Strings - Basics

December 12, 2022

Today

- Array Initialization

- Introduction to strings
  - Syntax
  - Basic I/O
  - Simple usage

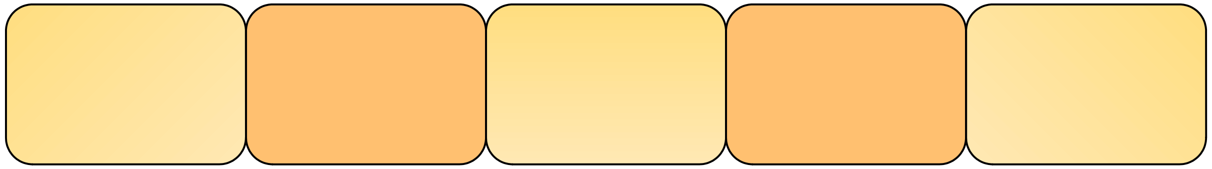
Reminder: Arrays

An array in C is defined similar to defining a variable.

int a[5];

The square parenthesis [5] indicates that a is not a single integer but an array, that is a consecutively allocated group, of 5 integers.

It creates five integer boxes or variables



a[0] a[1] a[2] a[3] a[4]

Array elements are consecutively allocated in memory.

The boxes are addressed as a[0], a[1], a[2], a[3] and a[4]. These are called the elements of the array.

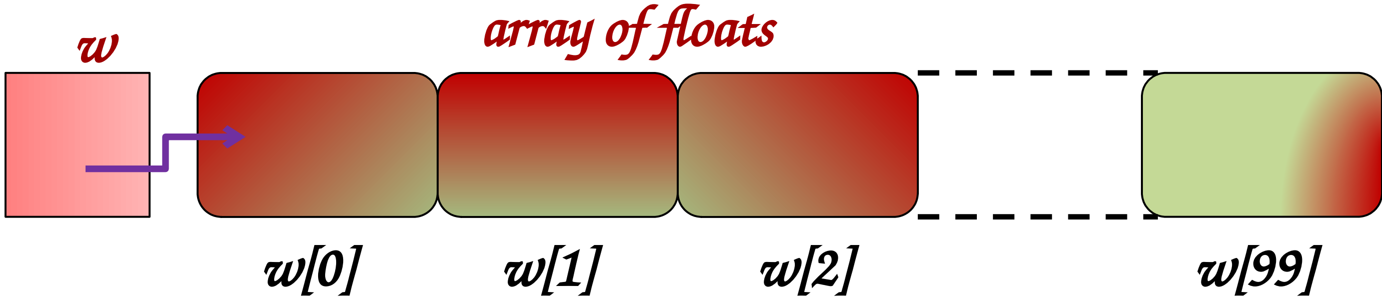
Recap about Arrays

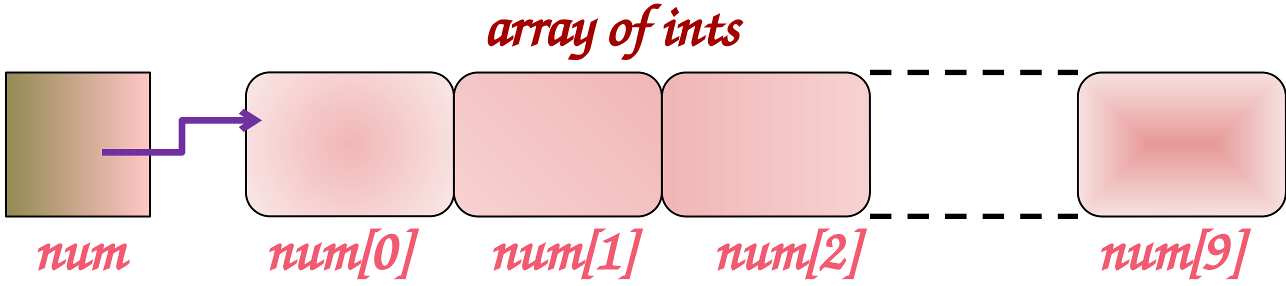
Basics: Arrays are defined as follows.

