CS101C: Introduction to Programming (Using C)

Autumn 2025

Nikhil Hegde
Achyut Mani Tripathi

Week6: More multidimensional Arrays, Pointers

Last class (4/9/2025)

- Recap of multidimensional arrays
- Demo program on reversing an array in multiple ways.

    1. Using a duplicate array (recall: `arr2[10]`)
    2. Using just one temporary variable (recall: `tmp=arr[i]`)

        - With and without multiple expressions Expression1/E1 and Expression2/E2 of for loop (recall: `for(i=0,j=9;i<5 && j>=5;i++,j--)`)

    3. Not using any temporary variable (recall: `a=a+b;b=a-b;a=a-b;`)

- Multiple Choice Questions (MCQ3.c posted in Google Classroom) covering arrays and loops.

Today’s class (8/9/2025)

- Demo program with multidimensional arrays
  a. Transpose of a Matrix of integers
- Pointers

Transpose of a Matrix

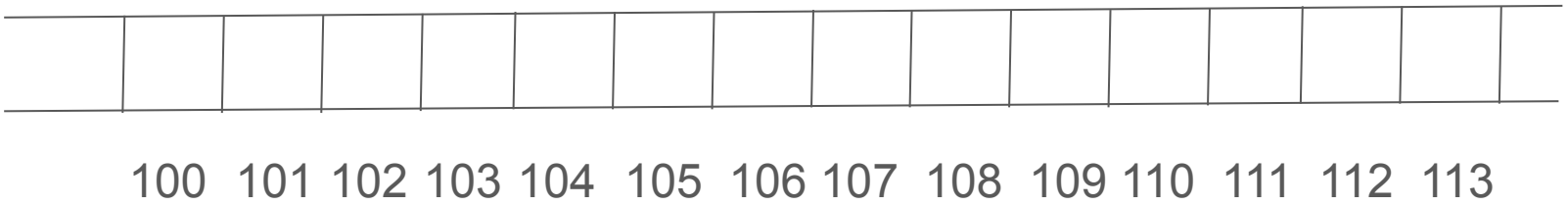
- Initialize 2D Array of size 3

```
for(int i = 0; i < 3; i++){
    for(int j = i; j < 3; j++){
        int tmp = a[i][j];
        a[i][j] = a[j][i];
        a[j][i] = tmp;
    }
}
```

- Print transposed array

Pointers

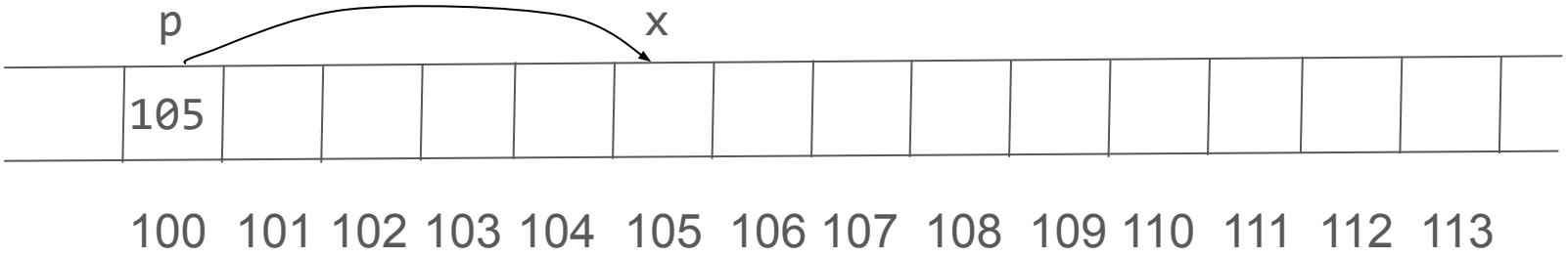
- A pointer is a variable that holds the address of another variable
- Machine memory consists of consecutively numbered cells (analogy: boxes). These numbers are addresses. Each cell is of width = 1 byte.



