LECTURE 5: C PROGRAMMING

Computer Systems and Networks

Dr. Pallipuram
(vpallipuramkrishnamani@pacific.edu)

Today's Class

- Pointer basics
- Pointers and multi-dimensional arrays
  - `malloc, calloc, free`
- 2D array manipulation for Lab 4
- Strings in C

Pointer Arithmetic

Only addition and subtraction are allowed with pointers.

All pointers increase and decrease by the length of the data-type they point to.

Example: If an integer pointer, `iptr` holds address 32, then after the expression `iptr++`,
`iptr` will hold 36 (assuming integer is 4 bytes).

Problem 1

The name of the array is actually a pointer pointing to the first element of the array.

Consider an integer array named `array`.

Subscript	[0]	[1]	[2]	[3]	[4]
Value	5	6	4	8	2
Address	65528	65532	65536	65540	65544

```
printf("\n %u:",array); //prints 65528
printf("\n %u:array+2); //prints 65536
printf("\n %u:*(array+1)); //literally translates to array[1]
//Prints 6
```

```
printf("\n", %u:array+3); //prints?_____
printf("\n", %u:*(array+3)); //prints?_____
```

Two methods of traversing 1-D array

Pointer Method

```
for (i=0;i<arraysize;i++)
  *(array+i)=*(array+i)+1;

//iterates through the
array and increments
contents by 1
```

Subscript Method

```
for (i=0;i<arraysize;i++)
  array[i]=array[i]+1;

//iterates through the
array and increments
contents by 1
```

More intuitive

Pointers and Functions: Call by value vs. Call by reference

Call by value

these are just copies.
No change to original variables

```
main(){
a=5,b=6;
update(a,b);
printf("%d",a);
}
update(int a, int b)
{
a=a-b;
}
```

modification to actual variable

Call by reference (pointer)

```
main(){
a=5,b=6;
update(&a,&b);
printf("%d",a);
}
update(int *a,int *b)
{
*a=*a-*b;
}
```

Example: Modify an array using function call

```
main(){
//assume int array a of size 5
update(a,5); //name of array is starting addr.
}

update(int *a,int size) {
int i=0;
for(i=0;i<size;i++)
     a[i]++;
}
```

Malloc – 1D

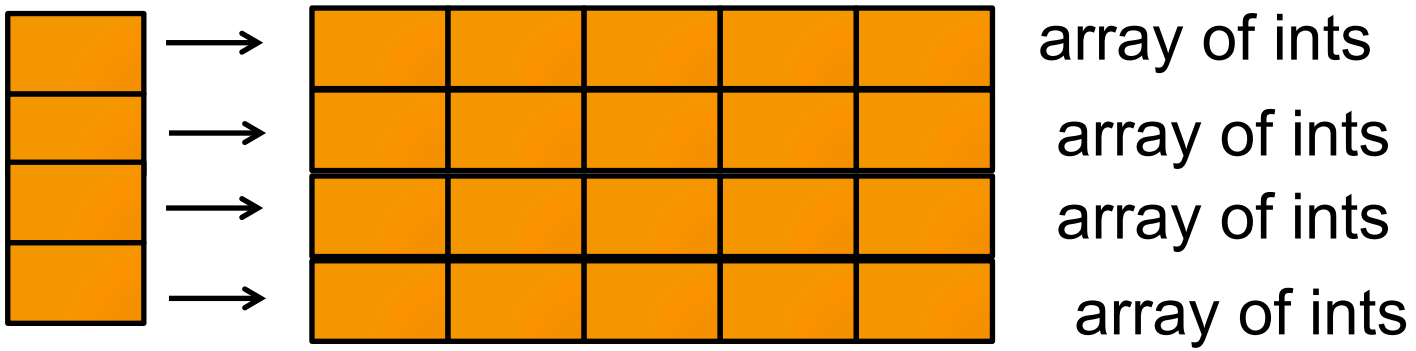
```
int *array;  //array of integers
array = (int *)malloc(sizeof(int)*5);
```

address:	60	64	68	72	76
value:	array[0]	array[1]	array[2]	array[3]	array[4]

array (pointer variable)
value: 60
pointer's addr: 32

Malloc – 2D Allocate 4x5 integers (important for lab 4)