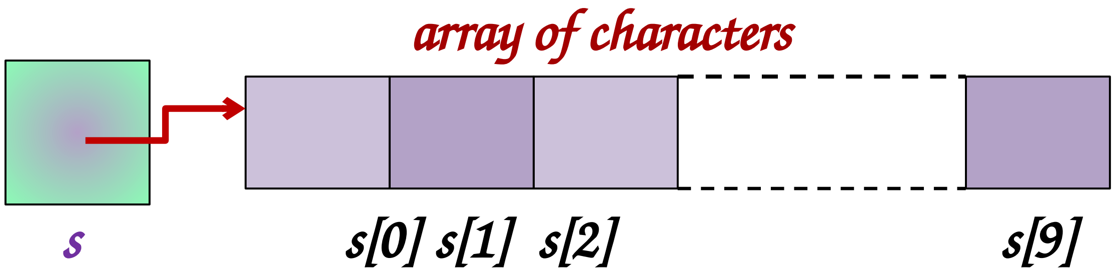
```
float w[100];
int num[10];
char s[10];
....
```

float w[100] defines 100 variables of type float. Their names are indexed: w[0],w[2],…w[99]

It also defines a variable called w which stores the address of w[0].







Reading into Arrays

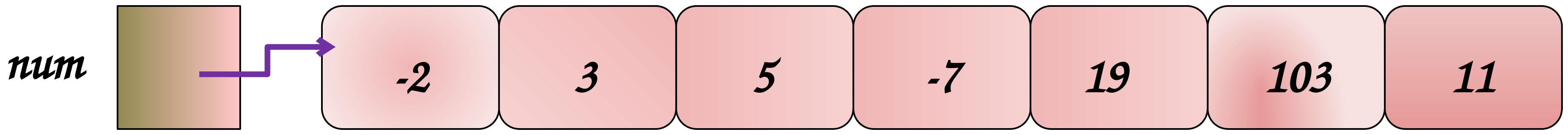
Read N numbers from user directly into an array

```
#include <stdio.h>
int main() {
   char word[10];
   for (i=0; i<10; i=i+1) {
        scanf(“%c”, &word[i]);
   }
   return 0;
}
```

scanf can be used directly, treat an array element like variable of the same data type.

1. *For reading elements of a char array s[], use scanf(“%c”, &s[j]).*
2. *This is a really clunky way of reading text. No?*

How can we create an int array num[] and initialize it to:



Method 1

```
int  num[] = {-2,3,5,-7,19, 103, 11};
```

1. Initial values are placed within curly braces separated by commas.
2. The size of the array need not be specified. It is set to the number of initial values provided.
3. Array elements are assigned in sequence in the index order. First constant is assigned to array element [0], second constant to [1], etc..

Method 2

```
int  num[10] = {-2,3,5, -7, 19, 103, 11};
```

Specify the array size. size must be at least equal to the number of initialized values. Array elements assigned in index order. Remaining elements are set to 0.

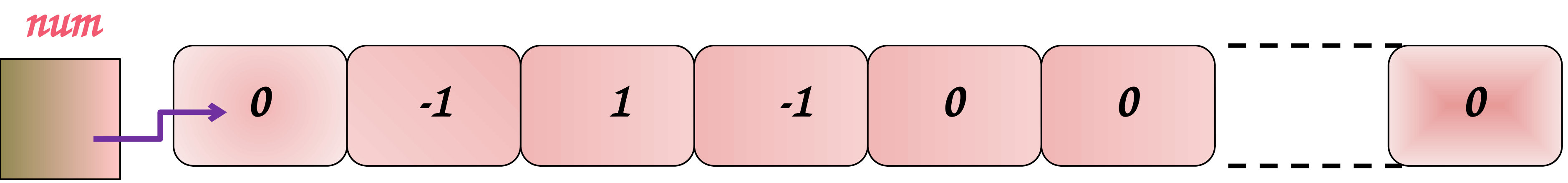
Recommended method: array size determined from the number of initialization values.

```
int  num[] = {-2,3,5,-7,19,103,11};
```

Is this correct?

```
int num[100] ={0,-1,1,-1};
```

YES! Creates num as an array of size 100. First 4 entries are initialized as given. num[4] … num[99] are set to 0.