- Two consecutive cells = `short`, 4 consecutive cells = `int`, one cell = `char`

Pointers

- If p is a pointer that holds the address of a char x



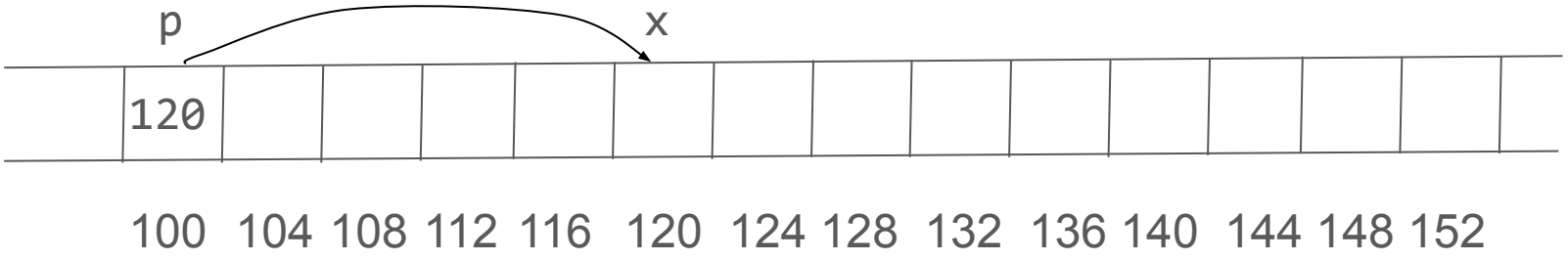
- How do we define the pointer p? How do we initialize it to hold an address i.e. how do we get the addresses?

Use the & (“address of”) operator to get the address of x

```
char x='A';

char* p; p=&x;
```

Pointers

- If p is a pointer that holds the address of a int x



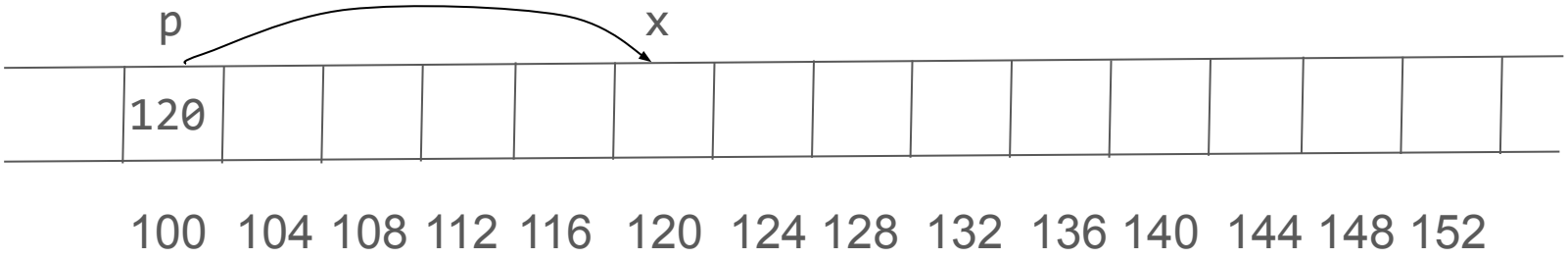

```
int x=1234;

int* p;

p=&x;
```

Note:

- & operator can't be applied to expressions and constants
- Pointers always point to specific data types (e.g. `int` and `char` in previous slides). Exception: `void` type

