Pointers, PT III

1320-Intermediate Programming

University of Texas at Arlington

Lecture Overview

- Quick Review
- Lecture
  - Operators
    - Unary, Binary and Ternary
- Before We Code
  - Arrays of pointers
    - Pointer to an array of pointers
- Sample Programs

QUICK REVIEW

Pass by Value vs Pass by Reference

How would we handle a problem like this?

Create a function that takes two students grades and swaps them.

```
void swap_grades (int g1, int g2)
{
        int temp;

        temp=g1;
        g1=g2;
        g2=temp;

}
```

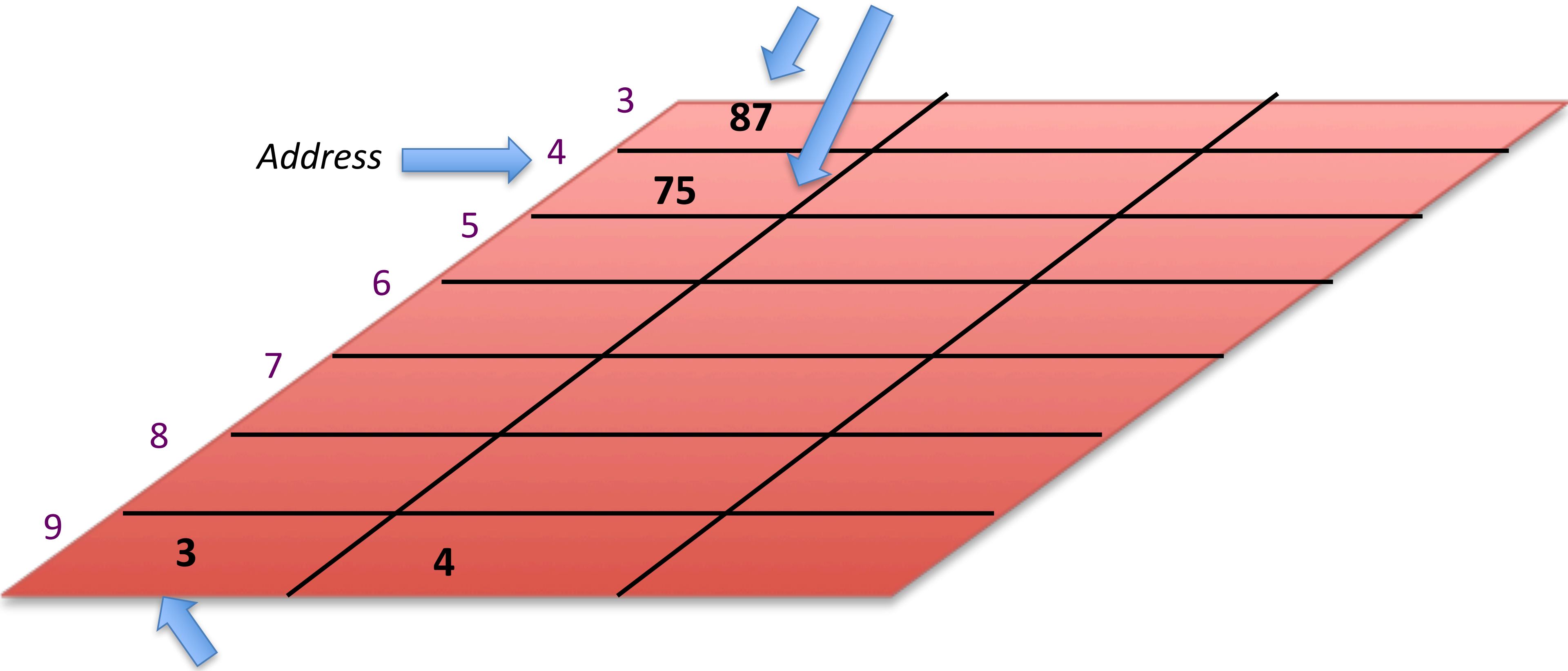
We are modifying copies of g1 and g2, not g1 and g2 themselves.

