String Processing in C

C Programming and Software Tools

N.C. State Department of Computer Science

Computer Science
NC STATE UNIVERSITY

Standard Library: `<ctype.h>`

- Many functions for checking whether a character is a digit, is upper case, …
  - `isalnum(c)`, `isalpha(c)`, `isspace(c)`,…
- Also, functions for converting to upper case and converting to lower case
  - `toupper(c)`, `tolower(c), …`
- Argument is an `int` and return is an `int`
  - Works fine with unsigned chars or 7-bit character types
  - Need to cast to **`unsigned char`** for safety

Computer Science
NC STATE UNIVERSITY

`<ctype.h>` (cont'd)

Checking:

`isalnum (c)`	c is a letter or a digit
`isalpha(c)`	c is a letter
`isdigit (c)`	c is a decimal digit
`islower (c)`	c is a lower-case letter
`isspace (c)`	c is white space (\f\n\r\t\v)
`isupper (c)`	c is an upper-case letter

Converting:

`tolower (c)`	convert c to lower case
`toupper (c)`	convert c to upper case

Only a partial list (see p. 612-613 or library for full list)

You Try It

- Code to convert lower-case to upper case, no change to rest?
  - **`char array[] = "abcde";`**
- Code to replace all "white space" with a underscore?
  - **`char array[] = "a b\fc\nd\re\tf\vg";`**
- Code to skip white space, convert ASCII digits to a number until non-digit encountered, and output the number?
  - **`char array[] = "1 2\f3\n4\r5\t6\v7";`**

Strings

- Simply 1-D arrays of type char, terminated by null character (`'\0'`)
- A variety of standard library functions provided for processing

`scanf()` and `printf()` for Strings

- `sscanf(s, "…", …)` scans a string (instead of stdin) for expected input
- `sprintf(s, "…", …)` outputs to a string (instead of stdout) the specified output

- You try it:
  - read integer and floating point numbers from a string
  - create a string with format "The number is xxxxx\n", where xxxxx is a number

Standard Library: `<string.h>`

- (`<strings.h>` on some machines)
- Lots of string processing functions for
  - copying one string to another
  - comparing two strings
  - determining the length of a string
  - concatenating two strings
  - finding a substring in another string
  - …
- Function headers at end of slides
- More details in King text book

A Useful Memory Operation: `memcpy()`

- Must `#include <string.h>`
- Syntax:
  note order!
  `void * memcpy (void *dest, void *src, size_t n)`
- Copy `n` bytes from memory pointed to by `src` to memory pointed to by `dest`
  - memory areas must not overlap!
- Returns pointer to `dest`

memcpy() (cont'd)

- Since C does not have an operator to assign one array to another, this is a handy function

```
#define SZ 1000
int *ip, *jp;

int A[1000], B[1000];

… assign some values to A …

(void) memcpy (B, A, 1000*sizeof(int));
```

Variant: memmove()

- **memmove()** works just like **memcpy()**, except **src** and **dest** areas may overlap

Another Useful Operation: `memcmp()`

- Syntax:

```
int memcmp (void *s1, void *s2,
            size_t n)
```

- Returns 0 if `n` bytes starting at `s1` are equal to `n` bytes starting at `s2`
- Else, return val < 0 if first non-equal byte of `s1` < byte of `s2,` > 0 if …
- Useful for comparing arrays, but byte-by-byte comparison only
  - e.g., don't use for comparing arrays of ints, floats, structs, etc.

`memcmp()`... (cont'd)

```
char X[1000], Y[1000];

int A[1000], B[1000];

… assign some values to A, B, X, Y …

if (memcmp(X, Y, 1000) < 0)
    ...X is less than Y...
```

Do not try this with A and B; why not?

You Try It

- Print the length of a string
- Concatenate two strings and print the result
- Compare two strings and copy the lesser to the greater
- Find how many times the character '?' occurs in a string
- Find the tokens in a string, print one by one

Good Practice

- You should be able to write the code for any of the standard library functions
  - e.g., computing the length of a string…

```
char s[1000] = “a string”;
char *p = s;

while (*p++)
    ;

return (p – s);
```

`<stdlib.h>` String Functions

- `double atof( char s[] )` converts a string to a `double`, ignoring leading white space
- `int atoi( char s[] )` converts a string to an `int`, ignoring leading white space
  - These don't return information about errors
- (instead of…)
- Could also use
  - `strtol`
  - `strtod/f`

```
int num = 0;
while (isspace(c = getchar()))
    ;
while (isdigit(c)) {
    num = num * 10 + c – '0';
    c = getchar();
}
```

Arrays of Strings

- Creating a two dimensional array of chars is inefficient
  - Wasted space when strings of different lengths
- Instead we want a ragged array
  - Create an array where the elements are pointers to strings

```
char *planets[] = {“Mercury”,
“Venus”, “Earth”, “Mars”, “Jupiter”,
“Saturn”, “Uranus”, “Neptune”);
```

Arrays of Strings (con't)