Pointers

- * is the indirection or dereference operator

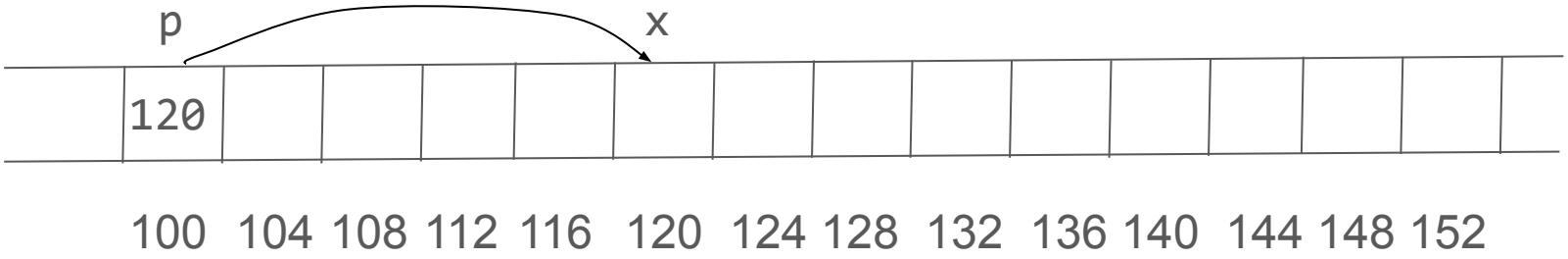


- When applied to a pointer, it gives you the <u>value at the address pointed to by the pointer</u>

```
int x=1234;

int* p;

p=&x; int y=*p;    // Now you can use *p wherever you wish to use x.
```

Pointers

- * is the indirection or dereference operator



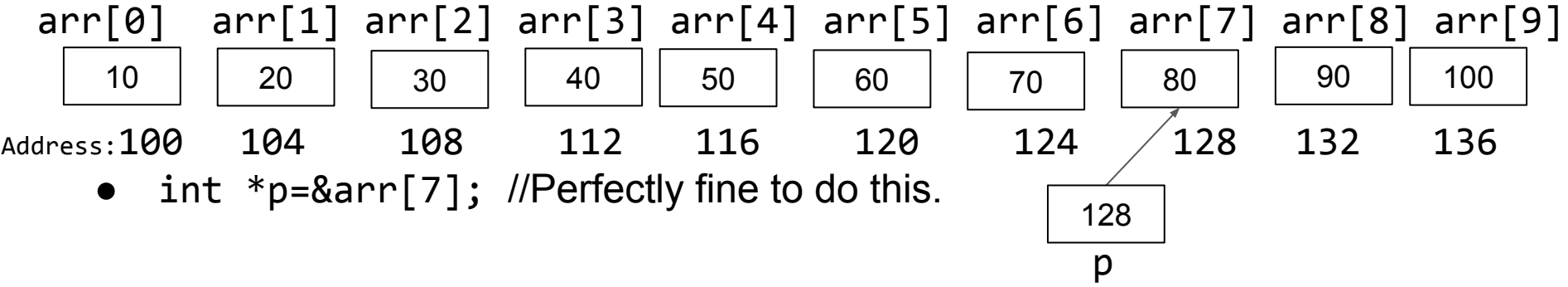
- When applied to a pointer, it gives you the <u>value at the address pointed to by the pointer</u>

```
int x=1234;

int* p;

p=&x;          // Now you can use *p wherever you wish to use x.

int y=*p;      // *p = *p +10; ++*p; what is the value of x after this?
```

Pointers and Arrays

- Pointers and Arrays share a strong relationship
- Recall:

```
int arr[10]={10,20,30,40,50,60,70,80,90,100}
```



- `int *p=&arr[7];` //Perfectly fine to do this.

- *p gives you the value 80. E.g. `printf("%d",*p);`
- p+1, by definition, points to the next element of the array, arr[8]. p-1 points to the previous element, arr[6]. `printf("%d",*(p+1));`

Today’s class (10/9/2025)

- More Pointers
- ~~Application of pointers~~
  a. ~~Modular programming (functions)~~
- Pointer to character arrays

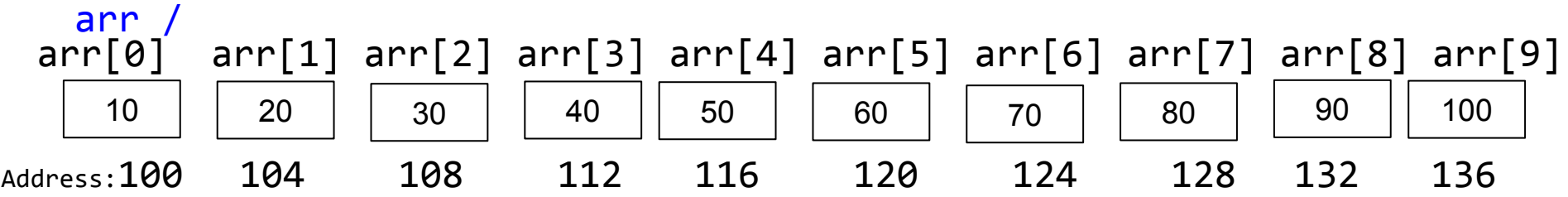
Exercise:

- What does *p++ give you?

