perplexity

Strings - Declaring a string variable, Reading and displaying strings, String related library functions – Programs for string matching teach me all this with appropriate points, differences to note, syntax and an example

Here's a comprehensive guide to **strings in C**, covering declaration, reading and displaying, important library functions, and a basic string matching program, with key points and differences:

1. Declaring a String Variable

- **Strings in C** are arrays of characters ending with a null character (`\0`)[1] [2] [3].
- **Syntax:**

```
char str[^20]; // Declares a string of max 19 characters + '\0'
```

- **Initialization Methods:**
  - **String literal (size specified):**

```
char str[^6] = "Hello"; // Compiler adds '\0'
```

  - **String literal (size inferred):**

```
char str[] = "Hello"; // Size is 6
```

  - **Character by character (with '\0'):**

```
char str[^6] = {'H','e','l','l','o','\0'};
```

  - **Pointer to string literal:**

```
char *str = "Hello"; // Points to read-only memory
```

- **Difference:**
  Using a char array (`char str[]`) creates a modifiable string, while a pointer to a string literal (`char *str`) points to read-only memory and should not be modified[1] [2].

2. Reading and Displaying Strings

Reading Strings

- **Using** `scanf()` (reads until whitespace):

```
char name[^50];
scanf("%s", name);
```

  Limitation: Stops at first whitespace[4] [2].

- **Using** `fgets()` (reads a line, including spaces):

```
fgets(name, sizeof(name), stdin);
```

  Advantage: Reads spaces and is safer (prevents buffer overflow)[4].

- **Using** `gets()` (unsafe, not recommended):

```
gets(name);
```

  Note: Deprecated due to buffer overflow risks[5].

Displaying Strings

- **Using** `printf()`:

```
printf("%s", name); // %s for string output[^1][^2][^7]
```

- **Using** `puts()`:

```
puts(name); // Adds a newline after output[^2][^10]
```

- **Using loops:**

```
for(int i = 0; name[i] != '\0'; i++)
    putchar(name[i]);
```

3. String-Related Library Functions

Include the header:

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

Function	Purpose	Example Usage
`strlen()`	Get length of string (excluding '\0')	`int len = strlen(str);`
`strcpy()`	Copy one string to another	`strcpy(dest, src);`

Function	Purpose	Example Usage
`strcat()`	Concatenate two strings	`strcat(dest, src);`
`strcmp()`	Compare two strings	`int cmp = strcmp(str1, str2);`
`strchr()`	Find first occurrence of char	`char *p = strchr(str, 'a');`
`strstr()`	Find substring within string	`char *p = strstr(str, "sub");`
`strncpy()`	Copy n characters	`strncpy(dest, src, n);`
`strncmp()`	Compare n characters	`strncmp(str1, str2, n);`

Examples:

```
char a[^20] = "Hello";
char b[^20];
strcpy(b, a); // b = "Hello"
strcat(a, " World"); // a = "Hello World"
int len = strlen(a); // len = 11
int cmp = strcmp(a, b); // cmp != 0
```

4. String Matching Program (Naive Search)

This checks if a substring exists within another string and returns its position:

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

int match(char text[], char pattern[]) {
    int text_length = strlen(text);
    int pattern_length = strlen(pattern);
    for (int i = 0; i <= text_length - pattern_length; i++) {
        int j;
        for (j = 0; j < pattern_length; j++) {
            if (text[i + j] != pattern[j])
                break;
        }
        if (j == pattern_length)
            return i; // Match found at position i
    }
    return -1; // Not found
}

int main() {
    char text[^100], pattern[^50];
    printf("Enter text: ");
    fgets(text, sizeof(text), stdin);
    printf("Enter pattern: ");
    fgets(pattern, sizeof(pattern), stdin);
    // Remove newline characters if present
    text[strcspn(text, "\n")] = '\0';
    pattern[strcspn(pattern, "\n")] = '\0';
```

```
    int pos = match(text, pattern);
    if (pos != -1)
        printf("Found at position: %d\n", pos + 1);
    else
        printf("Not found.\n");
    return 0;
}
```

5. Key Points & Differences

- **Declaration:** Always ensure space for the null terminator (`\0`)[1] [2].
- **Reading:** `scanf()` reads till whitespace; `fgets()` reads the whole line[4] [2].
- **Displaying:** Use `%s` with `printf()` or `puts()` for strings[6] [5].
- **Assignment:** Cannot assign a string to a char array after declaration; use `strcpy()` instead[2].
- **String Functions:** Use functions from `<string.h>` for safe and efficient string operations[7] [8] [5].
- **String Matching:** Can be done manually (as above) or with `strstr()` for simple substring search[7].

In summary:
C strings are character arrays ending with `\0`. Use `scanf()` or `fgets()` to read, and `printf()` or `puts()` to display. Manipulate strings using functions like `strlen`, `strcpy`, `strcat`, and `strcmp` from `<string.h>`. String matching can be implemented with loops or the `strstr()` function.

⁂

1. https://www.w3schools.com/c/c_strings.php
2. https://www.scaler.com/topics/c/c-string-declaration/
3. https://www.guru99.com/c-strings.html
4. https://www.shiksha.com/online-courses/articles/c-strings-with-examples/
5. https://www.programiz.com/c-programming/string-handling-functions
6. https://www.scaler.com/topics/how-to-print-string-in-c/
7. https://www.w3schools.com/c/c_ref_string.php
8. https://www.tutorialspoint.com/explain-string-library-functions-with-suitable-examples-in-c