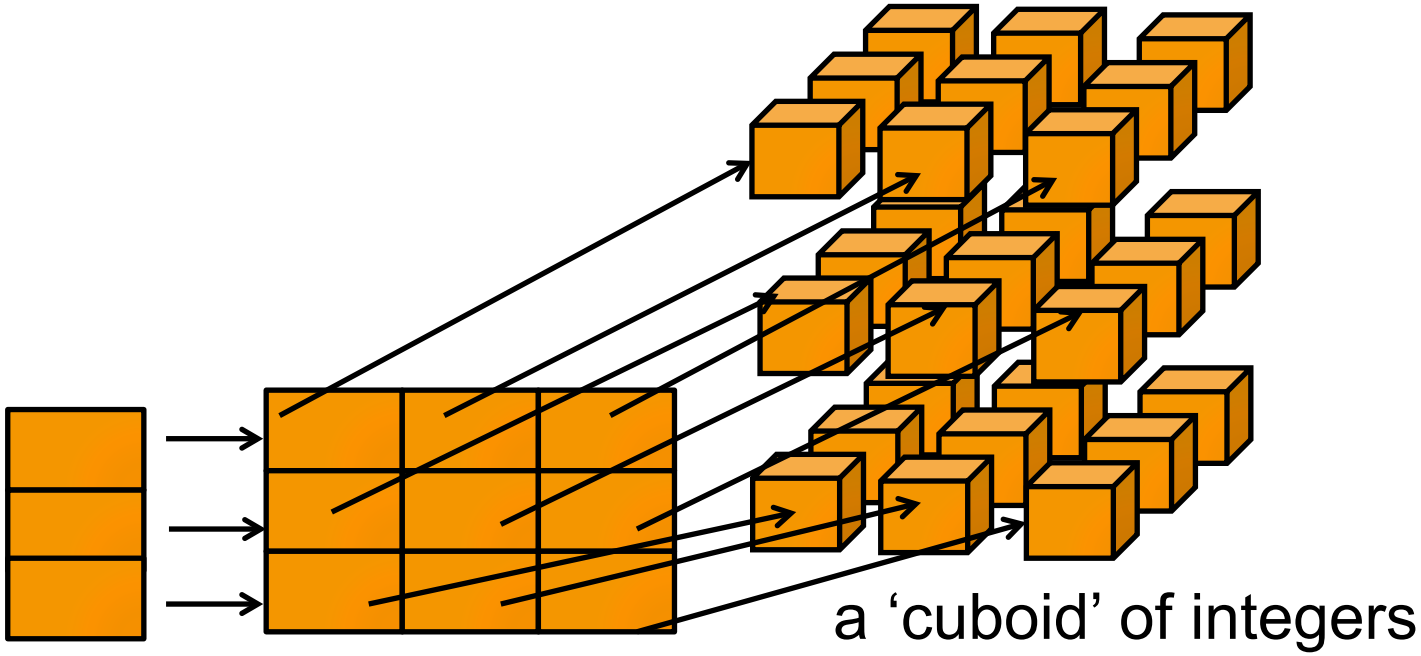
```
int **array; //a double pointer
array = (int **)malloc(sizeof(int *)*4);

 for(i=0;i<4;i++)
   array[i] = (int *)malloc(sizeof(int)*5);
```



an array of integer pointers

Malloc – 3D

```
int ***array; //a triple pointer
```



an array of double pointers

a matrix of single pointers

a ‘cuboid’ of integers

Problem 2

Dynamically allocate space for a 3-D color image of width, w; height, h; color channel, c. Any pixel is accessed as `image[height][width][c]`.

Calloc()

```
void * calloc(int count, int size)
```

- Basically the same as malloc!
  - Imagine you want an array of elements…
- Argument 1: # of elements to allocate
- Argument 2: Size of each element in bytes
- Return value: Pointer to the region

Realloc()

```
void * realloc(void *ptr, int size);
```

- **Resize** a dynamic region of memory
  - Note that it might **move** to a new address!
- Argument: Pointer to the original region
- Argument 2: Desired size in bytes of new region
- Return value: Pointer to the new region
  - It might be at the same address if you made it smaller
  - It might be at a new address if you made it larger

`#include <stdlib.h>`

Include this library to use malloc, realloc, and calloc!