That means that after the function call is over, the old values of g1 and g2 will remain the same in the actual program

Pass by Value vs Pass by Reference

- By using pointers, we are passing in addresses of variables, not variables themselves
  - In this way, we can overcome the pass by value issue with swapping grades.
  - Note with arrays we are always passing in the addresses-that’s why we don’t have to return anything

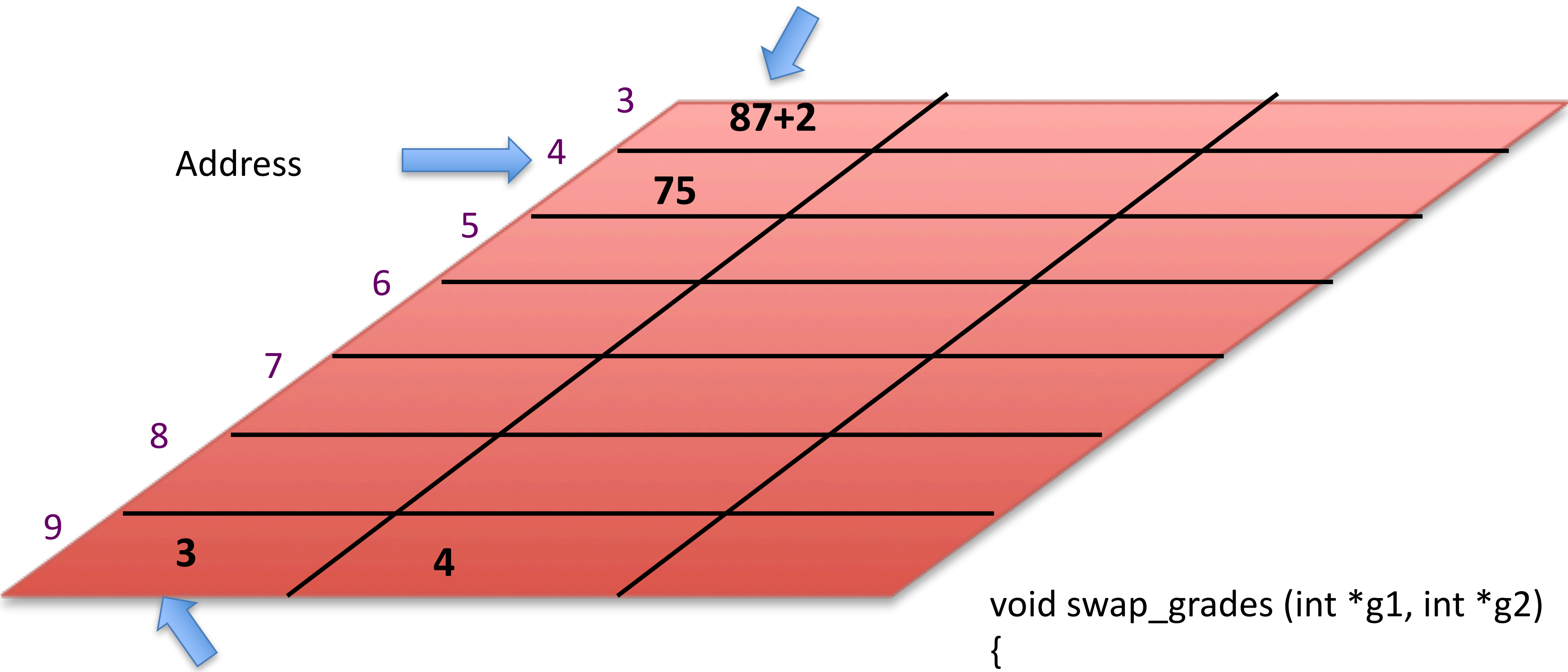
These are your original grades in the program:



We're passing in copies of the addresses of the original values

When we dereference the copies of the addresses, we end up accessing the original values.