Pointers and Arrays

- Array name is a synonym for the location of the first element i.e.

```
int arr[10]={10,20,30,40,50,60,70,80,90,100}
```



- How do you get the location of first element? `&arr[0]`
- How do you initialize a pointer to this location?

```
int * p= &arr[0];
```

Alternatively: `int *p=arr;`

Go through pointer1.c and pointer2.c shared in the code examples.

Recap: Pointers and Arrays

- Can we call an array of characters as String?

```
int main(){
    char arr[5]={'H','E','L','L','O'};
    char* ptrC=arr;
    for(int i=0;i<5;i++){
        printf("%c",*ptrC);
        ptrC++;
    }
    printf("%s",ptrC);
}
```

Recall: string is a sequence of characters
So, is `arr` same as the string “HELLO”

- `%s` is format specifier to print strings.

Is this `printf` statement safe?

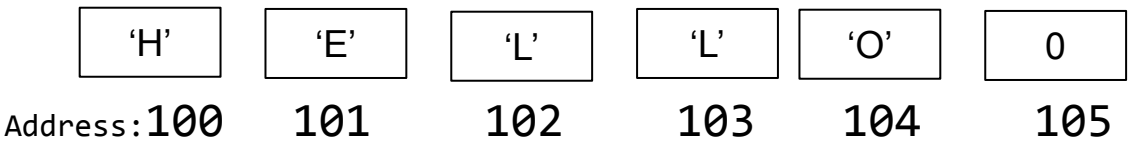
Recap: Pointers and Arrays

- Can we call an array of characters as String?

```
int main(){
    char arr[5]={'H','E',300,'L','O'};
    char* ptrC=arr;
    printf("%s",ptrC); //Is this safe? What does this print?
}
```

Recap: Pointers and Arrays

- What is a String and how does it differ from character array?
  - A string has an invisible NULL character at the end of the sequence of characters. The NULL character is also called as terminator.
  - When we refer to string “HELLO”, the sequence of characters consists of ‘H’, ‘E’, ‘L’, ‘L’, ‘O’, and 0



Recap: Pointers and Arrays

- Wait. How can 0 be part of a sequence of characters? 0 is an integer. ???
- **Recall:** a char is of size one byte in memory. The binary representation/equivalent of a character is decided as per the ASCII table. E.g. ‘A’ = 0x41. ‘0’=0x30 etc. ?=0x0
- When we print a string using format specifier %s, the printf peels of one character after another and prints until a 0 is encountered.

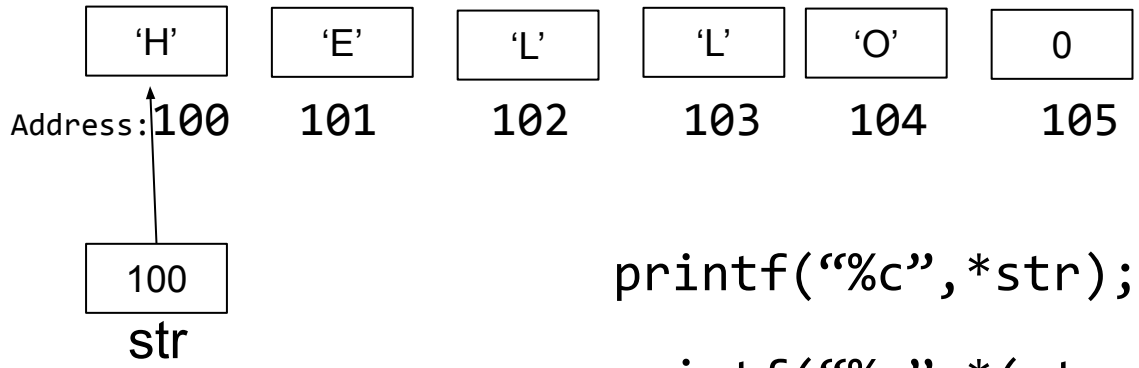
Today’s class (12/9/2025)

- More Pointers
- ~~Application of pointers~~
  a. ~~Modular programming (functions)~~
- Pointer to character arrays and string constants

