String Overview in C (string.h)

January 30, 2025

1 Strings in Memory

A string in C is stored as an array of characters. There is no explicit "string" data type that you might see in other programming languages.

```c
#include <stdio.h>
int main() {
    char a[] = "Hi";  // Simple string declaration: b = ['H','i','\0']
    char b[3] = "Hi"; // If we want to specify the size, need to be n+1 = 3
    char c[3];
    c[0] = 'H';
    c[1] = 'i';

    printf("%s\n",a); // Use %s in printf for strings
    printf("%s\n",b);
    printf("%s\n",c);
    printf("%c\n",c[2]); // Use %c in printf for char. Output is null char
    ('\0')
    return 0;
}
```

2 Understanding Function Format

A function in C has three main components:

- **Return type**: What the function gives back (e.g., `int`, `size_t`)
- **Function name**: Identifier for the function
- **Parameters**: Inputs the function receives (in parentheses)

```c
size_t strlen(const char *str);
return_type function_name(parameter_type parameter)
```

You will notice the use of * in documentation. This is a pointer, and it will be covered later in the course. For now, focus on understanding the data types and function usage.

3 size_t and sizeof()

- `size_t`: Unsigned integer type for representing sizes (use `%zu` in printf)
- `sizeof()`: Operator that returns size in bytes

```c
#include <stdio.h>
int main() {
    size_t size_of_int = sizeof(int);
    size_t size_of_short = sizeof(short);
    size_t size_of_char = sizeof(char);
    size_t size_of_float = sizeof(float);
```

```
    size_t size_of_double = sizeof(double);
    // Machine-dependent
    printf("Size of char: %zu bytes\n", size_of_char);
    printf("Size of short: %zu bytes\n", size_of_short);
    printf("Size of int: %zu bytes\n", size_of_int);
    printf("Size of float: %zu bytes\n", size_of_float);
    printf("Size of double: %zu bytes\n", size_of_double);
    return 0;
}
```

4 String Manipulation

These functions are defined in the library string.h. Make sure you have this at the top of your code:

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

4.1 strlen

Function: `size_t strlen(const char *str);`
Returns string length (excluding null terminator).

```
printf("Length: %zu\n", strlen("Hello"));  // Output: 5
```

4.2 strcpy & strncpy

strcpy: `char *strcpy(char *dest, const char *src);`
Copies entire string including null terminator.
strncpy: `char *strncpy(char *dest, const char *src, size_t n);`
Copies up to `n` characters (safer option, defined behaviour).

```
int main() {
    char dest1[20];
    char dest2[5];
    unsigned const int numToCopy = 4;
    strcpy(dest1, "Hello World");           // Full copy
    strncpy(dest2, "McMaster", numToCopy); // Copies "McMa" when numToCopy = 4
    printf("%s\n",dest1);
    printf("%s\n",dest2);
    return 0;

    // What would happen if we change numToCopy to 6?
}
```

4.3 strcat & strncat

strcat: `char *strcat(char *dest, const char *src);`
Appends source to destination.
strncat: `char *strncat(char *dest, const char *src, size_t n);`
Appends up to `n` characters.

```
char text[50] = "Hello";
strcat(text, " World!");     // "Hello World!"
strncat(text, "!!!", 2);     // "Hello World!!"
```

5 String Comparison

5.1 strcmp & strncmp

strcmp: `int strcmp(const char *str1, const char *str2);`
Returns 0 (equal), -1 (negative), or 1 (positive).

strncmp: `int strncmp(const char *str1, const char *str2, size_t n);`
Compares first `n` characters only.

```
int main() {
    // character by character lexicographic comparison
    printf("%d\n", strcmp("Apple", "Apple"));    //  0, all characters match up to '\0'
    printf("%d\n", strncmp("App", "Apple", 3));  //  0, the first n chars match for n = 3
    printf("%d\n", strcmp("Apple", "Bpple"));    // -1, 'A' < 'B'
    printf("%d\n", strcmp("Apple", "apple"));    // -1, 'A' < 'a'

    // ['A','p','p','l','e','\0']
    // vs. []'A','p','p','\0']
    printf("%d\n", strcmp("Apple", "App"));      //  1, 'l' > '\0', i.e. null char is "smaller"

    return 0;
}
```