These are your original grades in the program:

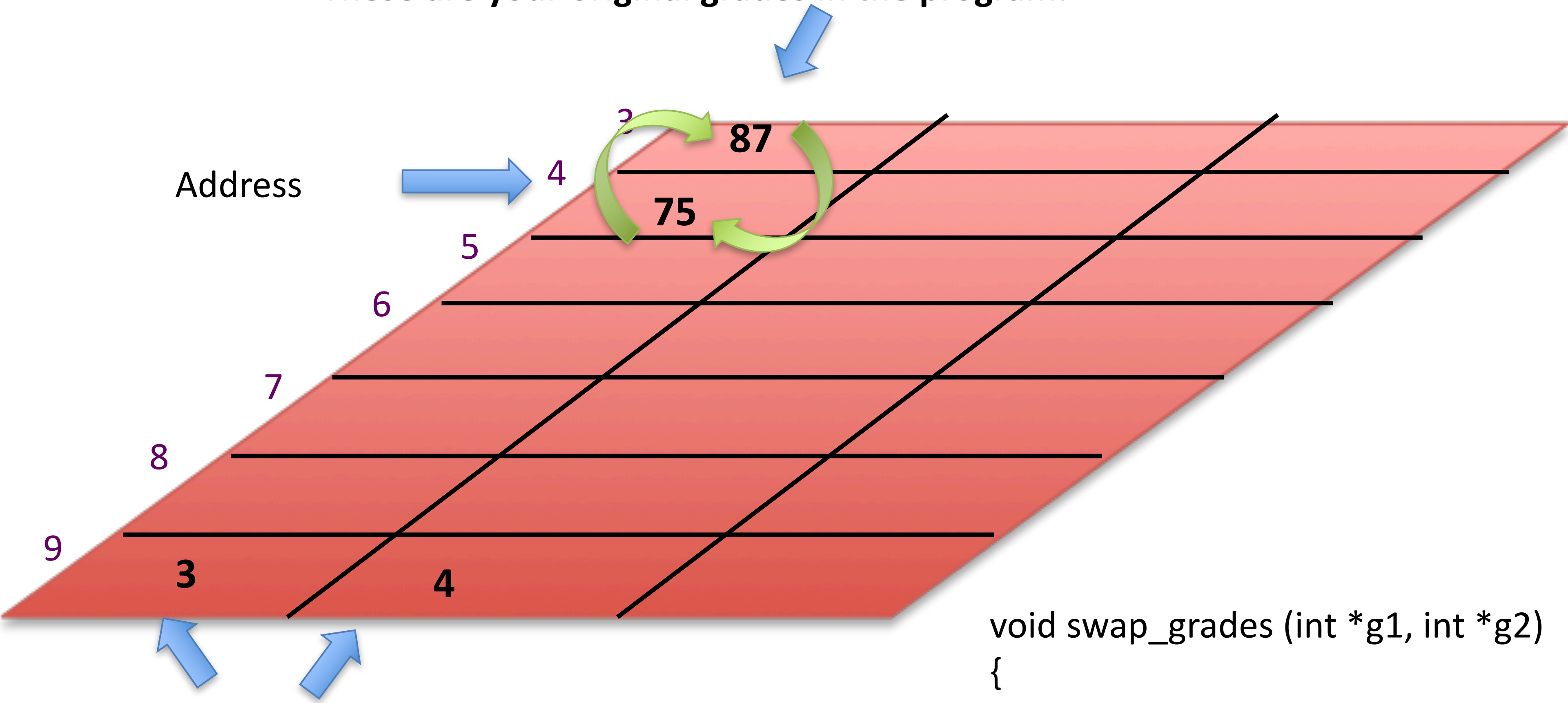


This is g1.
Dereferencing g1 means actually accessing what is at address 3 (the actual value of 87)

For example, if I said: *g1=*g1+2; I am changing the actual value.

```
void swap_grades (int *g1, int *g2)
{
        int temp;

        temp=*g1;
        *g1=*g2;
        *g2=temp;

        }
```

These are your original grades in the program:



This is g1.
Dereferencing g1 means actually accessing what is at address 3 (the actual value of 87)