Pointers and Strings

- When we refer to string “HELLO”, the sequence of characters consists of ‘H’, ‘E’, ‘L’, ‘L’, ‘O’, and 0
  - We can refer to this sequence using a pointer:

char *str=”HELLO”;

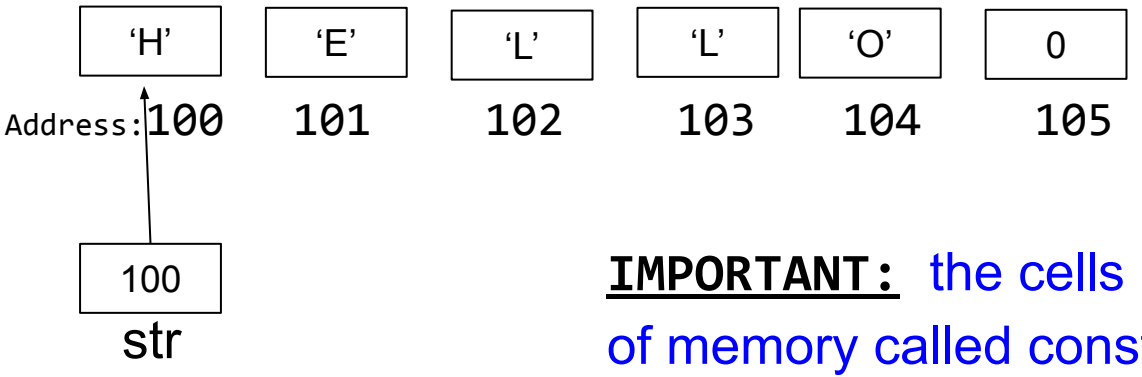



```
printf(“%c”,*str); //prints H

printf(“%c”,*(str+1)); //prints E
```

Pointers and Strings

- When we refer to string “HELLO”, the sequence of characters consists of ‘H’, ‘E’, ‘L’, ‘L’, ‘O’, and 0
  - We can refer to this sequence using a pointer:

```
char *str=”HELLO”;
```

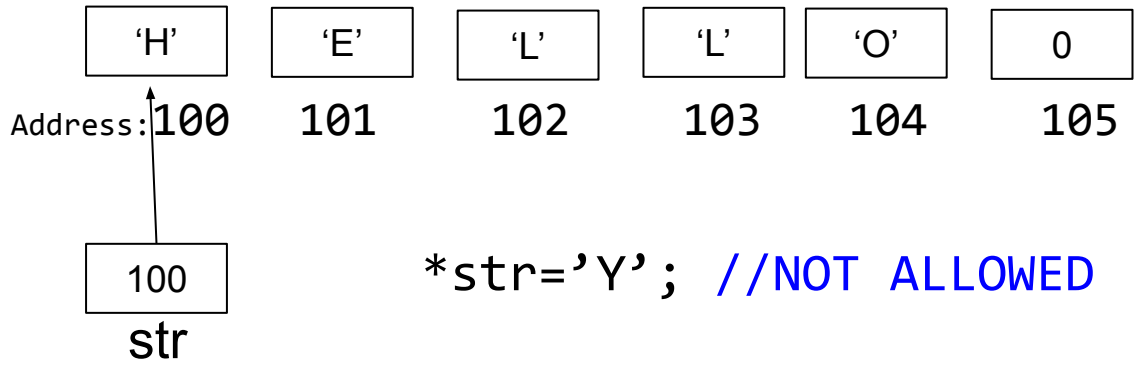


IMPORTANT: the cells belong to a region of memory called constant memory

Pointers and Strings

- the cells belong to a region of memory called constant memory

```
char *str=”HELLO”;
```

