



Computer Systems and Networks

ECPE 170 – Jeff Shafer – University of the Pacific

C Programming 1

Lab Schedule

Activities

➤ This Week

- Intro to Build Tools and Makefiles
- Intro to C 1
- **Lab 3 – Build Tools**

➤ Next Week

- Intro to C 2
- **Lab 4 – C Programming Project**

Deadlines

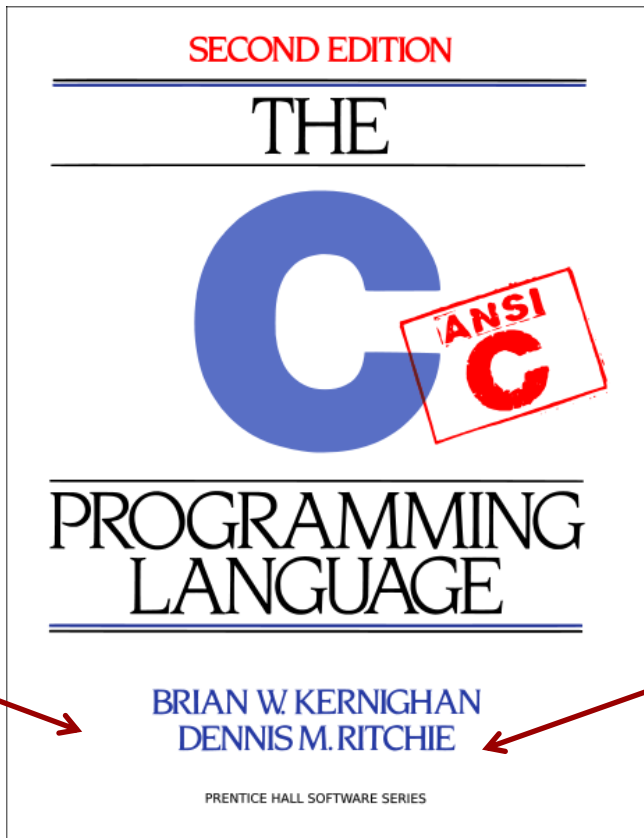
- **Lab 3 – Feb 2nd 2021 by 5am**
- **Lab 4 – Feb 16th 2021 by 5am**

Person of the Day: Dennis Ritchie



- Creator of **C programming language**
- Co-creator of **Unix**
(with Ken Thompson, Brian Kernighan, and others at Bell Labs)
- Winner of **ACM Turing Award**
- 9/9/1941—10/12/2011

Person of the Day: Dennis Ritchie



➤ *“Pretty much everything on the web uses those two things: C and UNIX. The browsers are written in C. The UNIX kernel — that pretty much the entire Internet runs on — is written in C. Web servers are written in C, and if they’re not, they’re written in Java or C++, which are C derivatives, or Python or Ruby, which are implemented in C. And all of the network hardware running these programs I can almost guarantee were written in C. It’s really hard to overstate how much of the modern information economy is built on the work Dennis did.”*

➤ Rob Pike, Bell Labs / Google



Dennis Ritchie and Ken Thompson use a teletypewriter to run a program on a UNIX-based computer system they co-founded at Bell Labs in New Jersey. Their development work more than 40 years ago facilitated the realization of the Internet.

C Programming



C++ Features Not in C

- No **classes** / object-oriented programming
- No **new** / **delete**
- No stream operators (<< and >>), cin, cout, ...
- No C++ Standard Libraries (e.g. iostream)
- `bool` keyword
 - Added in C99 standard
- Declare variables anywhere inside function
 - Added in C99 standard

Console I/O



Output with printf()

- `printf("This is a string\n");`
- `printf("The integer is %i\n", num);`
- `printf("The floating-point values are %g and %g\n", num1, num2);`

Output with printf()

Format “Type” Code	Corresponding Variable Type
d or i	int (interpret as signed 2’s comp)
u	int (interpret as unsigned)
x	int (print as hexadecimal)
f or g	float/double
c	char
s	string (null-terminated array of chars)

Prefix with l or ll (i.e. “long” or “long long” for larger 64-bit data types)


➤ Lots of formatting options not listed here...

➤ # of digits before / after decimal point?

➤ Pad with zeros?

Input with scanf()

- Input from console
- `scanf("%d %c", &myint, &mychar)`
- Requires the **address** of the destination variable
 - Use the `&` operator to obtain address
- Caveat: Array names are already the “address of”!
 - ```
char myarray[8];
scanf("%s", myarray)
```

 No `&` needed here!