C Structures

Structures are a nice way to bring certain related items together

```
struct database
{
  int id_number;
  int age;
  float salary;
};
int main()
{
  struct database employee; //an object
  employee.age = 22;
  employee.id_number = 1;
  employee.salary = 12000.21;
}
```

structure objects access members using dot operator

Problem 3 (Important for Lab 4)

Declare a structure called `board` that contains: a double character pointer `matrix`, two integer variables `height` and `width` denoting the number of rows and columns in the matrix. Inside main, do the following:

1. create a structure object called `myboard`, initialize `matrix` to NULL, set `height` to 7 and `width` to 7.
2. Dynamically allocate `matrix` to hold `height x width` elements

Traversing 2D array

```
main(){
//Assume a is dynamically allocated 2D array
update(a,5,5); //name of array is starting addr.
}

update(int **a,int height,int width) {
int i=0,j=0;
for(i=0;i<height;i++)
     for(j=0;j<width;j++)
          a[i][j]++;
}
```

Problem 4 (Useful for Lab 4)

Refer to Problem 3. Traverse the 2D `matrix` of dimensions `height` (rows) and `width` (columns). Find the first instance of small letter ‘e’. Obtain all the letters starting from ‘e’ placed diagonally downwards in this matrix. Store the letters in a 1D array, `buffer`. Make sure that buffer is of large enough size to contain all of the letters.

`free()` to free the Allocated space

```
free(variable name);
```

Remember to free all of the variables malloc'ed

Problem 5 – Free a 2D array (Useful for Lab 4)

`free()` is actually a reverse operation of `malloc`. The steps you use for `free` is opposite of the steps for `malloc`. Free a dynamically allocated 2D array.

String Operations

C Strings

There is no such thing as a “string” in C!

What do you get? **An array of characters**

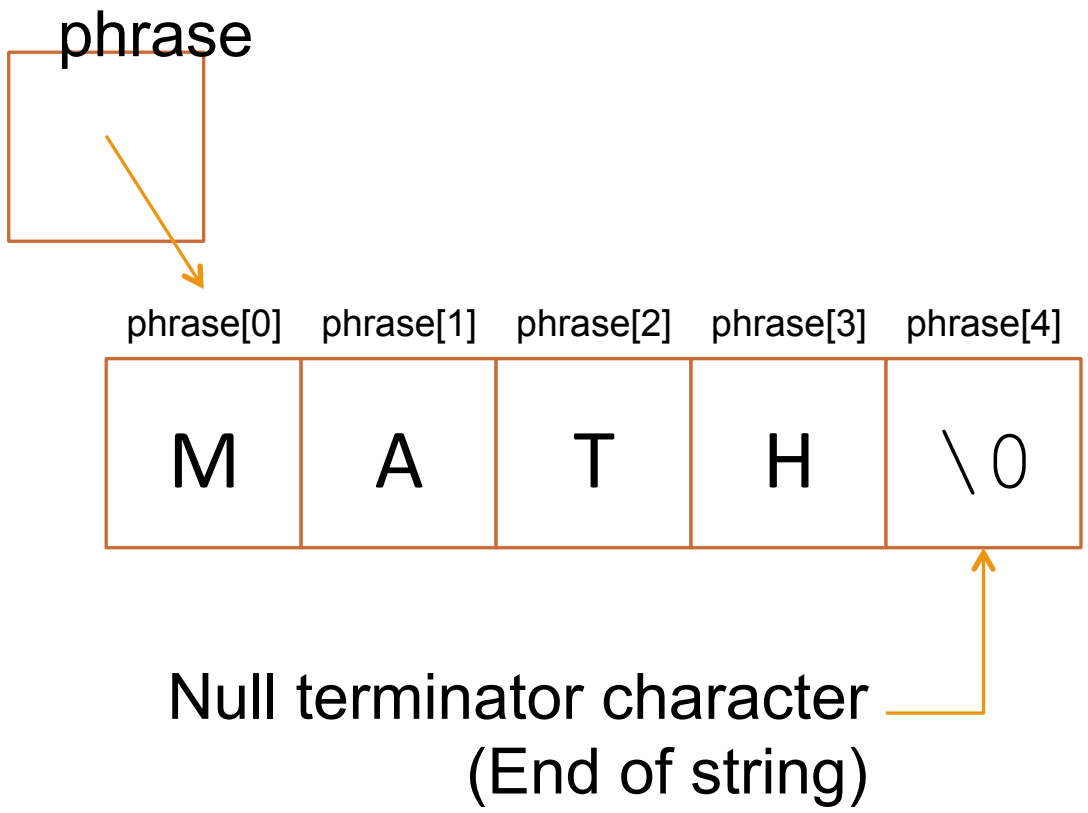
- Terminated by the null character `'\0'`