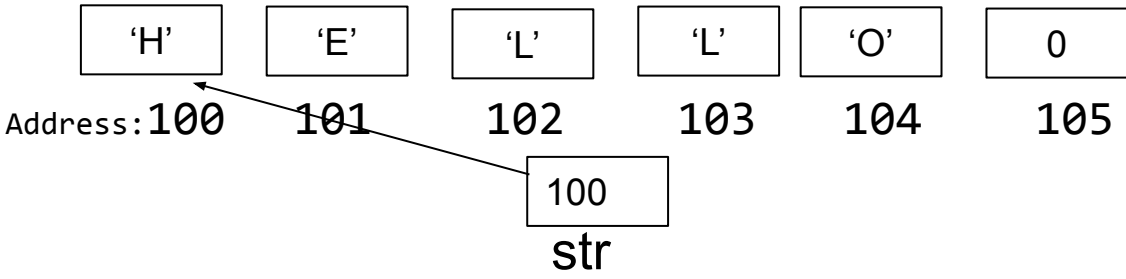


Pointers and Strings - Example

```
int main(){
    char *str;
```

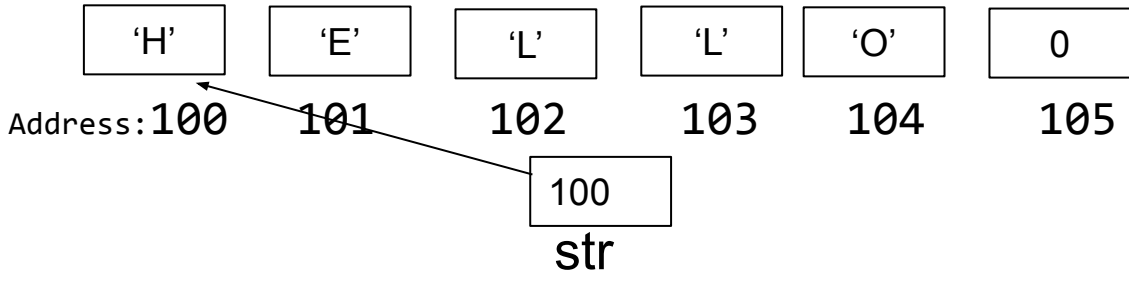
Pointers and Strings - Example

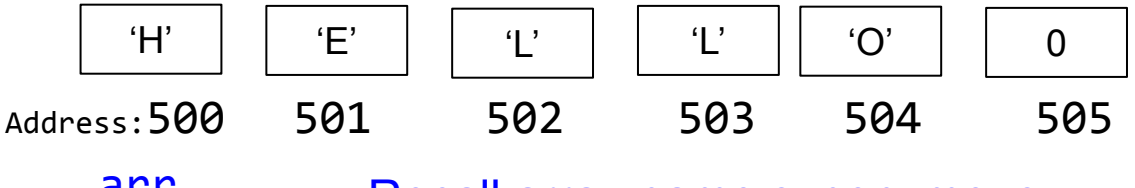
```
int main(){
    char *str;
    str = “HELLO”;
```



Pointers and Strings - Example

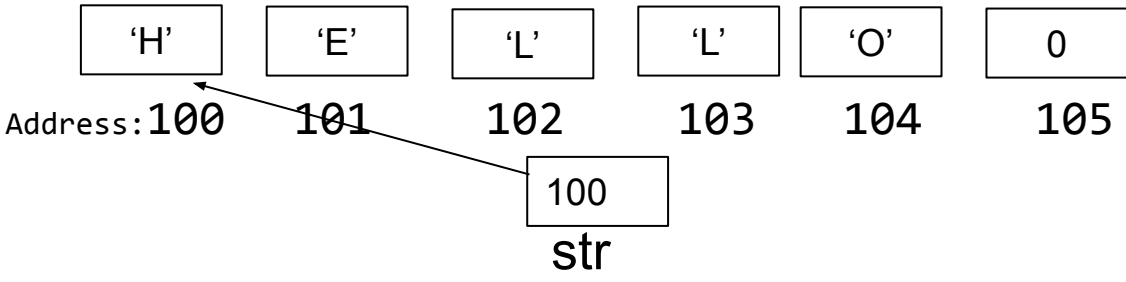
```
int main(){
    char *str;
    str = “HELLO”;
    char arr[6]={‘H’,’E’,’L’,’L’,”O”};
```