# Documentation

- **Man(ual) pages exist for common programming functions too**

```
$ man printf
```

```
$ man scanf
```

# Arrays



# Arrays

- Contiguous block of memory
- You can have arrays for `int`, `char`, `float`, `double`, structures...

```
int myarray[5]; //static declaration
```

|            |            |            |            |            |
|------------|------------|------------|------------|------------|
| <b>4</b>   | <b>8</b>   | <b>12</b>  | <b>16</b>  | <b>20</b>  |
| myarray[0] | myarray[1] | myarray[2] | myarray[3] | myarray[4] |

NOTE: Name of the array is the address of the first element

```
printf("%p", myarray); //prints what?
```

# 2D Arrays

```
int myarray[5][5]; //static declaration
```

Memory  
map:

|                             |                              |                              |             |              |
|-----------------------------|------------------------------|------------------------------|-------------|--------------|
| Address: 4<br>myarray[0][0] | Address: 8                   | Address: 12                  | Address: 16 | Address: 20  |
| Address: 24                 | Address: 28<br>myarray[1][1] | Address: 32                  | Address: 36 | Address: 40  |
| Address: 44                 | Address: 488                 | Address: 52                  | Address: 56 | Address: 60  |
| Address: 64                 | Address: 68                  | Address: 72<br>myarray[3][2] | Address: 76 | Address: 80  |
| Address: 84                 | Address: 88                  | Address: 92                  | Address: 96 | Address: 100 |

# Problem 1: Looping through an array

- Consider a 3-D array, `int image[256][256][3]` for an RGB color image. The first subscript denotes the number of rows, the second subscript denotes the number of columns, and the third subscript denotes the number of color channels. For example, a pixel at row `i` and column `j` will have an R value given by `image[i][j][0]`, G value given by `image[i][j][1]`, and B value given by `image[i][j][2]`. Any pixel has a yellow color if its R and G values are 255 and B value is 0.
- Write a `for` loop to search for the location of the very first yellow pixel in `image`. The search should terminate once the yellow pixel is found. Search in row-wise manner.

A yellow hexagon with a black border, containing the text "P1" in bold black font.

**P1**



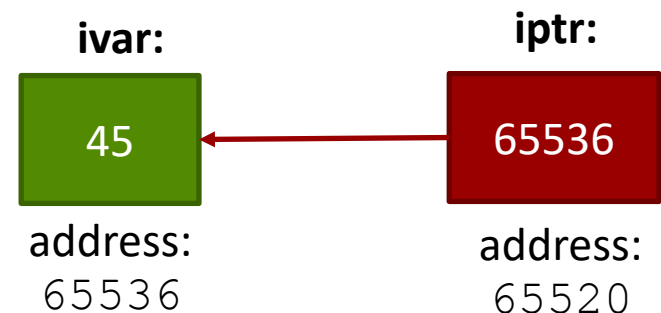
# Pointers



# Pointers

- **Pointers** are special variables that hold/store memory addresses of other variables
- When a pointer (e.g. *iptr*) holds the address of an integer variable (e.g. *ivar*), then we say: “*iptr* is an integer pointer that points to *ivar*”

```
int ivar=45;
int *iptr;
iptr = &ivar;
//iptr points to ivar
```



# Pointers

➤ **&** is ‘**address of variable**’ operator

➤ Example: `&ivar` translates to  
“address of variable `ivar`”

➤ **\*** is ‘**value at address stored in pointer**’ operator

➤ Example: `*iptr` translates to  
“value at address stored in pointer `iptr`”

# Pointers

- Can have multiple levels of “indirection”
- `int *myptr`
  - A pointer to an integer
- `int **myptr`
  - A pointer to a pointer to an integer
- `int ***myptr`
  - A pointer to a pointer to a pointer to an integer

# Problem 2 - Pointers

Consider the variables below:

|                     |                             |
|---------------------|-----------------------------|
| Variable Name: ivar | Pointer variable name: iptr |
| value: 5            | value:                      |
| Address: 0xFFABCD   | Address: 0xAFABAD           |

```

int ivar=5;
int *iptr;
iptr = &ivar;
printf("\n %d", ivar); prints _____
printf("\n %x", &ivar); prints _____
printf("\n %x", &iptr); prints _____
printf("\n %d", *iptr); prints _____

```



# Problem 3 – More Pointers

|                     |                             |                             |
|---------------------|-----------------------------|-----------------------------|
| Variable Name: ivar | Pointer variable name: iptr | Pointer variable name: dptr |
| value: 5            | value:                      | value:                      |
| Address: 0xFFABCD   | Address: 0xAFABAD           | Address: 0xFFACBD           |

```

int ivar=5;
int *iptr;
int **dptr;
iptr = &ivar;
dptr=&iptr;
printf("\n %x", dptr); prints _____
printf("\n %x", iptr); prints _____
printf("\n %u", **dptr); prints _____

//printf("\n %x", *dptr); prints _____
//printf("\n %x", &dptr); prints _____
//printf("\n %x", *(&iptr)); prints _____

```

**P3**

# Problem 4 – More Pointers

Collaborate with a  
partner and answer  
Question 4

A yellow hexagon with a black border, containing the text 'P4' in black.

