Today's Goals

- Strings and Pointers
- **`string.h`** functions
  - Pointer implementations
- Array of Strings
- Command-Line Arguments
  - **`argc`**
  - **`argv`**

Section 1

Strings in C

- String
  - An array of characters
  - Terminated with a special, null character `'\0'`
- E.g., **`"abc"`** is internally

'a'	'b'	'c'	'\0'	

- Declaration: `char s[5];`
- Initialization:
  - `char t[] = "abc";`
  - `s[0]='a'; s[1]='b'; s[2]='\0';`

Section 2

String Input/Output

- The **`gets`** function
- **`gets`** deletes the `'\n'`
- **`gets`** is dangerous because of fixed buffer size.

- **`printf`** with the '**`%s`**' specification

  Prints character elements until `'\0'` is reached

`printf` and Strings

```
int main() {
   char s[] = "01234";
   char *p;
   p = s;

   printf("%c\n", s[0]);
   printf("%c\n", *s);
   printf("%c\n", *(p+1));

   printf("%s\n", s);
   printf("%s\n", p+1);
}
```

- **`%d`**, **`%c`**, **`%f`**: Displays the given *value*
- **`%s`**: Displays characters from the specified *address* until `'\0'`

Displaying Substrings

```
int main() {
   char s[] = "01234";
   char *p;
   p = s;

   printf("%s\n", s);
   printf("%s\n", p);

   printf("%s\n", s + 0);
   printf("%s\n", &(s[0]));

   printf("%s\n", s + 2);
   printf("%s\n", &(s[2]));
}
```

Displaying Characters of a String

```
int main() {
   char s[] = "01234";
   char *p;
   p = s;

   printf("%c\n", s[0]);
   printf("%c\n", *p);
   printf("%c\n", *(p + 0));

   printf("%c\n", s[2]);
   printf("%c\n", *(p + 2));
}
```

String Input with `scanf`

- Use the **`scanf`** function with **`%s`**

```
scanf("%s", buf);
```

- Matches the input string up to the first white space or '**`\n`**' and stores it into **`buf`**
  - Given input **`"CSE 123\n"`**
  - **`scanf %s`** will have stored **`"CSE"`**
  - Input buffer after **`scanf`** call: **`"123\n"`**
- No need for **`&`** in front of **`buf`**

CS246 7 Lec10

String Output with `puts`

- The **`puts`** function

```
#define BUFLEN 200

int main() {
 char buf[BUFLEN];

  gets(buf);
  puts(buf);
  return 0;
}
```

puts adds '`\n`' to output, equivalent to `printf("%s\n", buf);`

CS246 8 Lec10

`sscanf` Function

- Another variation on **`scanf`**
- Instead of the keyboard, this function takes input from the specified string argument.
- **`int sscanf(char *s, "...", variableList);`**
- A common practice is to use **`gets/fgets`** to read lines from the command line, then parse it with **`sscanf`**

CS246 9 Lec10

Use `const` to Protect Pointers

- We already know that keyword **`const`** prevents the value of a variable from being changed.
  - **`const int x;`**
  - **`void f(const int *p);`**
    - Prevents **`*p`** from being changed
  - **`void f(int* const p);`**
    - Prevents the pointer **`p`** itself from being changed
  - **`void f(const int* const p);`**

CS246 10 Lec10

Section 3

String Operations

- Identify the end of a string
- Find the string length
- Copy a string
- Concatenate two strings
- Check the equality of two strings
- Search for a character/substring in a string
- Extract a substring

CS246 11 Lec10

Library String Functions

- **`#include <string.h>`**
- Find the Length of a string
  `size_t strlen(const char *str)`
- Copy a string (including the **`'\0'`**)
  `char *strcpy(char *t, const char *s)`
- Concatenate two strings
  `char *strcat(char *t, const char *s)`
- Compare two strings
  `int strcmp(const char *s1, const char *s2)`

Return: 0 if identical; +1 for, e.g., "abc" vs. "abb", -1 for "abc" vs. "abd"

CS246 12 Lec10

Example

```
int main() {
  char s[] = "ann", char s2[] = "abby";
  char s3[strlen(s)+strlen(s2)+1];

  printf("%d\n", strlen(s));
  printf("%d\n", strlen(&s[1]));

  strcpy(s3, s);
  strcat(s3, s2);
  printf("%s\n", &s3[2]);

  printf("%d\n", strcmp(s, s2));

  return 0;
}
```

Length Function **strlen**

```
int strlen(char s[]) {
  int i;
  for (i = 0; s[i] != '\0'; i++) ;
  return i;
}
```

```
int strlen(char *s)
{
  char *p=s;
  for(;*p!='\0';p++);
  return p-s;
}
```

```
int strlen(char *s) {
  int i=0;
  for(;*s!='\0';s++,i++);
  return i;
}
```

