CSE 220 – C Programming

Strings

Outline

- String literals
- String variables
- String operations
- Arrays of strings
- Command line arguments

String literals - Definition

- String literal: a sequence of characters enclosed within double quotes:

```
“I won last night”

“First place: Amy\nSecond place: Mary”

printf(“The average temperature is 89”)

scanf(“%d/%d”, &a, &b)
```

How do you have a double quote in a string literal?

- "\""
- "My favorite 'word' is \"Ferret\"!"
- You escape the double quote with a backslash
  - Note that single quotes don't need to be escaped in string literals.

Continuing a String Literal

```
printf(“Electricity travels at the speed of light -
more than 186,000 miles per second!”);  //Wrong, can’t
just split
```

- **Splicing**: if a string is too long to fit on one line, splice using backslash character: \

```
printf(“Electricity travels at the speed of light - \

more than 186,000 miles per second!”);     //Correct
```

  - No other characters may follow the \
  - The string must continue at the beginning of the next line, messing up the indentation structure.

Continuing a String Literal

- **Joining**: 2 adjacent strings (separated only by white space) are joined into one:

```
printf(“Electricity travels at the speed of light”
“- more than 186,000 miles per second!”);
printf("I love " "the weekend");
```

String Literals - Storage

- String literals are stored as character arrays
- C adds the **null character** \0 to denote the end of the string

Y	o	u		w	i	n	\0

- C uses n+1 bytes for a string literal of length n
- A string literal is a pointer of type char *

```
        char *p = “You win”;
```

- String literals can be subscripted

```
        char ch = “abc”[0];
```

Which of the following are strings?

```
1. char *a = "hi";
2. char b[] = "hi";
3. char c[] = {'h', 'i', '\0'};
4. char d[3] = {'\0'};
```

String Variables

- String: a character array terminated by '\0'

```
#define STR_LEN 100

…

char mystring[STR_LEN+1]
```

- Always make sure the array is one character longer than the string it holds
- The length of the string depends on the position of the null character

Initialization

- `char day[10] = “Wednesday”;`

W	e	d	n	e	s	d	a	y	\0

- `char day[10] = {‘W’, ’e’, ’d’, ’n’, ’e’, ’s’, ’d’, ’a’, ’y’,  ’\0’};`

W	e	d	n	e	s	d	a	y	\0

- `char day[11] = “Wednesday”;`

W	e	d	n	e	s	d	a	y	\0	\0

- `char day[6] = “Wednesday”;`

W	e	d	n	e	s

Unusable String

- `char day[ ] = “Wednesday”;`

W	e	d	n	e	s	d	a	y	\0

Compiler sets aside enough space to store string and \0

What does the array's contents look like for the following code:

```
char array[3] = "abc";
```

1.

a	b	c	\0

2.

a	b	c

3.

a	b	\0

4. Error

Which of the length of the char array named "array"?

```
char array[] = "abc";
```

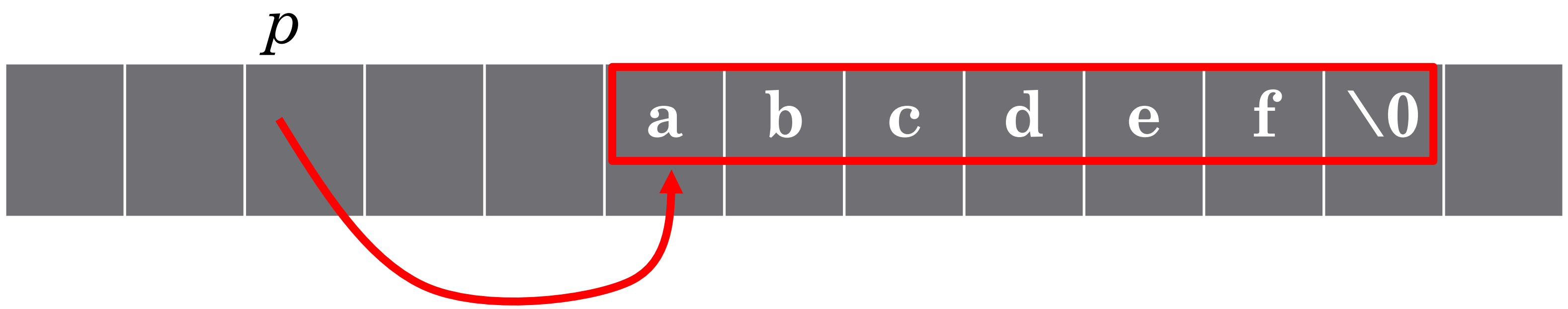
1. 5
2. 4
3. 3
4. 42

Changing a String Literal

- Attempting to modify a string literal causes undefined behavior

```
char *p = “abcdef”;
*p = ‘A’;
```

WRONG
The program may crash or behave erratically.



