Strings and File I/O

Character Type: char

In C, characters are indicated by *single quotes*:

```
char myChar;
myChar = 'a';
printf("myChar is %c\n", myChar);
```

The type is *char* for one (1) character

%c is the formal specifier for a char

String

A string is an array of characters, it is indicated by *double quotes*:

"this is a string"

But you can't use an assignment operator to assign a string to a character array

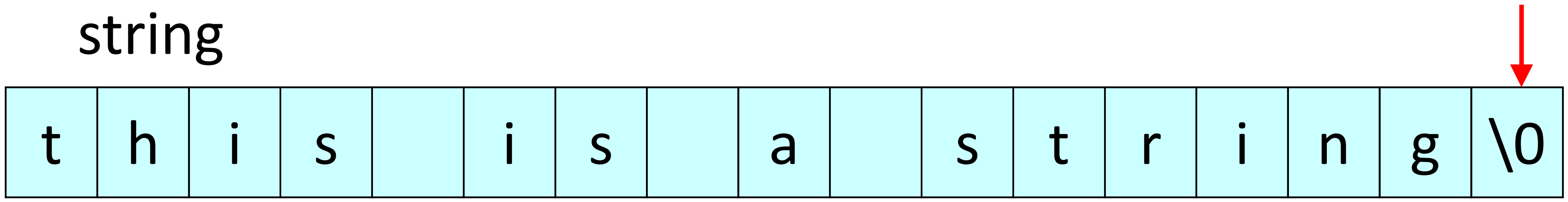
char myChar[80]; /* array of chars */

myChar[4] = 'a'; /* OK */

~~myChar = “this is a string”;~~ /* NOT OK */

String

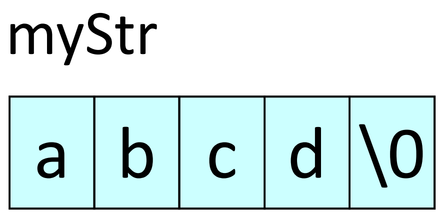
A null ('\0') character is placed to mark the end of each string



String functions use '\0' to locate end of string (so you don't have to pass string length as argument to the functions)

Initializing a String

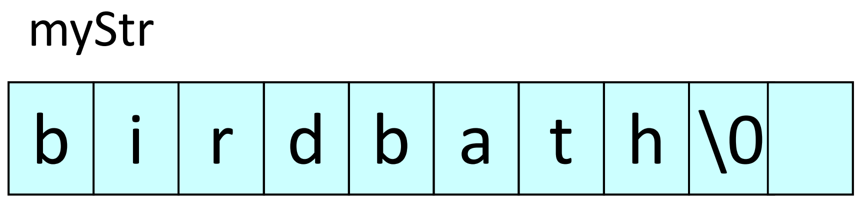
```
char myStr[5] = {‘a’, ’b’, ’c’, ’d’, 0};
printf(“myStr is %s\n”, myStr);
```



- Array size has exactly 5 elements
- A terminating ‘\0’ character at the end (in ASCII, char ‘\0’ is equivalent to int 0)
- Will print: “myStr is abcd”

Another way

```
char myStr[10]="birdbath";
printf("mySt is %s\n", myStr);
```



- Ok to use assignment <u>only</u> to initialize string
- myStr is an array of 10 characters (but only the first 8 elements used to store the word)
- a '\0' is added to the end automatically (so 9 elements in the array are occupied)

Better way

```
char myStr[]="birdbath";
printf("mySt is %s\n", myStr);
```

myStr

b	i	r	d	b	a	t	h	\0

- '\0' is added automatically to the end of myStr
- Space is allocated <u>automatically</u> to hold exactly what we need (9 locations, including '\0')

printf for Strings

char myStr[] = “abc123”;

Output:

- Use **%s** to print the entire string:

```
printf(“%s\n”,myStr);      /* outputs abc123 */
                           /* ‘\0’ is not printed */
```

- Use **%c** to print a character:

```
printf(“%c\n”,myStr[1]);   /* outputs:  b */
```

String input

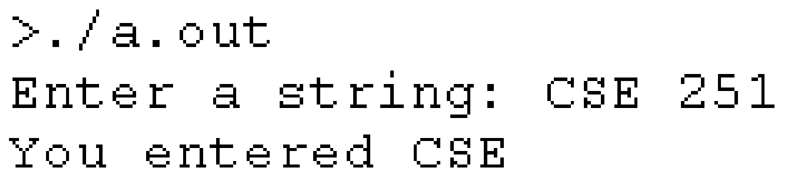
There are several functions available

Part of the stdio.h library.

scanf

scanf reads up to the first white space, ignores the stuff typed in after that. **Be careful when using it.**

```
char myStr[10];
printf(“Enter a string: “);
scanf(“%s”, myStr);
printf(“You entered %s\n\n”, myStr)
```