So when I am swapping, I am now swapping the original values.

```
void swap_grades (int *g1, int *g2)
{
        int temp;

        temp=*g1;
        *g1=*g2;
        *g2=temp;

        }
```

LECTURE

Operators

- By now, you should be familiar with the idea of operators
  - We see them in math and you saw them in 1310
- We know there are different types, and different programming languages support different operators
- Operators in C (look at the website): https://www.tutorialspoint.com/cprogramming/c_operators.htm

Operators

- A universal concept with operators is the number of operands
  - Is this a unary operator?
    - Does it only need one thing to work?
  - Binary operator?
    - Does it need two things to work?
  - Ternary operator?
    - Does it need three things to work?

Operators

- We’ll be using most of these operators shown on this website this semester
- I expect you guys to use this website (or another website you are used to) to refer back to operators
- I won’t go over every single one in class

BEFORE WE CODE

Before We Code

- I'll be showing you some more advanced concepts with pointers today
  - You'll see me use pointers with arrays and arrays of pointers
- As long as you understand the basic concept of what a pointer actually is you will be fine
- You just need practice
  - Drawing out what is happening is pretty helpful

Before We Code

- Before we continue, note that:

```
int num=4;

int *ptr=&num;

int nums[]={3,4};

int *nums[];
```

`int num=4;`	This is an int
`int *ptr=&num;`	This is a pointer at an int
`int nums[]={3,4};`	This is an array containing ints
`int *nums[];`	This is an array containing int pointers. It can also be written like: int **nums

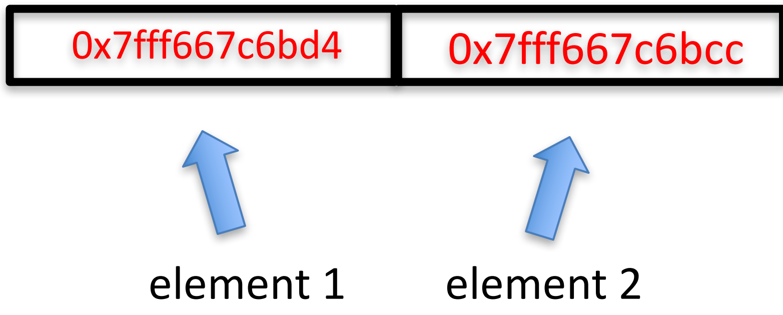
Array of Pointers

```
#include <stdio.h>

int main(int argc, char **argv)
{

        int n[]={3, 7};
        printf("First element( address of n): %p\n", n);
        int n1[]={4, 5};
        printf("First element( address of n1): %p\n", n1);

        int *num_ptr1=n;
        int *num_ptr2=n1;

        int* nums[]={num_ptr1, num_ptr2};

        printf("value: %p, deref: %d\n", nums[0], *nums[0]);
        printf("value: %p, deref: %d\n", nums[1], *nums[1]);

}
```

Output:

```
First element( address of n): 0x7fff667c6bd4
First element( address of n1): 0x7fff667c6bcc
value: 0x7fff667c6bd4, deref: 3
value: 0x7fff667c6bcc, deref: 4
```

nums is an array of pointers (note since an array is known by its address, we could have directly put n and n1)

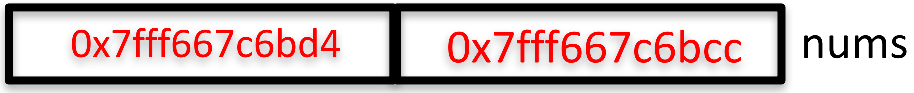


```
#include <stdio.h>

int main(int argc, char **argv)
{

        int n[]={3, 7};
        int n1[]={4, 5};
        int n3[]={90,10, 11};

        int *num_ptr1=n;
        int *num_ptr2=n1;
        int *num_ptr3=n3;

        int* nums[]={num_ptr1, num_ptr2};

        printf("value: %p, deref: %d\n", nums[0], *nums[0]);

        nums[0]=num_ptr3;

        printf("value: %p, deref: %d\n", nums[0], *nums[0]);

}
```