Writing Strings

- Use **printf** with %s specifier:

```
char str[ ] = “What time is it?”;
printf(“%s\n”, str);                //prints: What time is it?
printf(“%.4s\n”, str);              //prints: What
```

- The conversion specification %m.ps:
  - m: min number of char, adds spaces if needed
  - p: number of characters to be displayed
- Use **puts(str)** (optional content)
  - One argument, the string
  - Advances to new line

Reading Strings

- Must make sure input fits in array:

```
char str[20];
```

- Use **scanf** with %s specifier:

```
scanf(“%s”, str);     //Don’t need &
```

- scanf skips whitespaces, reads characters, stops when reaches a white space:
  - **Strings read using scanf will never contain whitespace**

- Does not check if array is long enough to fit the word read

What is the string stored in the character array?

Input is:
Cao is my name.

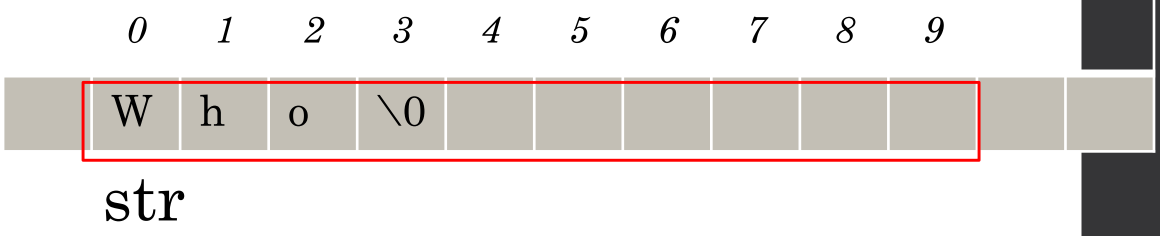
```c
char array[100];
scanf("%s", array);
```

```
1. "Cao is my name."
2. "Cao is my name.\0\0\0\0...."
3. "Cao"
4. Illegal Operation
```

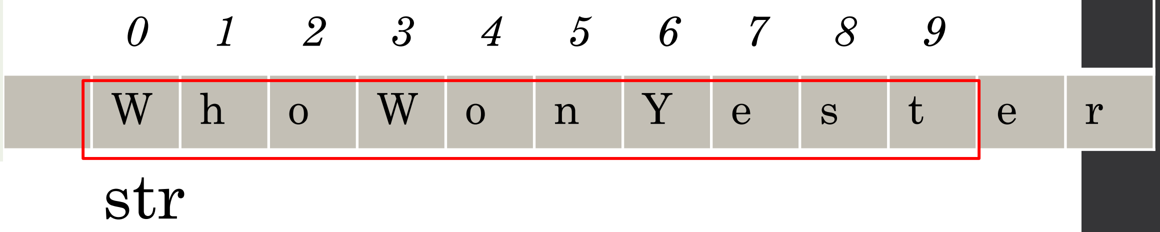
Reading Strings

```
char str[10];
scanf("%s", str);
```

User input:
Who won yesterday?



User input:
WhoWonYesterday?



Invalid string (not terminated by a \0

What is the string stored in the character array?

Input is:
Cao is my name.

```
char array[3];
scanf("%2s", array);
```

1. "Cao"
2. "Ca"
3. "C"
4. Illegal Operation

Reading Strings (Best Way)

- Must make sure input fits in array:

```
char str[21];
```

- Use **scanf** with %ns specifier:

```
scanf(“%20s”, str);
```

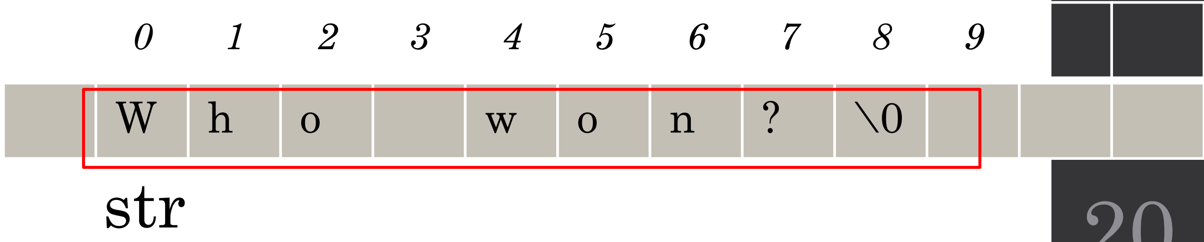
This scanf specifies the max number of characters to read into a string. It will always fit in an array of characters that is one larger.