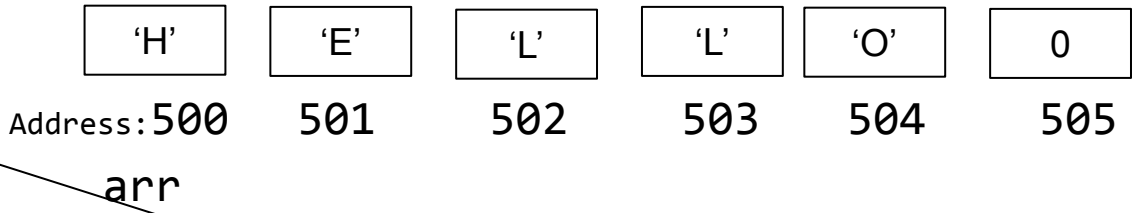


Recall:array name synonymous with location of first element.
Also notice that array has 6 cells reserved. The sixth one is not explicitly initialized in the initializer list. So, 0 is put in the 6th cell.

Pointers and Strings - Example

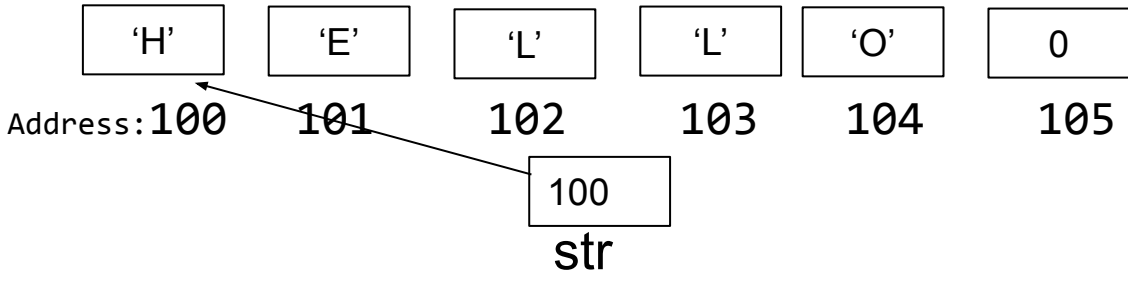



```
int main(){
    char *str;
    str = "HELLO";
    char arr[6]={'H','E','L','L',"O"};
    *str = 'Y';
```



This is legal syntax. Why? Refer to slide 8: `int x=1234; int* p; p=&x; int y=*p;` //use *p wherever you wish to use x.

Pointers and Strings - Example



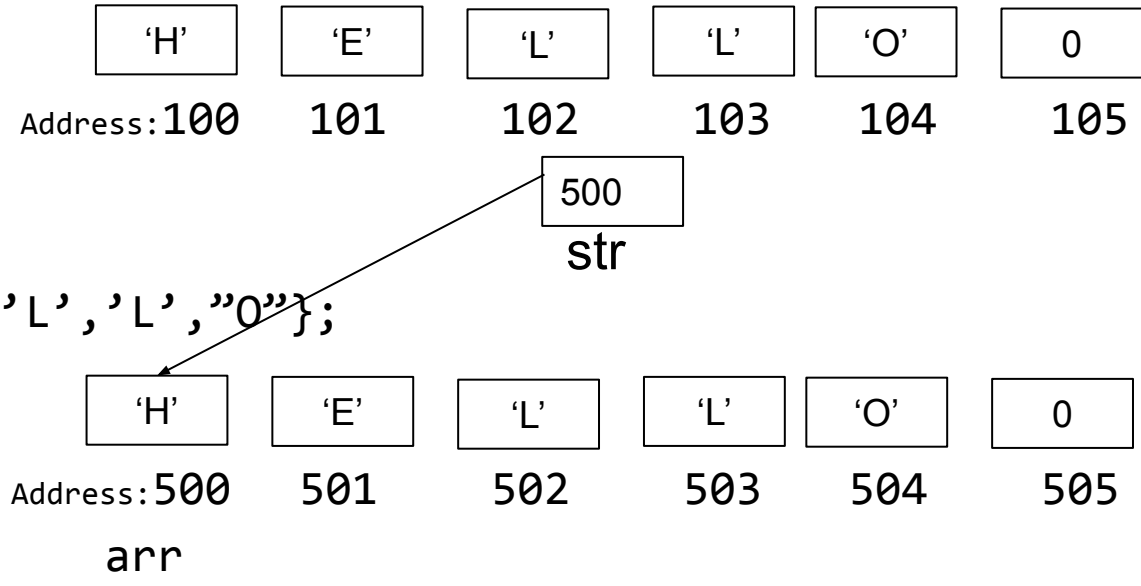

```
int main(){
    char *str;
    str = "HELLO";
    char arr[6]={'H','E','L','L',"O"};
    *str = 'Y';
```

But we see a segmentation fault if we do this assignment. Because address 100 is in constant memory: string constants are allocated cells in constant memory.