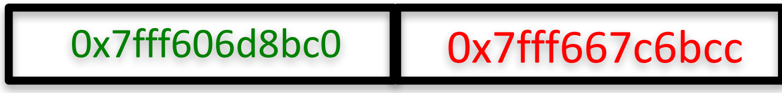
Output:

```
value: 0x7fff606d8bd4, deref: 3
value: 0x7fff606d8bc0, deref: 90
```

NOTE: You can treat an array of pointers like any other array-you change the values out



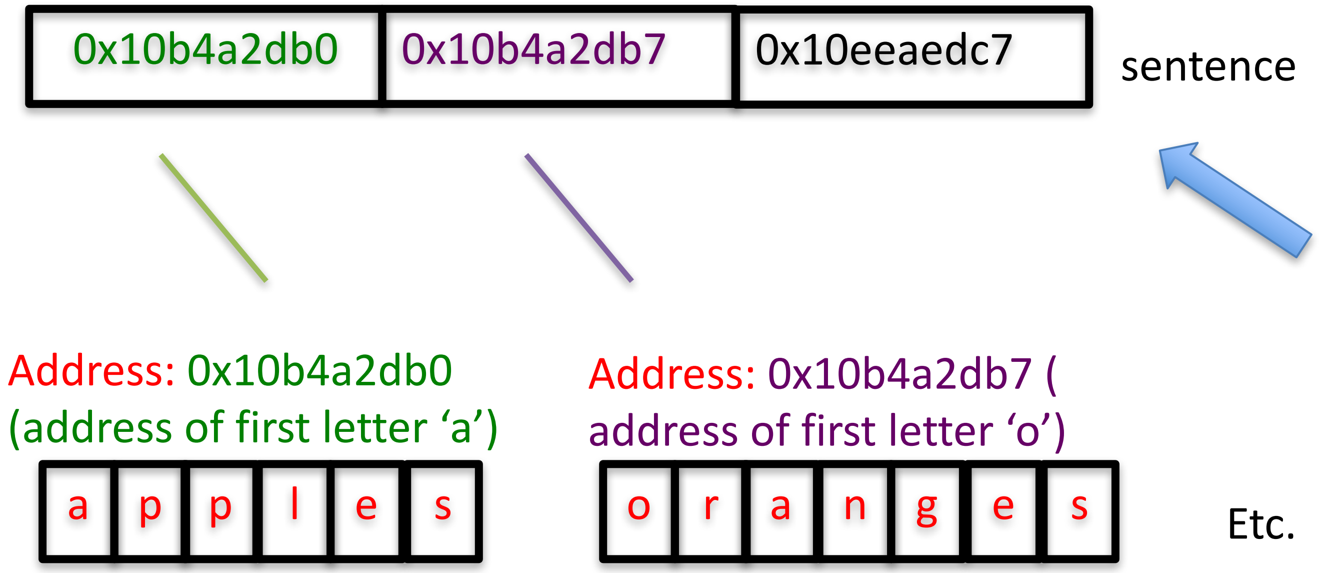
Change the address in the first element of the array of pointers to another address.



Value is changed now

```
char* sentence[]={“apples", ”oranges", ”grapes!”};
```

This is an array of char pointers (char *). The value of each element of the array is an address. Note you could also write char ** sentence.



0x10b4a2db0
0x10b4a2db7
0x10eeaedc7
sentence
Address: 0x10b4a2db0 (address of first letter ‘a’)
Address: 0x10b4a2db7 (address of first letter ‘o’)
a p p l e s
o r a n g e s
Etc.

Each address is the address of a char array (a single string)
Remember, a char array address is just the address of the first letter.