Reading Strings (Optional Content)

- Use **gets(str)**
  - Does not skip whitespaces
  - Reads until the new line character, and discards it
  - Does not make sure there is enough memory allocated for the string

```
char str[10];
gets("%s", str);
```

User input:
Who won?



C String library (Optional Content)

- Cannot copy or compare in straightforward way:

```
char str1[10] = “abc”, str2[10];
str2 = “abc”;                      //Wrong!
if (str1 == str2 ) { … }           //Wrong!
//Checks if str1 and str2 have same address
```

- C provides a set of functions for dealing with strings:

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

- Declared with pointers (instead of arrays):
  - char *s : can modify what the pointer points to
  - const char *s : cannot modify what the pointer points to, can modify it to point to something else

C String library (Optional Content)

- char *strcpy(char *s1, const char *s2);
  - Copies content of s2 into s1, returns pointer s1
- size_t strlen(const char *s);
  - size_t: defined using typedef as unsigned int
  - Returns length of string up to but not including \0
- char *strcat(char *s1, const char *s2);
  - Appends content of s2 to the end of s1, returns s1;
- short *strcmp(const char *s1, const char *s2);
  - Compares content of s1 and s2: returns 0 if the same, negative int if s1 < s2 and positive int if s1 > s2 (lexicographic order)

Example

```
#include <stdio.h>
#include <string.h>
#define max 101

int main(void) {
    char originalStr[max];
    char upperStr[max];
    printf(“Enter a sentence:\n”);
    gets(originalStr);    //Read the whole line from user
    strcpy(upperStr, originalStr);       //Make a copy
    char *ptr = upperStr;
    while (*ptr != ‘\0’) {
        //If lower case
        if (*ptr >= ‘a’ && *ptr <= ‘z’)
            //Make upper case
            *ptr = *ptr + ‘A’ – ‘a’;
        //Move to the next character in the string
        ptr++;
    }
}
```

Example

```
char *ptr = upperStr;
while (*ptr != ‘\0’) {
   //Process
   …      …       …
   //Move to the next character in the string
   ptr++;
}
```

Accessing String Content

- Use subscript or pointer
- To declare a string parameter: pointer or array

```
int count_spaces(char *s);
int count_spaces(char s[ ]);
```

- Can call count_spaces with pointer and array:

```
char str[ ] = “a bc d e”, *p = str;
count_spaces(str);
count_spaces(p);
```

Char Arrays vs Char Pointers

```
char day1[ ] = “Monday”;   //declare as array
char *day2 = “Monday”;     //declare as pointer
```

- The two versions are **not** equivalent:
  - Characters in day1 can be modified. Characters in day2 should not be modified.
  - day1 and day2 are both pointers. But day1 cannot be assigned to a different value
- Declaring a pointer: `char *p;` causes the compiler to set aside memory for a pointer, not for a string.

Which lines are legal after this code?

```
char array[3] = "ab";
char *ptr = "cd";
```

```
1. array[0] = 't'
2. ptr[0] = 't'
3. array = ptr;
4. ptr = array;
```

String Idioms

- Find length of a string:

```
int strlen(char *s) {
     int n;
     for (n=0; *s != ‘\0’;
             s++) {
          n++;
     }
     return n;
}
```

```
int strlen(char *s) {
     int n = 0;
     for (; *s; s++)
          n++;
      return n;
}
```

```
while (*s) {
     s++ ;
     n++;
}
```

```
while (*s++)
     n++;
```

```
char *p = s;
while (*s++);
return s – p;
```