Is this correct?

NO! it is wrong compiler warning!

```
int  num[6] = {-2,3,5,-7,19,103,11};
```

Why?

1. num is declared to be an int array of size 6 but 7 values have been initialized.
2. Number of initial values must be less than equal to the size specified.

Initialization values could be constants or constant expressions. Constant expressions are expressions built out of constants.

```
int num[] = { 109, 7+3, 25*1023 };
```
✓

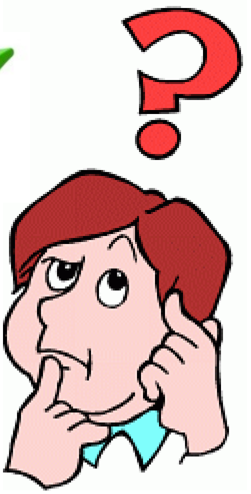
Type of each initialization constant should be promotable/demote-able to array element type.

E.g.,
```
int num[] =  { 1.09, 'A', 25.05};
```
✓

Float constants 1.09 and 25.05 downgraded to int

Would this work?

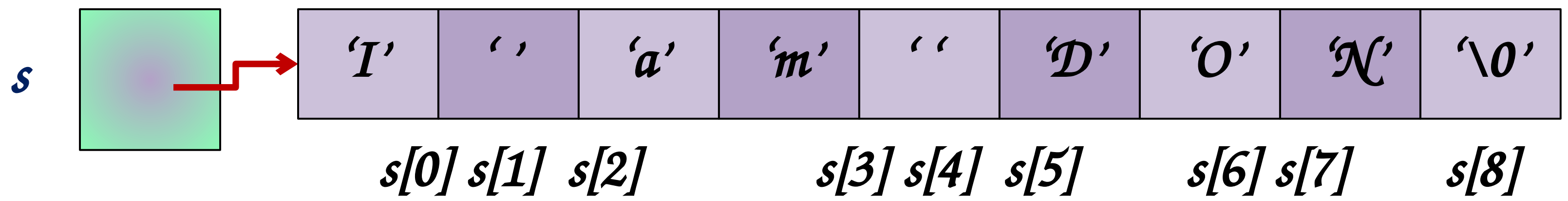
```
int curr = 5;
int num[] = { 2, curr*curr+5};
```
✓



YES! ANSI C allows constant expressions AND simple expressions for initialization values. "Simple" is compiler dependent.

Character Array Initialization

Character arrays may be initialized like arrays of any other type. Suppose we want the following char array.



We can write:

```
s[]={'I',' ','a','m',' ','D','O','N','\0'};
```

BUT! *C allows us to define* ***string constants.***
We can also write:

```
s[] = "I am DON";
```

1. *"I am DON" is a* ***string*** *constant. Strings constants in C are specified by enclosing in double quotes.*
2. *It is equivalent to a character array* ***ending*** *with '\0'.*
3. *The '\0' character (also called NULL char) is automatically added to the end.*

Printing Strings

We have used string constants many times. Can you recall?

printf and scanf: the first argument is always a string.

1. printf("The value is %d\n", value);
2. scanf("%d", &value);

Strings are printed using %s option.