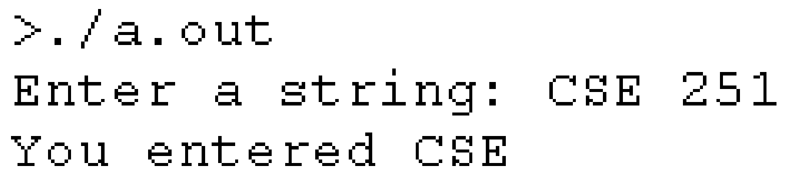


```
>./a.out
Enter a string: CSE 251
You entered CSE
```

scanf - safer

Use %[width]s to copy only up to a maximum number of character. But, does not append the ‘\0’

```
char myStr[10];
printf(“Enter a string: “);
scanf(“%9s”, myStr);
myStr[9] = ‘\0’;                    /* Do this yourself just in case */
printf(“You entered %s\n\n”, myStr)
```



```
>./a.out
Enter a string: CSE 251
You entered CSE
```

getchar

getchar will fetch one (1) character from the input stream

```
char myChar;
printf("Enter a character: ");
myChar = getchar();
printf("You entered %c\n\n", myChar);
```

```
>./a.out
Enter a character: this
You entered t
```

fgets

char * fgets(charArray, lengthLimit, filePtr)

- fetches a *whole line*, up to the size limit or when it sees a new line
- It will add a '\0' at the end of string
- Example:

```
char myStr[80];
fgets(myStr, 80, stdin);  // fetch from console
```

- Returns a NULL if something went wrong, otherwise a pointer to the array

The functions that start with "f" work with any source of input. stdin is the C "Standard Input".

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

int main ()
{
   int inputLength=20
   int cnt=0;
   char str1[20], str2[20];

   printf("\nEnter a string: ");
   fgets(str1, inputLength, stdin);
   printf("You entered %s\n",str1);

   printf("Enter another string: ");
   scanf("%s",str2);
   printf("You entered %s\n",str2);
}
```

Make sure you're clear the difference between fgets and scanf

Note the extra new line when using fgets

str1 (read using fgets)

t	h	i	s		i	s		a		s	t	r	i	n	g	\n	\0		

str2 (read using scanf)

t	h	i	s	\0															

```
>./a.out
Enter a string: this is a string
You entered this is a string

Enter another string: this is another string
You entered this
```

String manipulation functions

#include <string.h>

strcpy – Copies a string

strcat – Concatenates two strings

strlen – Returns the length of a string

strcmp – Compares two strings

string copy (strcpy)

strcpy(destString, srcString);

```
char [] strcpy (char dest[],
                const char src[]);
```

- copy string contents from src (2nd arg) to dest (1st arg) including `'\0'`
- dest is changed, src unmodified (but can do some weird things)
- returns a pointer to the modified dest array

string copy (strcpy)

```
char [] strcpy (char dest[],
                const char src[]);
```

- There is no error checking!
  - If dest array is shorter than src array, no errors. Weird things could happen, but no compile errors and often no runtime errors
  - Tracking these “bugs” down is very hard!

Safer version of string copy

strncpy(destString, srcString, 80);

```
char [] strncpy (char dest[],
                 const char src[], int N);
```

- copies at most N characters from src to dest (or up to '\0'
- If length of src is greater than N, copies only the first N characters
- If length of src is less than N, pad the remaining elements in dest with '\0'

Does not copy the '\0' if the string length is >= N

concatenation (strcat)

```
char [] strcat(char dest[],
               const char src[])
```

- contents of src are added to the end of dest.
- dest is changed, src is not
- ‘\0’ added to the end of dest
- return a pointer to dest
- no bounds check (again, this is C)

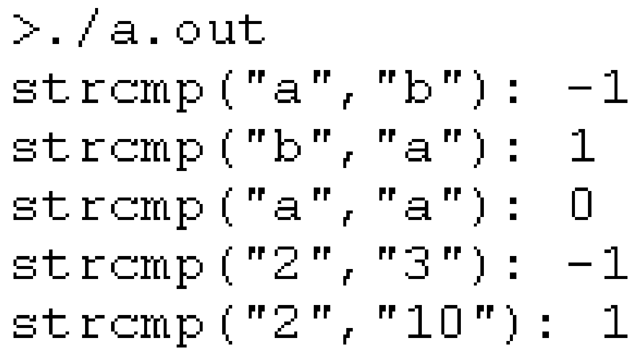
comparison (strcmp)