Array of Strings

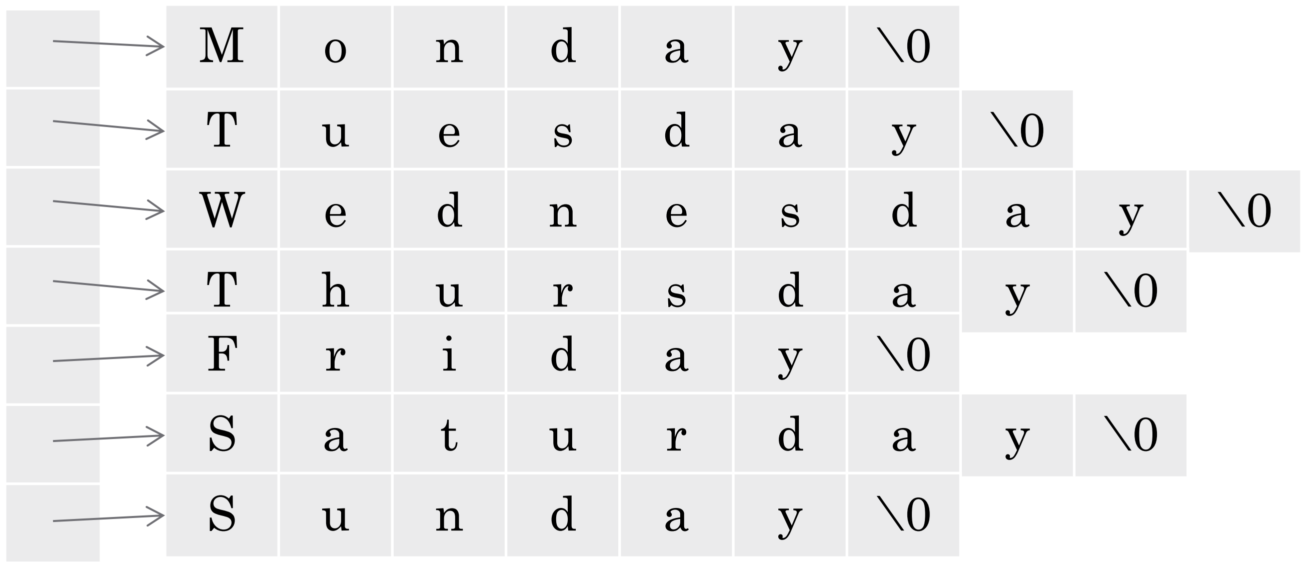
char daysOfWeek[ ][10] = {“Monday”, “Tuesday”, “Wednesday”, “Thursday”, “Friday”, “Saturday”, “Sunday”};

M	o	n	d	a	y	\0	\0	\0	\0
T	u	e	s	d	a	y	\0	\0	\0
W	e	d	n	e	s	d	a	y	\0
T	h	u	r	s	d	a	y	\0	\0
F	r	i	d	a	y	\0	\0	\0	\0
S	a	t	u	r	d	a	y	\0	\0
S	u	n	d	a	y	\0	\0	\0	\0

Some wasted space in rows containing shorter strings

Array of Strings

char *daysOfWeek[ ] = {“Monday”, “Tuesday”, “Wednesday”, “Thursday”, “Friday”, “Saturday”, “Sunday”};



Does it matter which of the two options below you use?

```
char daysOfWeek[ ][10] = {“Monday”,
“Tuesday”, “Wednesday”, “Thursday”,
“Friday”, “Saturday”, “Sunday”};
char *daysOfWeek[ ] = {“Monday”,
“Tuesday”, “Wednesday”, “Thursday”,
“Friday”, “Saturday”, “Sunday”};
```

1. Not really
2. Only if you really want to save space
3. Functions that need to be given a size may care

Command Line Arguments (Optional Content for now)

- Used to supply information to the program
- Define main as a function with 2 parameters:

```
int main(int argc, char *argv[ ])
```

- Run your program:

```
./add 13 100
```

- argc is 3
- argv[0] is "./add"
- argv[1] is "13"
- argv[2] is "100"

Pitfalls

- Don't use a character when a string is required:

```
printf('a');                //Wrong
printf("a");                //Correct
printf("%c", 'a');
printf("%s", "a");
```

- Don't modify a string literal

```
char *p = "abc";
*p = 'A';    //WRONG. Can't change the value p points to
p = "def";   //VALID. Can make p point to something else
```

- Although array name is a pointer: cannot assign it to a new value:

```
char a[10];
a = b;       //WRONG
a++;         //WRONG
```

Summary

- String literals
- String variables
- Reading and writing strings
- String library
- Arrays of strings
- Command line arguments

Practice Problems

Does the following code copy a string?

```
char array_1[] = "abc";
char array_2[10];
char * ptr = array_1;
char * ptr_2 = array_2;
for ( ; *ptr != '\0'; ++ptr, ++ptr_2) {
    *ptr_2 = *ptr;
}
```

1. Yes
2. No, the null character isn't copied.
3. No, it causes an error.
4. No, the arrays are different lengths.

What is the length of the array?

```
char array[] = "a\n\"b";
```

1. 7
2. 6
3. 5
4. 4

What is the length of the array?

```
char array[] = "abc" "def";
```

```
1. 7
2. 6
3. 5
4. 4
```