077	63	3F	?

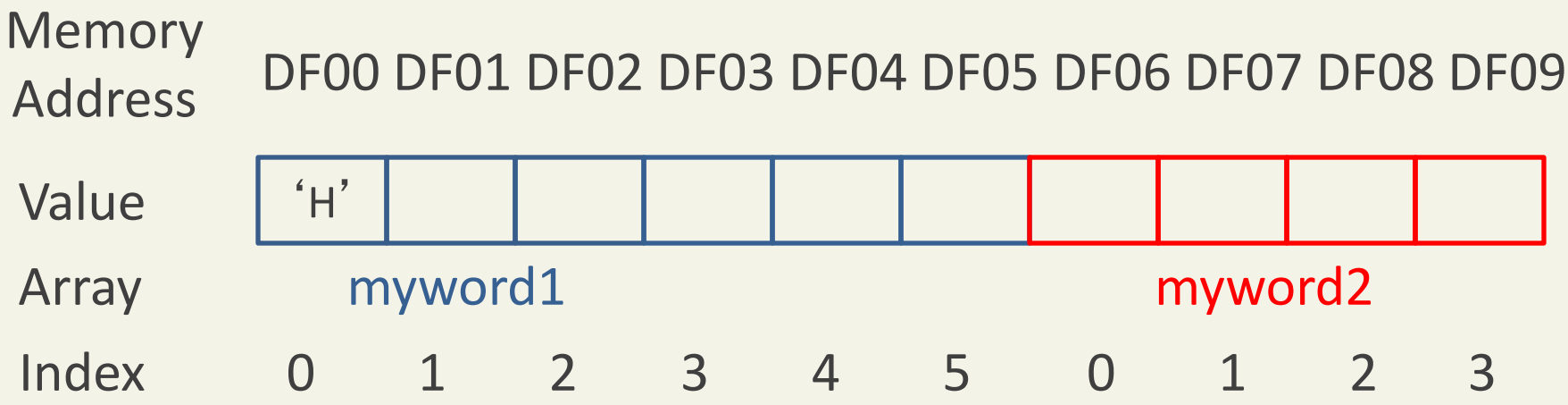
Binary	Oct	Dec	Hex	Glyph
100 0000	100	64	40	@
100 0001	101	65	41	A
100 0010	102	66	42	B
100 0011	103	67	43	C
100 0100	104	68	44	D
100 0101	105	69	45	E
100 0110	106	70	46	F
100 0111	107	71	47	G
100 1000	110	72	48	H
100 1001	111	73	49	I
100 1010	112	74	4A	J
100 1011	113	75	4B	K
100 1100	114	76	4C	L
100 1101	115	77	4D	M
100 1110	116	78	4E	N
100 1111	117	79	4F	O
101 0000	120	80	50	P
101 0001	121	81	51	Q
101 0010	122	82	52	R
101 0011	123	83	53	S
101 0100	124	84	54	T
101 0101	125	85	55	U
101 0110	126	86	56	V
101 0111	127	87	57	W
101 1000	130	88	58	X
101 1001	131	89	59	Y
101 1010	132	90	5A	Z
101 1011	133	91	5B	[
101 1100	134	92	5C	\
101 1101	135	93	5D	]
101 1110	136	94	5E	^
101 1111	137	95	5F	_

Binary	Oct	Dec	Hex	Glyph
110 0000	140	96	60	`
110 0001	141	97	61	a
110 0010	142	98	62	b
110 0011	143	99	63	c
110 0100	144	100	64	d
110 0101	145	101	65	e
110 0110	146	102	66	f
110 0111	147	103	67	g
110 1000	150	104	68	h
110 1001	151	105	69	i
110 1010	152	106	6A	j
110 1011	153	107	6B	k
110 1100	154	108	6C	l
110 1101	155	109	6D	m
110 1110	156	110	6E	n
110 1111	157	111	6F	o
111 0000	160	112	70	p
111 0001	161	113	71	q
111 0010	162	114	72	r
111 0011	163	115	73	s
111 0100	164	116	74	t
111 0101	165	117	75	u
111 0110	166	118	76	v
111 0111	167	119	77	w
111 1000	170	120	78	x
111 1001	171	121	79	y
111 1010	172	122	7A	z
111 1011	173	123	7B	{
111 1100	174	124	7C	|
111 1101	175	125	7D	}
111 1110	176	126	7E	~

Character Strings in C

- Examples:

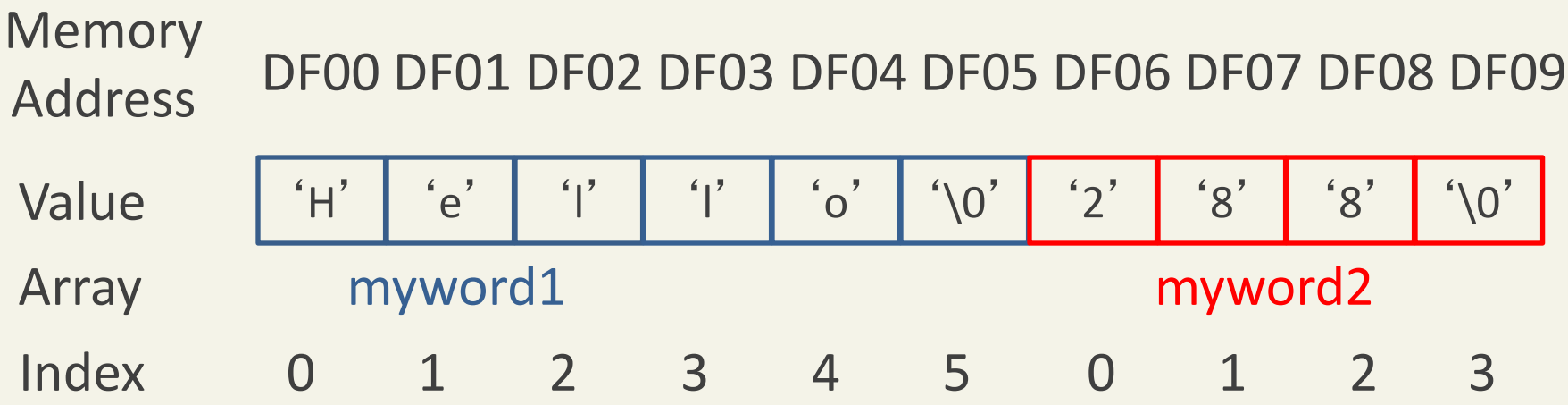
```
char myword1[6] = “Hello”;  //  declare and initialize
char myword2[4]   = “288”;  //  declare and initialize
```



Character Strings in C

- Examples:

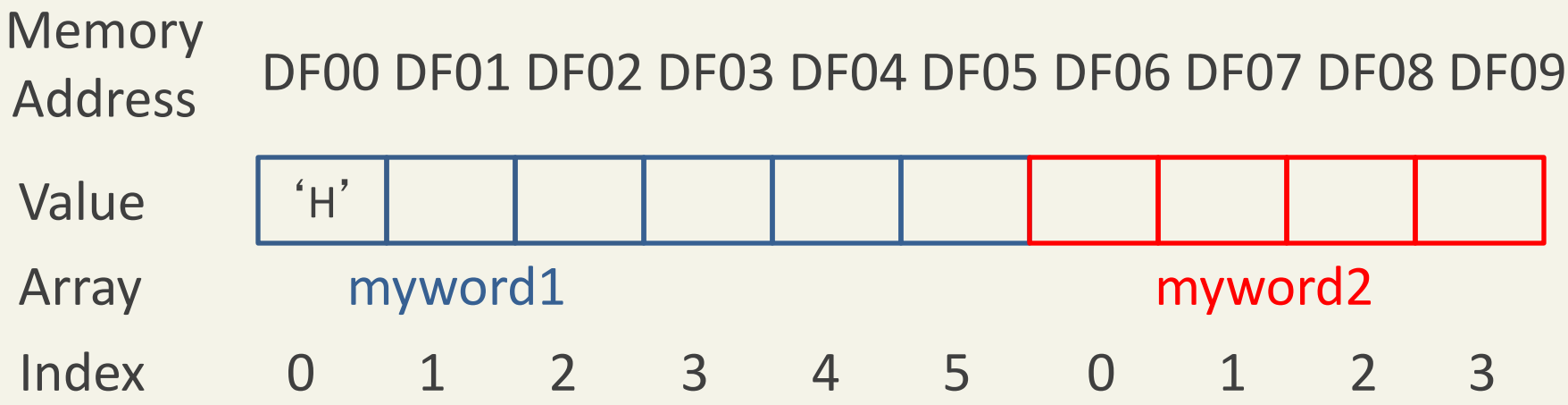
```
char myword1[6] = “Hello”;  //  declare and initialize
char myword2[4]   = “288”;  //  declare and initialize
```



Character Strings in C

- Examples:

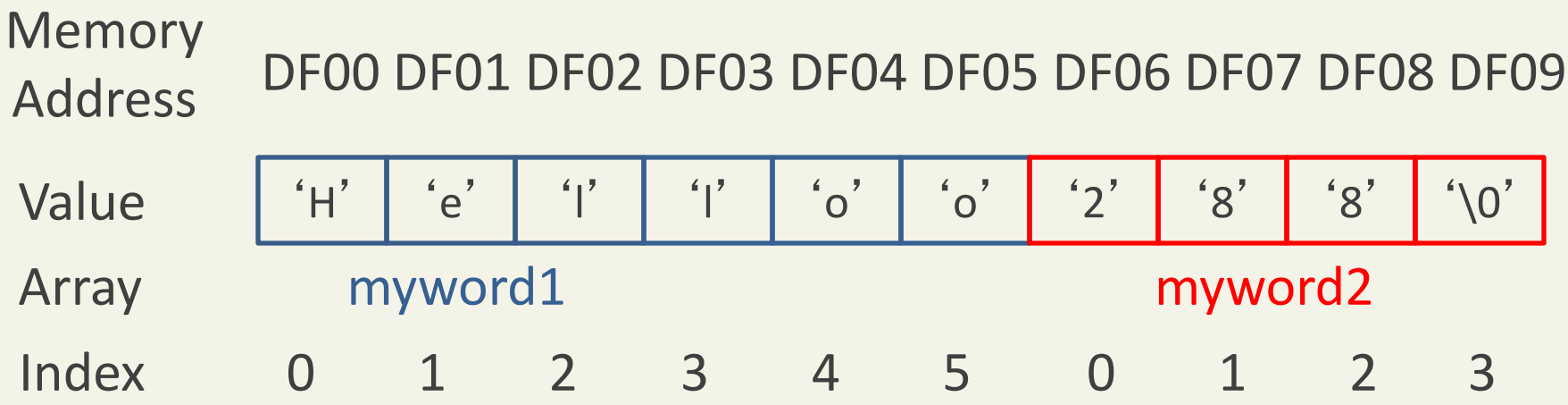
```
char myword1[6] = “Helloo”;  //  declare and initialize
char myword2[4]   = “288”;  //  declare and initialize
```



Character Strings in C

- Examples:

```
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:

```
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

Binary	Oct	Dec	Hex	Abbr	Carrot	Escape	Description
000 0000	0	0	0	NUL	^@	\0	Null character
000 0111	7	7	7	BEL	^G	\a	Bell
000 1000	10	8	8	BS	^H	\b	Backspace
000 1001	11	9	9	HT	^I	\t	Horizontal Tab
000 1010	12	10	0A	LF	^J	\n	Line feed
000 1011	13	11	0B	VT	^K	\v	Vertical Tab
000 1100	14	12	0C	FF	^L	\f	Form feed
000 1101	15	13	0D	CR	^M	\r	Carriage return
001 1011	33	27	1B	ESC	^[	\e	Escape
010 0111	47	39	27	'		\'	Single Quote
010 0010	42	34	22	"		\"	Double Quote
101 1100	134	92	5C	\		\\	Backslash

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", 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*

Character	Argument type; Printed As
d,i	int; decimal number
o	int; unsigned octal number (without a leading zero)
x,X	int; unsigned hexadecimal number (without a leading 0x or 0X), using abcdef or ABCDEF for 10, ...,15.
u	int; unsigned decimal number
c	int; single character
s	char *; print characters from the string until a '\0' or the number of characters given by the precision.
f	double; [-]*m.dddddd*, where the number of *d*'s is given by the precision (default 6).
e,E	double; [-]*m.ddddddx*e+/-*xx* or [-]*m.ddddddx*E+/-*xx*, where the number of *d*'s is given by the precision (default 6).
g,G	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.
p	void *; pointer (implementation-dependent representation).
%	no argument is converted; print a %

Formatting Strings: Example % formats

```
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
```

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 \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); //??

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

END

LAB 1 QUICK OVERVIEW

Lab 1: Introduction to the Platform

Purpose: Introduction to Code Composer and CyBot Platform

- Code Composer: The integrated development environment (IDE) for TI platforms
- CyBot: An integrated hardware platform of iRobot Create 2 and LaunchPad EK-TM4C123GXL microcontroller board

Code Composer

An IDE from TI for microcontroller platforms

- Source code editing
- Compiling building
- Download binary to boards
- Debug

Lab 1

Lab 1: Introduction to Code Composer

- Part 1 “Hello, world”
  - Build, download, and execute
- Part 2 Debug 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

W	e	l	c	o	m	e		t	o		C	P	R	E		2	8	8

Programming Example

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

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

Programming Example

Some details to take care

"display 10 chars from the starting pos."

"shift the starting pos. to the next position"

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 0.3 second delay
- Continue rotating until the screen becomes clear, with 0.3 second delay
- Repeat this procedure

Lab 1 Programming Exercise

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

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

OR

```
void print_banner(char msg[], int start, int end);
```

This makes the rest of programming easier

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

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