- Accessing a string in the array
  - **`planets[i]`**
- Accessing a character in a string
  - **`planets[i][j]`**

```
Example:
for (int i = 0; i < 8; i++)
   if (planets[i][0] == ‘M’)
      printf(“%s\n”, planets[i]);
```

Computer Science NC STATE UNIVERSITY

Command Line Arguments

- To use command line arguments, define main as:

  `int main(int argc, char *argv[]) {}`
  - **`argc`**: argument count
    - Includes the program itself
  - **`argv`**: argument vector
    - Array of pointers to command line arguments stored as strings
    - **`argv[0]`**: name of program
    - **`argv[1]-argv[argc-1]`**: other arguments
    - **`argv[argc]`**: null pointer

Computer Science NC STATE UNIVERSITY

Processing Command Line Args

- Using arrays

```
for (int i = 1; i < argc; i++)
   printf(“%s\n”, argv[i]);
```

- Using pointers

```
for (char **p = &argv[1]; *p != NULL; p++)
   printf(“%s\n”, *p);
```

<string.h>: Copying

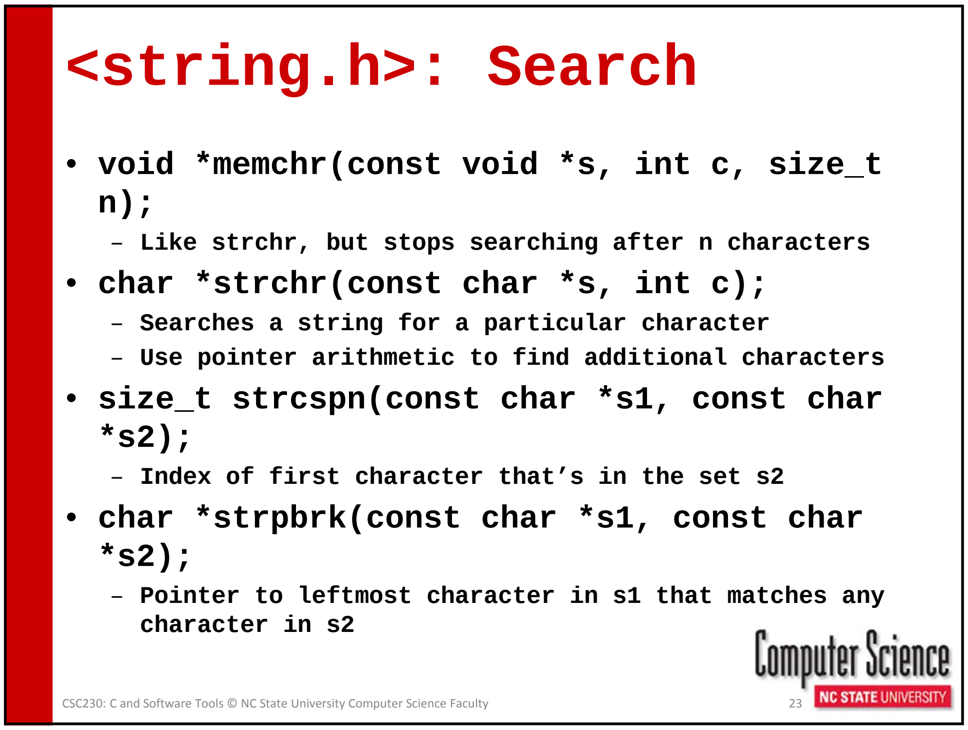
- `void *memcpy(void * restrict s1, const void * restrict s2, size_t n);`
- `void *memove(void *s1, const void *s2, size_t n);`
- `char * strcpy(char * restrict s1, const char * restrict s2);`
- `char *strncpy(char * restrict s1, const char * restrict s2, size_t n)`

<string.h>: Concatenation

- char *strcat(char * restrict s1, const char * restrict s2);
- char *strncat(char * restrict s1, const char * restrict s2, size_t n);