E.g. 1

```
printf("%s", "I am DON");
```

Output

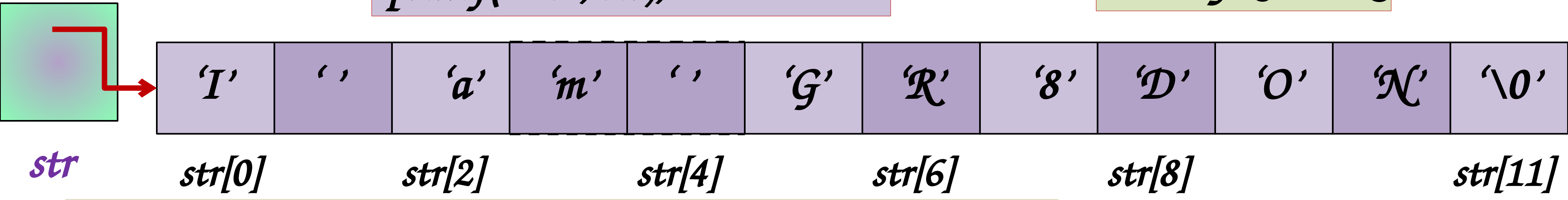
```
I am DON
```

E.g. 2

```
char str[]="I am GR8DON";
printf("%s", str);
```

Output

```
I am GR8DON
```



State of memory after definition of str in E.g. 2. Note the NULL char added in the end.

This NULL char is not printed.

Strings

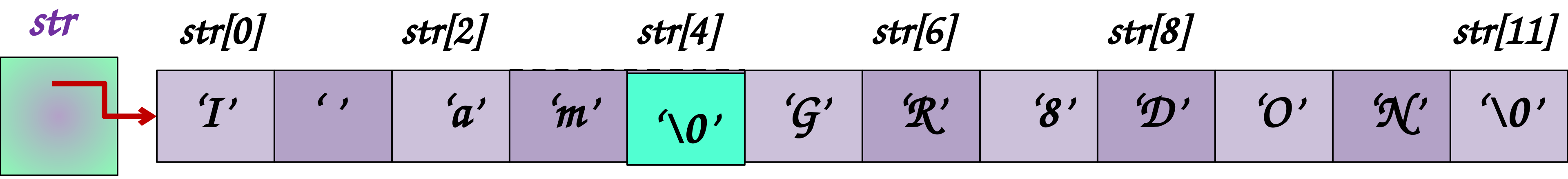
Consider the fragment.

```
char str[]="I am GR8DON";
str[4] = '\0';
printf("%s", str);
```

This defines a constant string, i.e., character array terminated by, but not including, '\0'.

What is printed?

Let us trace the memory state of str[].



Output

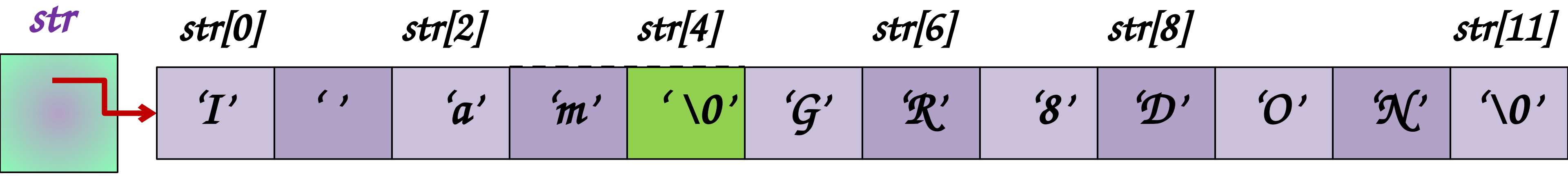
I am

1. A string is a sequence of characters terminated by '\0'. This '\0' is not part of the string.

2. There may be non-null characters after the first occurrence of '\0' in str[]. They are not part of the string str[] and don't get printed by printf("%s", str);

So did we lose the chars after the first '\0' ? Where did they go?

Of course not, they remain right where they were. They were not printed because we used %s in printf. Let's take a look.



```
char str[]="I am GR8DON";
str[4]='\0';
printf("%s", str);
```

Output

```
I am
```

```
int i;
for (i=0; i < 11; i++) {
      putchar(str[i]);
}
```

Output

```
I amGR8DON
```

The character '\0' may be printed differently on screen depending on terminal settings.

Reading a String (scanf)

- Placeholder: %s
- Argument: Name of character array.
- No & sign before character array name. (?)
- Input taken in a manner similar to numeric input.
- With %s, scanf skips whitespaces.
  - There are three basic whitespace characters in C : space, newline ('\n') and tab ('\t').
  - Any combination of the three basic whitespace characters is a whitespace.