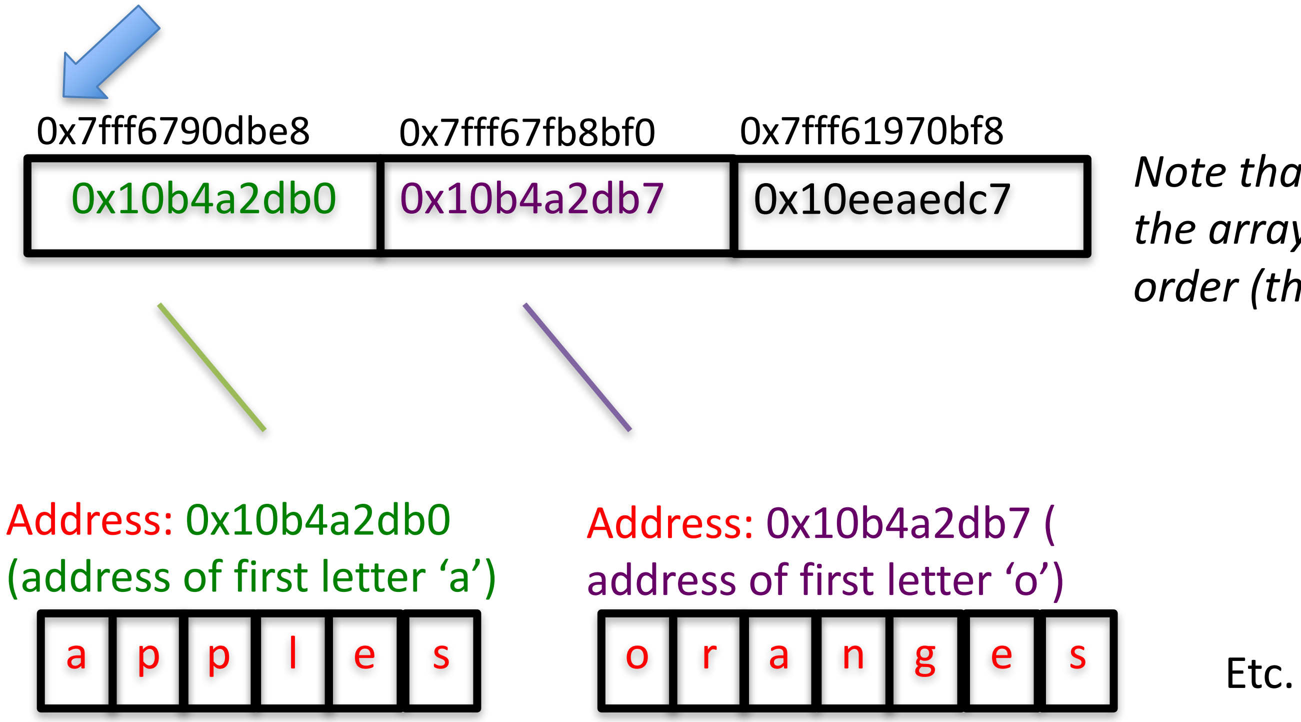
```
printf("value %p, %s\n", sentence[0], sentence[0]);
printf("value %p, %s\n", sentence[1], sentence[1]);
printf("value %p, %s\n", sentence[2], sentence[2]);
```

Output:

```
value 0x10b4a2db0, apples
value0x10b4a2db7, oranges
value 0x101386de3, grapes
```

```
char* sentence[]={“apples", ”oranges", ”grapes!”};
```

Remember a pointer is 8 bytes (on my machine). When we make an array of pointers, the address increments by 8 (size of a pointer)



Note that the values actually stored in the array (addresses) are not in any order (they can be any addresses)

Before We Code

- Now that we know about pointers, we can use more functions!
- Today we will see the function strtok
  - Declaration is in the string.h header file in the C standard library
- We will also see fgets (another way to get input)
  - Declaration is in the stdio.h header file in the C standard library

Before We Code

Function declarations (look them up):

Return type is char pointer (char *)

char *fgets(char *str, int n, FILE *stream)

This parameter is a char pointer (char *)

This parameter is a FILE pointer (FILE *)- for now think of FILE as a type (kind of like how *int* or *char* is a type)

Return type is char pointer (char *)

char *strtok(char *str, const char *delim)

This parameter is a char pointer (char *)