**P4**

# C-Strings (Arrays of Characters)



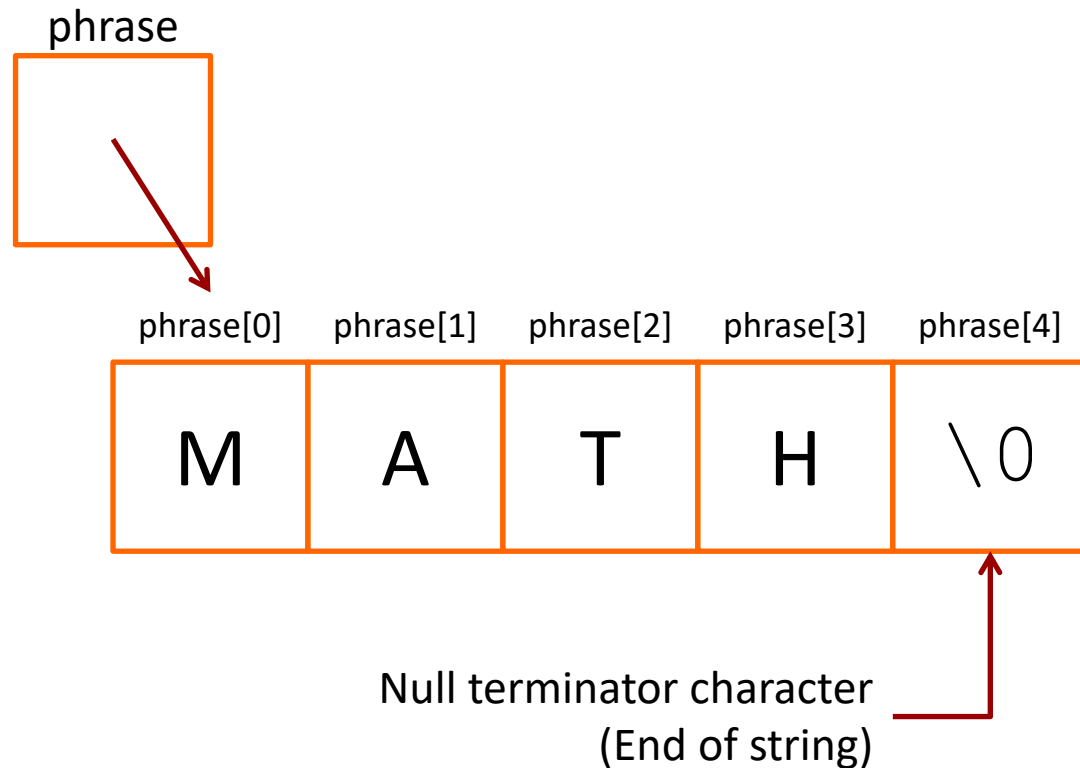


# C Strings

- **There is no such thing as a “string” in C!**
- What do you get? **An array of characters**
  - Terminated by the null character `'\0'`
- Must manipulate element by element...
  - Not enough room in the array? Need a bigger array