Copy Function **strcpy**

```
char* strcpy(char to[], char from[]) {
  int i;
  for (i = 0; from[i] != '\0'; i++)
    to[i] = from[i];
  to[i] = '\0'; return to;
}
```

```
char* strcpy(char *to, char *from) {
  char *tmp = to;
  while ((*to = *from)!= '\0'){
    to++; from++;
  }
  return tmp;}
```

strcpy, pointer version

```
char* strcpy(char *to, char *from) {
  char *tmp = to;
  while ((*to++ = *from++)!= '\0');

  return tmp;
}
```

```
char* strcpy(char *to, char *from) {
  char *tmp = to;
  while (*to++ = *from++);

  return tmp;
}
```

Concatenation Function **strcat**

```
char* strcat(char *to, char *from){
  char *tmp = to;
  while(*to)
    to++;

  while (*to++ = *from++);

  return tmp;
}
```

Comparison Function **strcmp**

```
int strcmp(char *s, char *t) {

  for(;*s == *t; s++,t++) {
    if (*s == '\0')
      return 0;
  }

  return *s-*t;
}
```

Cannot compare strings using `==`

- Returns `0` if `s == t`
- If not, returns the difference btw the first chars that differ

Other Library String Functions

- **n** functions

```
char *strncpy(char *t, const char *s, size_t n)
char *strncat(char *t, const char *s, size_t n)
int strncmp(const char *s1, const char *s2, size_t n)
```

  - Same as the none-n functions, only works on n chars
- Search for a character in a string

```
char* strchr(char *s, char c)
```

- Search for a (sub)string in a string

```
char* strstr(char *s, char *substr)
```

`string.h` Functions Example

```
int main() {
  char s[] = "abcdejklijkl", s2 = 'b';
  char *s3 = "jkl", *s4, *s5;

  s4 = strchr(s, s2);
  s5 = strstr(s, s3);

  printf("%s\n", s4);
  printf("%c\n", s5);       //?
  printf("%d\n", s5-s4);
  printf("%s\n", &(s[2]));
}
```

`string.h` Functions Example

```
int main() {
  char s[] = "abcdefghijkl", s2 = 'g';
  char *s3 = "jkl", *s4, *s5;

  s4 = strchr(s, s2);
  s5 = strstr(s, s3);

  printf("%s\n", s+6);
  printf("%c\n", *(s+1));
  printf("%c\n", *(++s4));
  printf("%c\n", ++(*s4));
  printf("%d\n", s5-s4);
  printf("%s\n", &(s[2]));
}
```

Using library functions

```
int main() {
  char s1[] = "The name is Bond";
  char s2[] = "Bond, James Bond";
  char s3[100];

  strncpy(s3, s1, 12);
  s3[12] = '\0';
  strncat(s3, &s2[6], 5);

  printf("%s\n", s3);
}
```

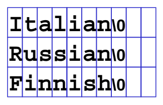
- Remember to always null-terminate a string
- **`string.h`** functions may have undefined behaviors otherwise

Section 4

Array of Strings

- Array of strings ==> two-dimensional character array

```
char langs[3][10]
  = {"Italian", "Russian", "Finnish"};
```

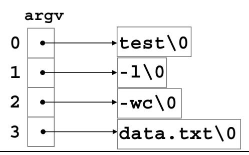


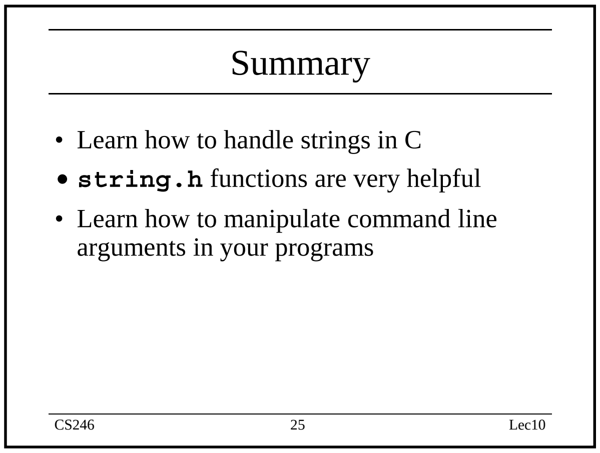
Only the first index may be omitted.

Section 5

Command-Line Arguments

- Available through two parameters to **`main`**
  - **`main(int argc, char *argv[])`**
  - **`argc`** – argument count, i.e. number of args
  - argv – an array of pointers to the arguments
- **`test -l -wc data.txt`**





Summary

- Learn how to handle strings in C
- **`string.h`** functions are very helpful
- Learn how to manipulate command line arguments in your programs