Must manipulate element by element…

- Not enough room in the array? Need a bigger array

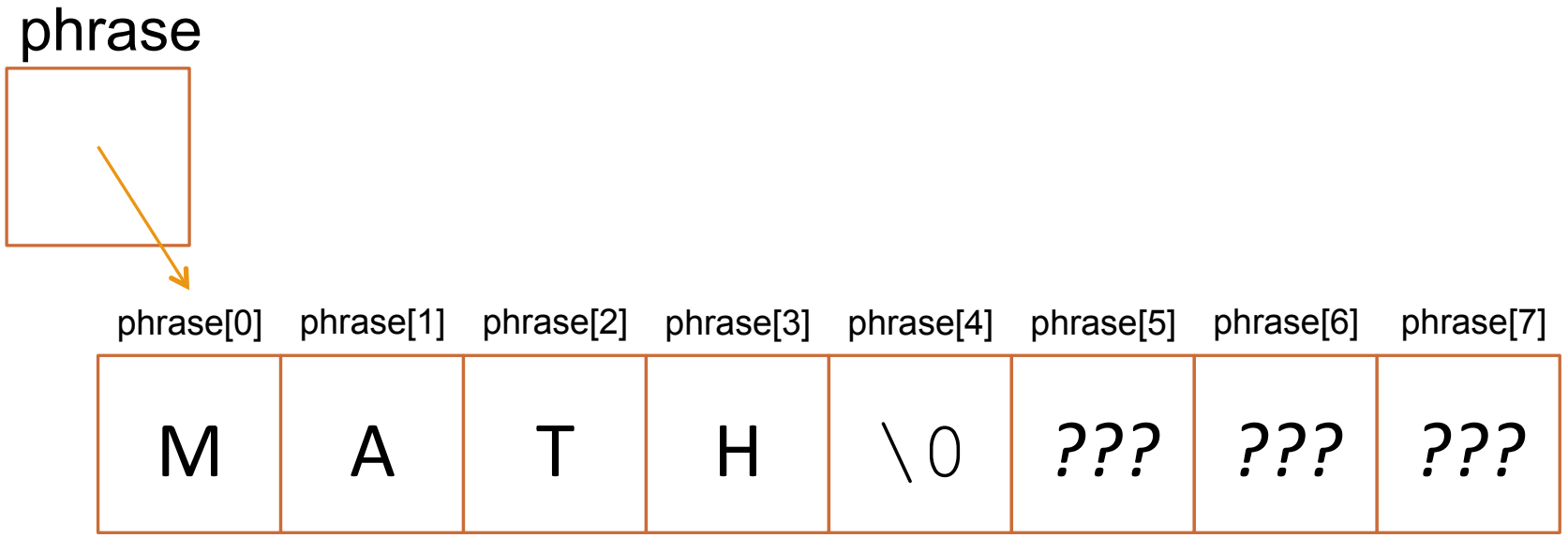
Arrays of Characters

```
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

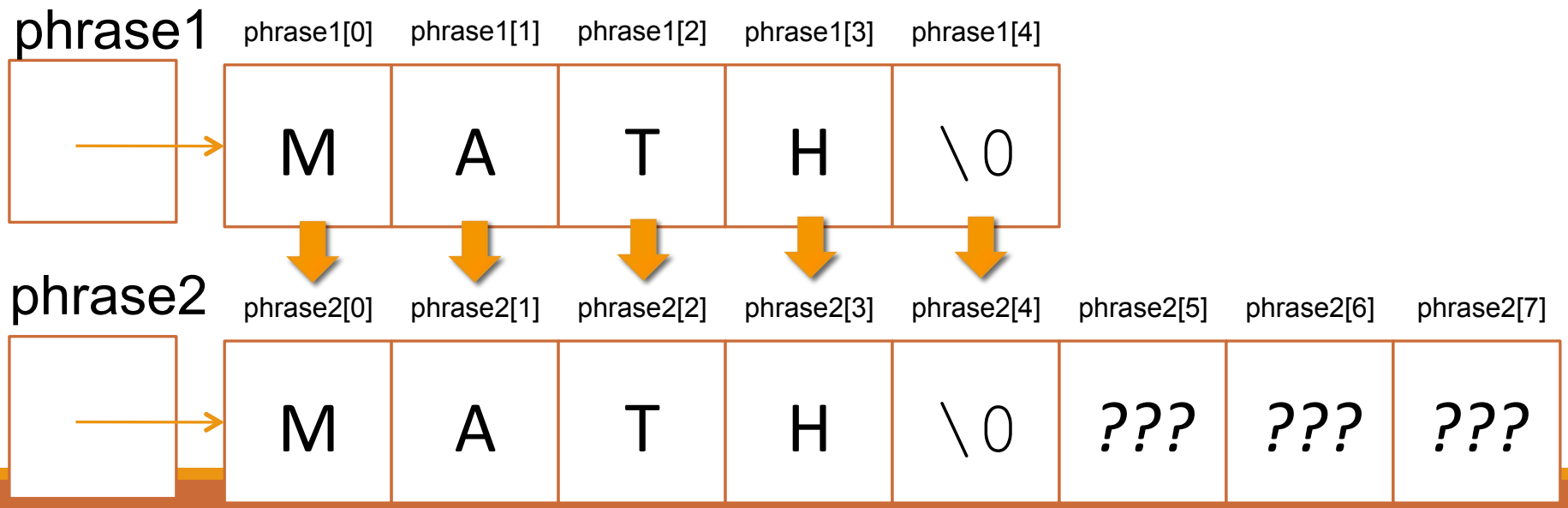
```
#include <string.h>
```

Useful functions

- `strcpy`
- `strcmp` – Google it!
- `strlen` – Google it!
- `strcat`

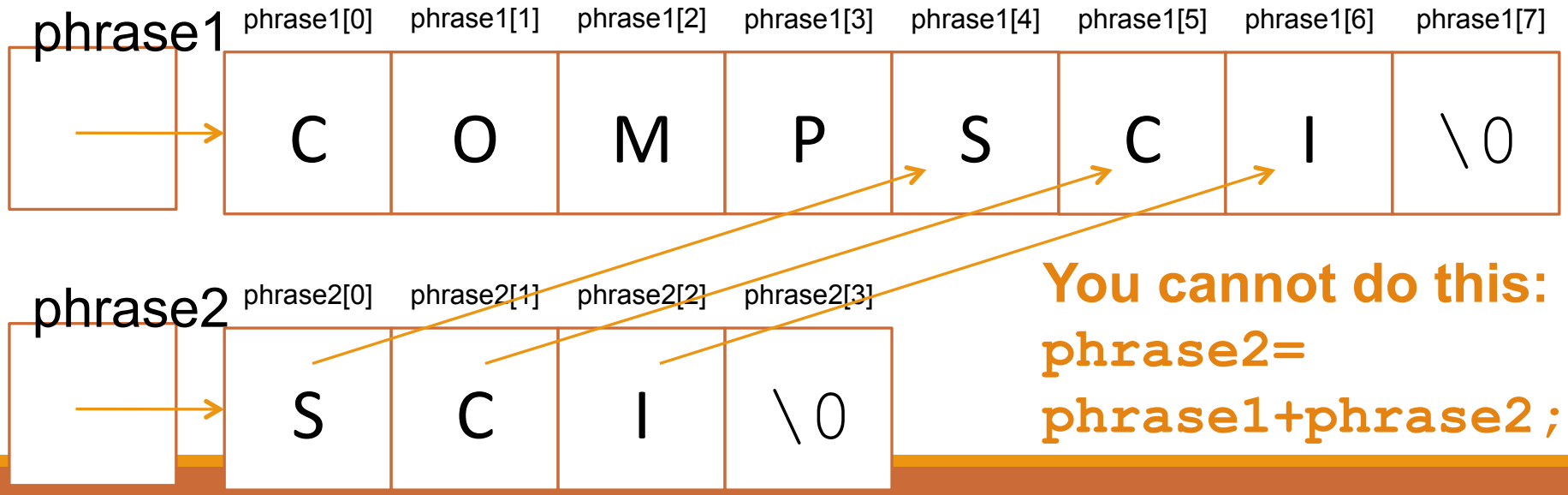
String Copy

```
char phrase1[] = "Math";
char phrase2[8];
strcpy(phrase2, phrase1);
```



String Concatenation

```
char phrase1[8] = "Comp";
char phrase2[] = "Sci";
strcat(phrase1, phrase2);
```



In-Class Participation: String Reversal (Useful for Lab 4)

Assume a character string called `word`. Reverse this string (you can use another character buffer to store the reverse string). For the matrix (note it was part of a structure) in Problem 3, write a C snippet to check if this reverse string is placed horizontally anywhere in the matrix. Feel free to use string functions.

For example, if the word is: elephant, then check if tnahpele is in the matrix.

Next Class

File I/O

Structures and Pointers