Today's Goals

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

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[1] = "abc";
  □ s[0] = 'a'; s[1] = 'b'; s[2] = 'c';

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

```c
int main() {
    char s[] = "01234";
    char *p;
    p = s;
    printf("%s\n", s[0]);
    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 Substrings

```c
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

```c
int main() {
    char s[] = "01234";
    char *p;
    p = s;
    printf("%s\n", s[0]);
    printf("%s\n", *p);
    printf("%s\n", s[0]);
    printf("%s\n", *p + 0));
    printf("%s\n", s[2]);
    printf("%s\n", *(p + 2));
}
```
String Input with `scanf`

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

String Output with `puts`

- The `puts` function
  ```c
  #define BUFLEN 200
  int main() {
    char buf[BUFLEN];
    gets(buf);
    puts(buf);
    return 0;
  }
  ```
- Given input "CSE 123"
  ```c
  scanf %s
  ```
  Will have stored "CSE"
- Input buffer after `printf` call:
  ```c
  puts %s
  ```
- No need for & in front of `buf`

`sscanf` Function

- Another variation on `scanf`
- Instead of the keyboard, this function takes input from the specified string argument.
- ```c
  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`

Use `const` to Protect Pointers

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

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

Library String Functions

- ```c
  #include <string.h>
  ```
- Find the Length of a string
  ```c
  size_t strlen(const char *str)
  ```
- Copy a string (including the `\0`)
  ```c
  char *strcpy(char *t, const char *s)
  ```
- Concatenate two strings
  ```c
  char *strcat(char *t, const char *s)
  ```
- Compare two strings
  ```c
  int strcmp(const char *s1, const char *s2)
  ```
  Return: 0 if identical, +1 for e.g., "abc" vs. "abc", -1 for "abc" vs. "abc"
Example

```c
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`

```c
int strlen(char s[]) {
    int i;
    for (i = 0; s[i] != '\0'; i++)
        return i;
    return 1;
}
```

Copy Function `strcpy`

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

Concatenation Function `strcat`

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

Comparison Function `strcmp`

```c
int strcmp(char *s, char *t) {
    for (; *s == *t; s++, t++)
        if (*s == '\0')
            return 0;
    return *s-*t;
}
```

- Returns 0 if `s == t`
- If not, returns the difference between the first chars that differ
Other Library String Functions

- **n functions**
  
  - `char *strncpy(char *, const char *, size_t n)`
  - `char *strncat(char *, const char *, size_t n)`
  - `int strncmp(const char *, const char *, size_t n)`
  
  - Same as the `none-n` functions, only works on `n` chars

- **Search for a character in a string**
  `char* strchr(char *, char c)`

- **Search for a (sub)string in a string**
  `char* strstr(char *, char *)`

---

string.h Functions Example

```c
int main() {
    char s[] = "The name is Bond";
    char *s1 = "Bond, James Bond";
    char *s2, *s3;
    s2 = strchr(s, 'g');
    s3 = strchr(s, 'a');
    printf("%s
", s1);
    printf("%s
", s2);
    printf("%s
", s3);
    printf("%s
", &s[0]);
}
```

---

Using library functions

```c
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
", s3);
}
```

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

---

Array of Strings

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

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

Only the first index may be omitted.

---

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

```c
int main() {
    char *argv[] = {"test", "-l", "-wc", "data.txt"};
    int test = -1;
    int wc = 0;
    int data = 0;
    printf("%d %d %d
", test, wc, data);
}
```

---

CS246 Lec10
Summary

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