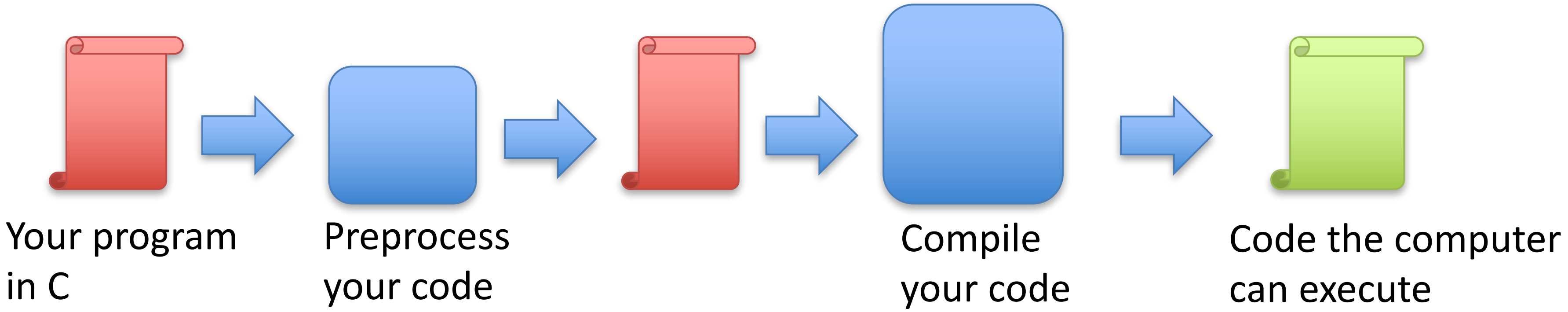
This parameter is a char pointer (char *)- don't worry about **const** for now

As you can see, pointers are EVERYWHERE in C

Before We Code

- We will also see another preprocessor directive (so far we have been using #include)
  - Today we will see #define
- For now, you can think of a preprocessor directive as some steps taken on your code before actually compiling your program
  - Preprocessing your code

(Note: this is a VERY high level view-there are many other details NOT included below)



Before We Code

- More info about preprocessing:

https://www.tutorialspoint.com/cprogramming/c_preprocessors.htm

https://gcc.gnu.org/onlinedocs/gcc-2.95.3/cpp_1.html

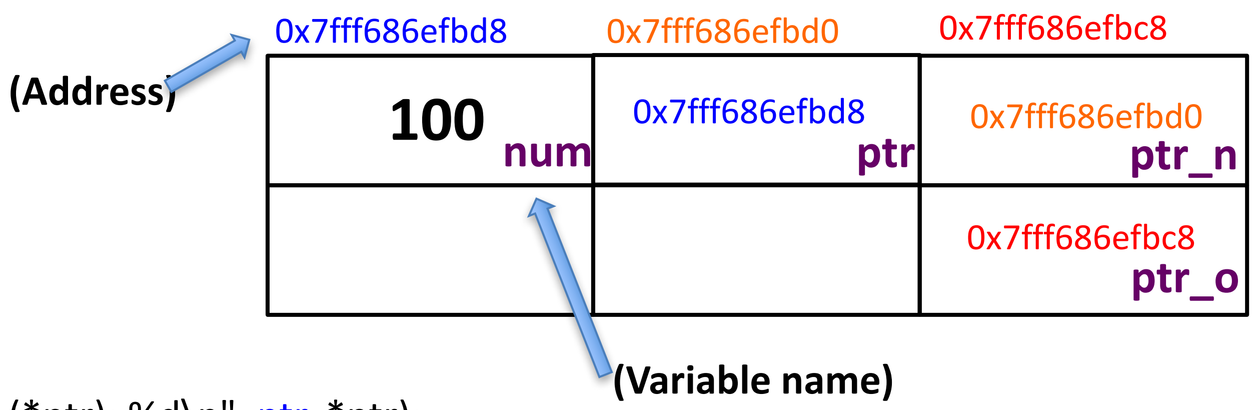
Before We Code

- Pointer to an array of pointers
  - Just like we can have an array of ints or chars, we can also have an array of pointers
    - They’re variables right?   Just like an int can hold different values like 2 and 9, a pointer can hold different values like 0x7fff61c8cb20j and 0x7fff61c8cb0ch
  - To navigate an array of pointers, we need to have a pointer point at the array of pointers
    - We can start it pointing at the first element of the array and then use pointer arithmetic to move through the array