6 String Searching

6.1 strchr & strrchr

strchr: `char *strchr(const char *str, int c);`
Finds first occurrence of character `c`.
strrchr: `char *strrchr(const char *str, int c);`
Finds last occurrence of character `c`.

```
int main() {
    const char str[] = "McMaster";
    char *pos1 = strchr(str, 'M');  // Points to first 'M'
    char *pos2 = strrchr(str, 'M'); // Points to second 'M'
    printf("%s\n",pos2);            // Since pos is a pointer, prints string from pos2 until '\0'

    const char lookFor = 'D';
    char *pos = strchr(str, lookFor);
    if (pos) {
        printf("Found character at: %s\n", pos);
    } else {
        printf("Character %c not found in %s\n", lookFor, str);
    }
    return 0;
}
```

6.2 strstr

Function: `char *strstr(const char *haystack, const char *needle);`
Finds first substring occurrence.

```
printf("%s\n", strstr("Hello World", "Wo"));  // "World"
```

6.3 strtok

Function: `char *strtok(char *str, const char *delim);`
Splits string into tokens.

```
int main() {
    char str[] = "C Programming Is Fun";
    char *token = strtok(str, " ");      // The delimiter is ' ' (space)
    while(token != NULL) {
```

```
        printf("%s\n", token);
        token = strtok(NULL, " ");       // Passing in NULL tells it to keep looking for the next delim until '\0'
    }
    return 0;
}
```

7 Memory Handling

7.1 memcpy & memmove

memcpy: `void *memcpy(void *dest, const void *src, size_t n);`
Copies memory blocks (no overlap handling).
memmove: `void *memmove(void *dest, const void *src, size_t n);`
Safe for overlapping memory.

```
int main() {
    char a[100] = "Goodbye";
    // Works for overlap
    memmove(a + 4, a, strlen(a)+1); // Note the strlen+1 to include null char '\0'
    printf("%s\n", a);
    return 0;
}
```

7.2 memcmp & memset

memcmp: `int memcmp(const void *ptr1, const void *ptr2, size_t n);`
Compares memory blocks.
memset: `void *memset(void *ptr, int value, size_t n);`
Fills memory with value.

```
int main() {
    char buf[10];
    memset(buf, 'A', 5);                        // "AAAAA"
    memset(buf + 5, 'B', 3);                    // "AAAAABBB"
    printf("%s\n",buf);
    printf("%d\n", memcmp("AAA", "AAB", 3));  // Negative (-1) since 'A' < 'B'

    int n1[] = {10, 20, 30, 40};
    int n2[] = {10, 20, 35, 40};

    // Compare first sizeof(s1) bytes of s1 and s2
    int res = memcmp(n1, n2, sizeof(n1)); // Using sizeof() to supply the # bytes to memcmp
    // Question: what does sizeof(n1) return?
    // Hint: think about the size of int in memory (typically), and how an array works
    printf("%zu\n",sizeof(n1));

    if (res == 0)
        printf("Arrays are identical");
    else
        printf("Arrays differ");
    return 0;
}
```

8 Miscellaneous

8.1 strdup

Function: `char *strdup(const char *str);`
Creates heap-allocated duplicate.

```
char *copy = strdup("Hello");
free(copy);  // Essential, must free dynamically allocated memory
```

8.2 strerror

Function: `char *strerror(int errnum);`
Returns error description.

```
# include <errno.h> // Necessary library
FILE *f = fopen("missing.txt", "r");
if(!f) printf("Error: %s\n", strerror(errno));
```