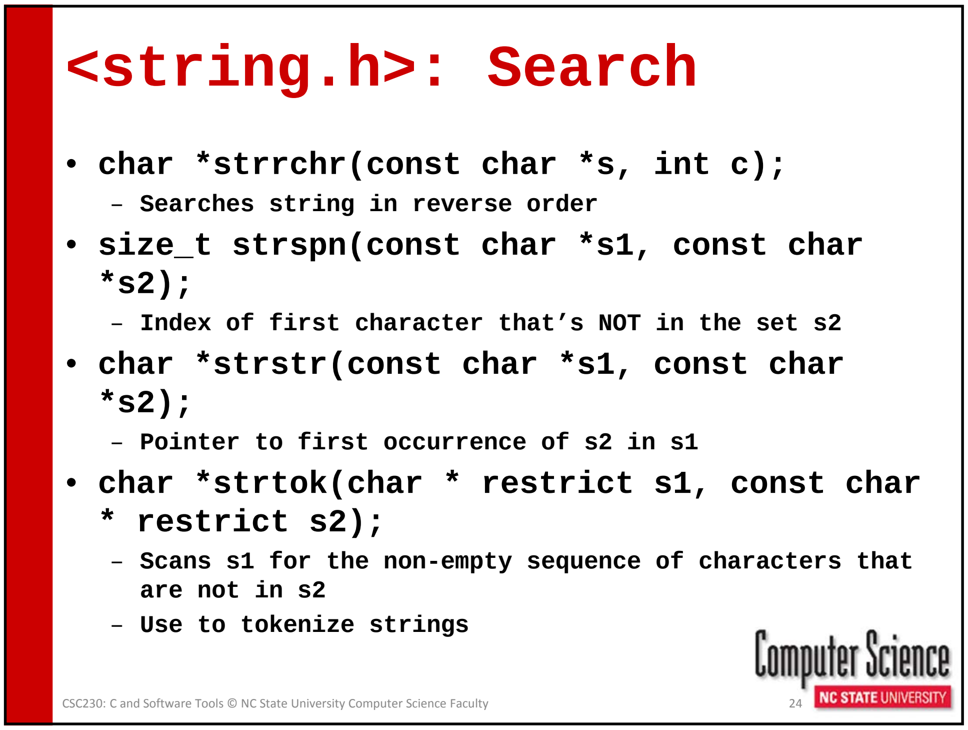
<string.h>: Comparison

- int memcmp(const void *s1, const void *s2, size_t n);
  - n comparisons
- int strcmp(const char *s1, const char *s2)
  - Stops when reaches null in either string
- int strcoll(const char *s1, const char *s2);
  - Locale dependent
- int strncmp(const char *s1, const char *s2, size_t n);
  - Stops when reaches null in either string or n comparisons, which ever is first



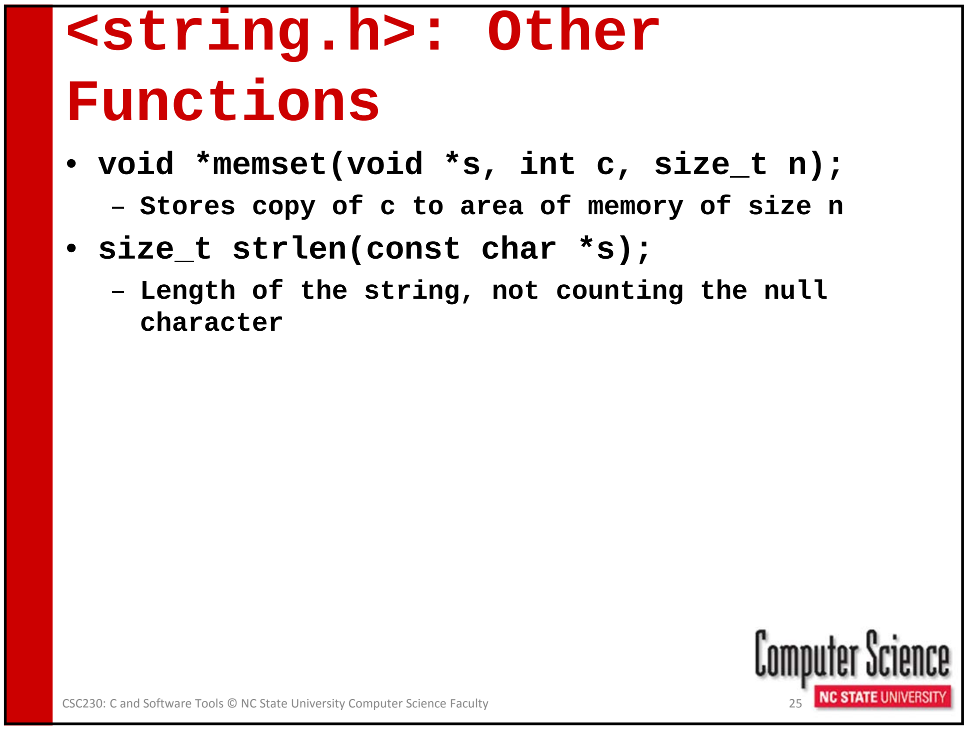
<string.h>: Search

- void *memchr(const void *s, int c, size_t n);
  - Like strchr, but stops searching after n characters
- char *strchr(const char *s, int c);
  - Searches a string for a particular character
  - Use pointer arithmetic to find additional characters
- size_t strcspn(const char *s1, const char *s2);
  - Index of first character that's in the set s2
- char *strpbrk(const char *s1, const char *s2);
  - Pointer to leftmost character in s1 that matches any character in s2



<string.h>: Search

- char *strrchr(const char *s, int c);
  - Searches string in reverse order
- size_t strspn(const char *s1, const char *s2);
  - Index of first character that's NOT in the set s2
- char *strstr(const char *s1, const char *s2);
  - Pointer to first occurrence of s2 in s1
- char *strtok(char * restrict s1, const char * restrict s2);
  - Scans s1 for the non-empty sequence of characters that are not in s2
  - Use to tokenize strings



<string.h>: Other Functions

- `void *memset(void *s, int c, size_t n);`
  - Stores copy of c to area of memory of size n
- `size_t strlen(const char *s);`
  - Length of the string, not counting the null character