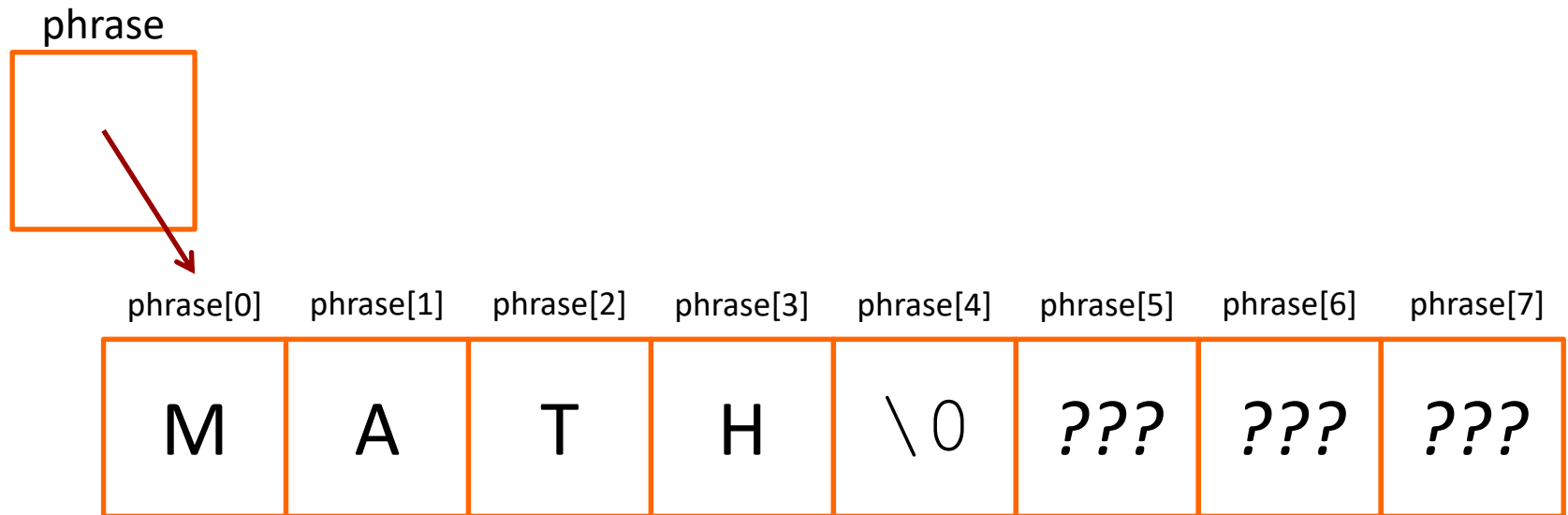
# Arrays of Characters

➤ `char phrase[] = "Math";`



# Arrays of Characters

➤ `char phrase[8] = "Math";`



`printf("%s\n", phrase);`    **Prints until it reaches the `\0` character!**

# Helpful Library for Character Arrays

➤ `#include <string.h>`

➤ Useful functions

➤ `strcpy` - String copy

➤ `strcmp` - String compare

➤ `strlen` - String length

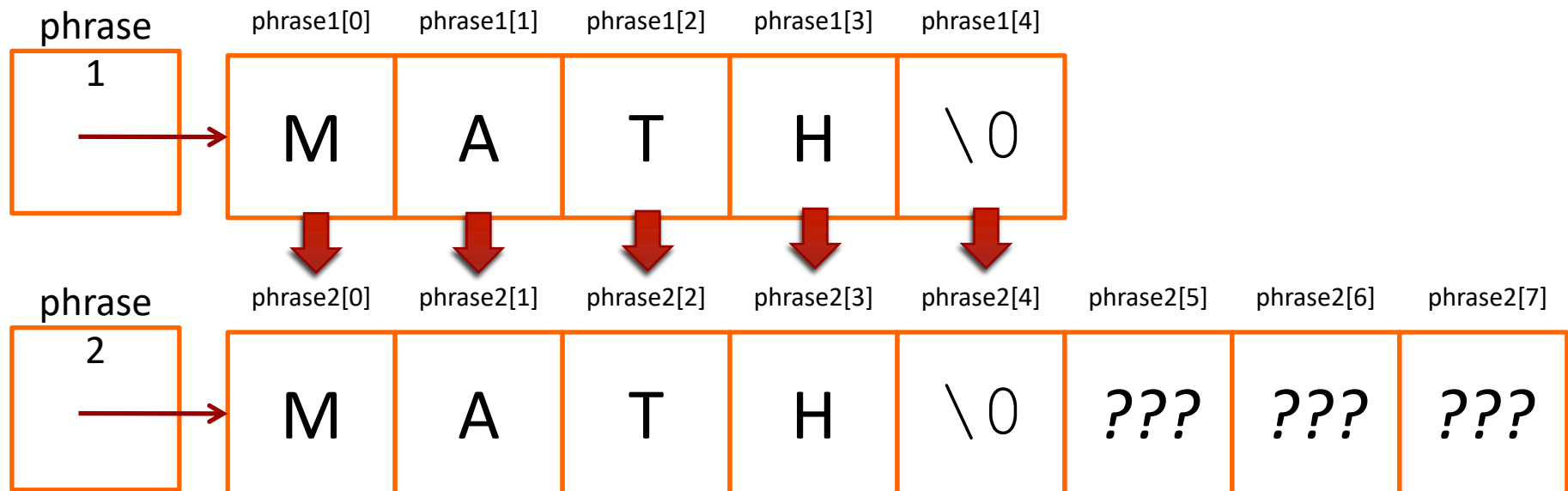
➤ `strcat` - String concatenate

# String Copy

➤ `char phrase1[] = "Math";`

➤ `char phrase2[8];`

➤ `strcpy(phrase2, phrase1);`

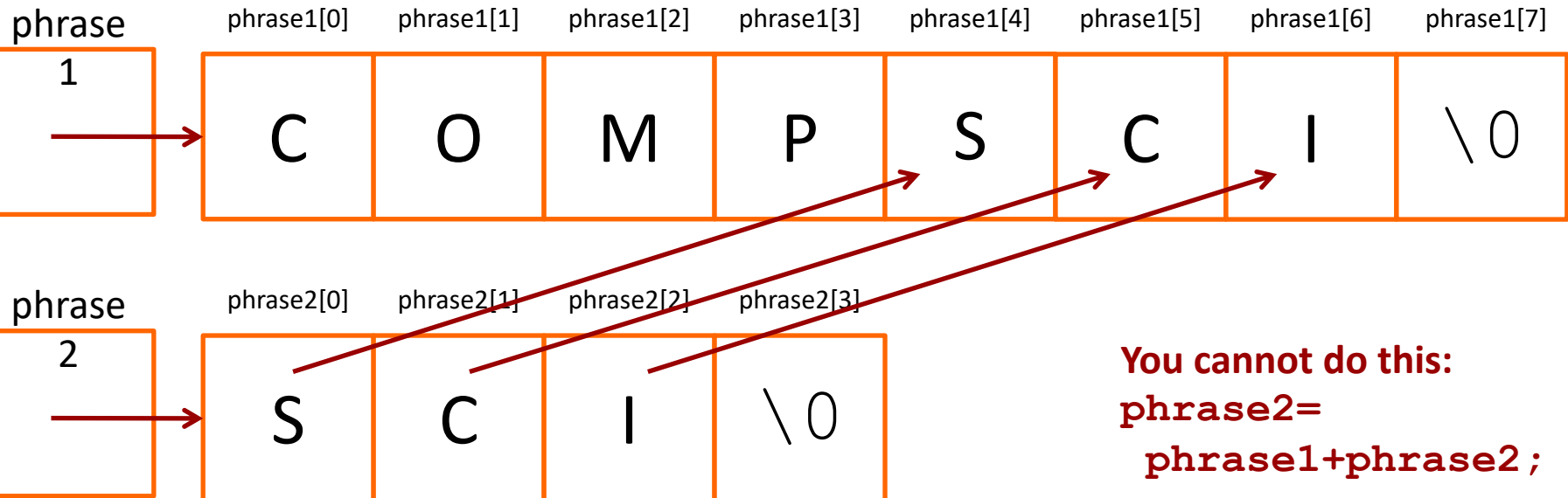


# String Concatenation

➤ `char phrase1[8] = "Comp";`

➤ `char phrase2[] = "Sci";`

➤ `strcat(phrase1, phrase2);`



**You cannot do this:**  
`phrase2 = phrase1 + phrase2;`

# ctype Library

- Useful for character manipulation
- `#include <ctype.h>`
- **`toupper(char)` / `tolower(char)`** – Converts character to uppercase or lowercase
  - Example:

```
char c = toupper('a');
printf("%c", c); // A
```

# cctype Library

- **isalpha(char)** – Is the character a letter?
- **isdigit(char)** – Is the character a number 0-9?
- **isspace(char)** – Is the character whitespace?  
(space or newline character)
- **ispunct(char)** – Is the character punctuation?  
(technically, a visible character that is not whitespace, a letter, or a number)
- ... and several other variations



# File I/O



# File I/O Functions

- `fopen` – opens a text file
- `fclose` – closes a text file
- `feof` – test for end-of-file
- `fgets` – reads a string from a file, stopping at EOF or newline
- `fwrite` – writes array of characters to a file
- `fgetc` – reads a character from a file
- `fputc` – prints a character to a file

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
 FILE *ptr_file;
```

```
 char buf[1000];
```

```
 ptr_file = fopen("input.txt", "r");
```

```
 if (!ptr_file)
```

```
 return 1;
```

```
 while (fgets(buf, 1000, ptr_file) != NULL)
```

```
 printf("%s", buf);
```

```
 fclose(ptr_file);
```

```
 return 0;
```

```
}
```