Reading a String (scanf)

- Starts with the first non-whitespace character.
- Copies the characters into successive memory cells of the character array variable.
- When a whitespace character is reached, scanning stops.
- scanf places the null character at the end of the string in the array variable.

```
#include <stdio.h>

int main() {
 char str1[20], str2[20];

 scanf("%s",str1);
 scanf("%s",str2);

 printf("%s + %s\n",
              str1, str2);

 return 0;
}
```

INPUT
IIT Bhilai

OUTPUT
IIT + Bhilai

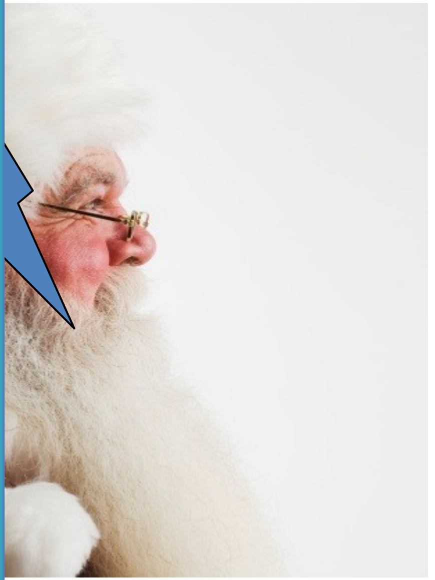
INPUT
I am DON

OUTPUT
I + am

Why is there no & when we read character array?

Remember parameter passing?

- *A simple variable can not be modified from inside a function call (Recall swap() function)*
- *However, Arrays can be modified from inside a function call as we pass the array name(pointers)*



NULL character ‘\0’

- ASCII value 0
- Marks the end of the string
- C needs this to be present in every string in order to differentiate between a character array and a string
- Size of char array holding the string
   1 + length of string
   - Buffer overflow otherwise!

NULL Character ‘\0’

- What happens if no '\0' is kept at the end of string?
  - ‘\0’ is used to detect end of string, for example in printf(“%s”, str)
  - Without ‘\0’, such functions will keep reading array elements beyond the array bound (out of bound access)
  - We can get an incorrect result or a Runtime Error

Reading a Line as an Input

- scanf, when used with the %s placeholder, reads a block of non-whitespace characters as a string.
- What if we want to read a line as a string?
- We will define our own function to read a line.
- **EXERCISE:** Take as input a line (that ends with the newline character) into a character array as a string.

```
#include <stdio.h>
// read a line into str, return length
int read_line(char str[]) {
   int c, i=0;
   c = getchar();
   while (c != '\n' && c != EOF) {
      str[i] = c;
      c = getchar();
      i++;
   }
   str[i] = '\0'; // we want a string!
   return i; // i is the length of the string
}
```

Buffer overflow possible

```
#include <stdio.h>
// read a line into str, return length.
//   maximum allowed length is limit
int read_line(char str[]          ) {   , int limit
   int c, i=0;
   c = getchar();
   while (c != '\n' && c != EOF) {
      str[i] = c;
      c = getchar();
      i++;
      if (i == limit-1) break;
   }
   str[i] = '\0'; // we want a string!
   return i; // i is the length of the string
}
```

Safer version!

Special string I/O

- C has special I/O functions for strings
  - Use gets() for string input
  - Use puts() for string output

```
int main(){
      char line[80];
      gets(line);
      puts(line);
}
```

- But gets is a terrible function
- Never use it!

Reading with fgets

- The *gets()* function doesn't know the size of the buffer it is writing to
  - can easily overflow
- Fix: use *fgets()* instead
- Syntax
    - char* fgets(char *string, int length, FILE * stream);
- To read from standard input, just set file stream to *stdin*
  - E.g. fgets(str, 30, stdin) will read a string of name *str* and size 30 from the standard input

Strings: recap

1. Character array terminated by ‘\0’ character.
2. Declare: char str[] = “Delhi”;
3. Print: printf(“this is the string: %s \n”, str);
4. Read: scanf(“%s”, str); \\ no & before name
    1. Reads from first non-whitespace to next whitespace
5. Null character: ‘\0’
    1. Marks the end of string. Not same as EOF
    2. ASCII value 0.