This is legal syntax. Why? Refer to slide 8: `int x=1234; int* p; p=&x; int y=*p;` //use *p wherever you wish to use x.

Pointers and Strings - Example

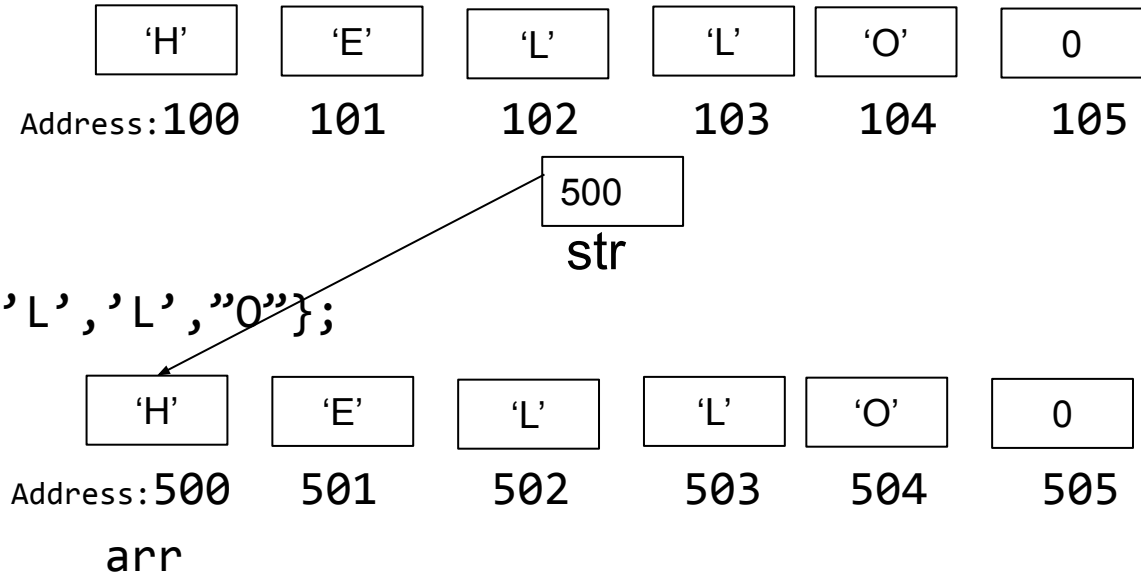
```
int main(){
    char *str;
    str = “HELLO”;
    char arr[6]={‘H’,’E’,’L’,’L’,”O”};
    //*str = ‘Y’;
    str=arr;
```



We assign/overwrite a different value inside str’s cell. Now str contains the address of the first element of arr.

Pointers and Strings - Example

```
int main(){
    char *str;
    str = “HELLO”;
    char arr[6]={‘H’,’E’,’L’,’L’,”O”};
    //*str = ‘Y’;
    str=arr;
    printf(“%s”,str);
```



Printf starts peeling one character after another starting from address 500 until 0 is encountered.

Go through string.c shared in the code examples.

Next week

- ~~Application of pointers~~
  a. ~~Modular programming (functions)~~
- Call-by-value, Call-by-reference
- Global variables, static variables
- Sorting

Functions

- You have seen functions `main, printf, scanf, pow` (anybody?)

```
int main()

printf(“My name is %s”,name);

scanf(“%d”,&x);
```

Return values, function name, function arguments / parameters

Functions

- Let us define our own function to swap.

```
void swap(int a, int b){
   int tmp = a;
   a = b;
   b=tmp;
   return;
}
```

Function parameters, return statement, void type

Functions

- Let us call the function swap from main.

```
int main(){
   int a=10;
   int b=20;
   swap(a, b);
   printf(“a=%d b=%d”,a,b);
}
```

Function call, call site.