Before We Code

First, make sure you understand this concept of pointers to pointers to pointers:

```
int num=100;
int * ptr=&num;
int **ptr_n=&ptr;
int ***ptr_o=&ptr_n;
```



```
printf("value in ptr: %p,   ptr deref: (*ptr): %d\n", ptr, *ptr);
printf("value in ptr_n: %p, ptr_n deref: (*ptr_n): %p\n", ptr_n, *ptr_n);
printf("value in ptr_o: %p, ptr_o deref: (*ptr_o): %p\n", ptr_o, *ptr_o);
```

Output:

```
value in ptr: 0x7fff686efbd8,   ptr deref: (*ptr): 100
value in ptr_n: 0x7fff686efbd0, ptr_n deref: (*ptr_n): 0x7fff686efbd8
value in ptr_o: 0x7fff686efbc8, ptr_o deref: (*ptr_o): 0x7fff686efbd0
```

Navigating an array of pointers-the value of the pointer is changing

```
#include <stdio.h>

int main(int argc, char **argv)
{
	char * french_words[]={"amour", "soleil", "oiseau", "coeur"};
	char **ptr_fr=french_words;

	int i;

	for(i=0;i<4;i++)
	{
		printf("Word: %s  Address: %p\n", *ptr_fr, ptr_fr);
		ptr_fr++;
	}
}
```

This is a pointer to an array of pointers. Since the an array is referred to by its first element (in this case the elements are pointers), this is technically a pointer to a pointer (like the previous slide)

Output:

```
Word: amour  Address: 0x7fff6f299b68
Word: soleil  Address: 0x7fff6f299b70
Word: oiseau  Address: 0x7fff6f299b78
Word: coeur  Address: 0x7fff6f299b80
```

-Notice the actual value of ptr_fr changes

-Also notice it is incrementing by 8 bytes (the size of an address on my computer)

```c
#include <stdio.h>

int main(int argc, char **argv)
{
	char * french_words[]={"amour", "soleil", "oiseau", "coeur"};
	char **ptr_fr=french_words;

	int i;

	for(i=0;i<4;i++)
	{
		printf("Word: %s  Address: %p\n", *(ptr_fr+i), ptr_fr);

	}
}
```

Navigating an array of pointers-the value of the pointer is NOT changing

We're just adding the value of i to the pointer to print out

Output:

```
Word: amour  Address: 0x7fff69bdeb68
Word: soleil  Address: 0x7fff69bdeb68
Word: oiseau  Address: 0x7fff69bdeb68
Word: coeur  Address: 0x7fff69bdeb68
```

Notice the actual value of ptr_fr DOES NOT change.

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int num_of_args=argc;
    int i=0;

    while (i<num_of_args)
    {
        printf("%s\n", argv[i]);
        i++;
    }
}
```

Notice the second parameter of the main function is a pointer to a pointer
-it is actually pointing to string arguments (char arrays) like the previous slide

If we run:

```
computer$ ./a.out    first    second
```

argv[0] argv[1] argv[2]

Output:

```
./a.out
first
second
```

SAMPLE PROGRAM

Sample Programs

- Array of pointers
- Pointer arithmetic on an array of pointers
  - Same idea as last class
- Sample word problem (next slide)

Program 1

Aimée wants a user to enter a sentence (don’t worry about capitalization) and check whether it is French or English. All she has a small dictionary of words:

English	**French**
love	amour
sun	soleil
bird	oiseau
heart	coeur

Program 1

- It is up to the programmer to decide how to solve this problem
- One way could be to check if a sentence entered contains any French words
  - If it does, we can assume it is a French sentence
  - Otherwise, we can assume it is English
- Obviously, in a real application more details would be considered
  - I also won’t consider things like French words used in English sentences: faux pas for example