```
int strcmp (const char s1[],
            const char s2[]);
```

- Compares 2 strings
  - if s1 precedes s2, return value is less than 0
  - if s2 precedes s1, return value is greater than 0
  - if s1 equal to s2, return value is 0
- Comparison is based on lexicographic order (ASCII order) – e.g., “a” < “b”

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

int main()
{
   printf("strcmp(\"a\",\"b\"): %d\n",  strcmp("a","b"));
   printf("strcmp(\"b\",\"a\"): %d\n",  strcmp("b","a"));
   printf(“strcmp(\"a\",\"a\"): %d\n",  strcmp("a","a"));
   printf("strcmp(\"2\",\"3\"): %d\n",  strcmp("2","3"));
   printf("strcmp(\"2\",\"10\"): %d\n",  strcmp("2","10"));
}
```



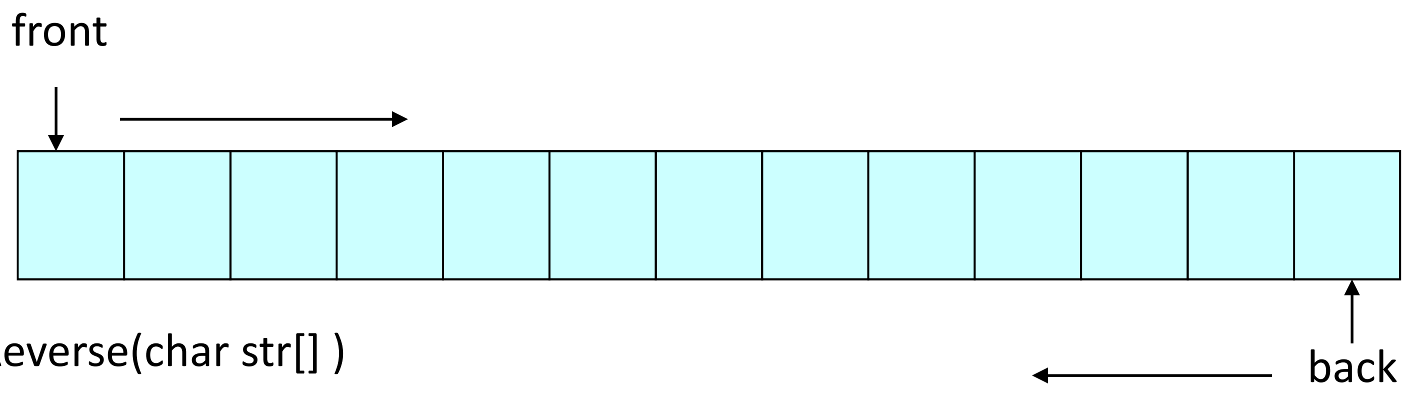
```
>./a.out
strcmp("a","b"): -1
strcmp("b","a"): 1
strcmp("a","a"): 0
strcmp("2","3"): -1
strcmp("2","10"): 1
```

Lexicographic ordering is not the same as ordering numbers

Reversing a string

Input: Math is fun
Output: nuf si htaM

How would you do this?



```
void Reverse(char str[] )
{
   int front = 0;
   int back = strlen(str) - 1;
   char t;          /* A temporary place to put a character */

   while (front < back)
   {
      t = str[front];
      str[front] = str[back];
      str[back] = t;
      front++;
      back--;
   }
}
```

#include <ctype.h>

Built-in functions for characters

- These all return boolean (1/0).
  - isalnum(c): is c alphanumeric?
  - isalpha(c): is c alphabetic?
  - isdigit(c): is c a digit?
  - iscntrl(c): is c a control character?
  - islower(c): is c lower case?
  - isupper(c): is c upper case?
  - ispunct(c): is c a punctuation character, that is a printable character that is neither a space nor an alphanumeric character

converters

- char tolower(c)
  - return the character as lower case
- char toupper(c)
  - return the character as upper case

File IO

How to write programs to read from or write to files

So far, we've only looked at how to read/write to the console

Open files

- Files are opened with fopen
  - the fopen function is part of stdio.h
  - It takes two args: a string (the name of a file) and a mode string (how the file is opened).
  - It returns a file pointer
    - `fptr = fopen(“myfile.txt”,”r”);`

Declare and assign file pointer

- fopen returns a NULL pointer if it cannot perform the operation

```
FILE *infile;
inFile = fopen(“file.txt”,”r”);
if (inFile == NULL)
  // bad file opening
```

file modes

- “r”, read from the beginning of an existing file
- “w”, make an empty file (wipe out old contents), start writing.
- “a”, find an existing file, start writing at the end of that file

Close files

- Files are closed with fclose
  - the fclose function is part of stdio.h
  - It takes one argument:
    pointer to the file to be closed

```
fclose(fptr);
```

The “f” functions

- most of the I/O functions we know have an equivalent “f” version to work with a file.
- The first argument is the file pointer value of an opened file
  - fprintf(filePtr, “format string”, values)
  - fscanf(filePtr, “format string, addresses)
  - feof(filePtr) end of file test