Characters and Strings

char

- Type that stores a single character
- A literal char is surrounded by single quotes

Output and Input

```c
#include <stdio.h>

int main() {
    char character = 'c';

    printf("%c\n", character);

    return 0;
}
```

- Use %c with printf() and scanf()

Output and Input

```c
#include <stdio.h>

int main() {
    char character = 'c';

    putchar(character);
    putchar(character);
    putchar(character);

    return 0;
}
```

- Use putchar() to print a single character

Output and Input

```c
#include <stdio.h>

int main() {
    char character = 'c';

    putchar(character);
    putchar(character);
    putchar(character);

    putchar('\n');


    return 0;
}
```

- Escape characters with a backslash (\)
  - '\n' is a newline
  - '\t' is a tab

```c
#include <stdio.h>

int main() {

    char user_input = getchar();

    putchar(user_input);
    putchar('\n');

    return 0;
}
```

- Use getchar() to read a single character

char size

- One byte in size
  - 8 bits
  - 2^8 = 256 different values
  - Can only represent English characters, certain characters with accent marks, punctuation marks, numbers, and some other special characters

String

```c
#include <stdio.h>

int main() {

    char string1[] = {'H', 'e', 'l', 'l', 'o', '\0'};

    printf("Here is string1: %s\n", string1);

    return 0;
}
```

- A string is an array of characters (values of type char)
- A string array is null-terminated, which means its last element is the null character which is represented by a '\0’.

```c
#include <stdio.h>

int main() {

    char string1[] = {'H', 'e', 'l', 'l', 'o', '\0'};
    char string2[] = "Hello";

    printf("Here is string1: %s\n", string1);
    printf("Here is string2: %s\n", string2);

    return 0;
}
```

- Both of these initializations will create arrays with 6 elements.
- Using double quotes as with "Hello" automatically puts the '\0' at the end of the string

String

- Only the characters before the first '\0' in the array are considered part of the string
- The length of the string is the number of characters before the first '\0' in the array.

Oversized String

```c
#include <stdio.h>

int main() {
    char oversized_string[100] = "Goodbye";

    printf("Here is the oversized string: %s\n", oversized_string);

    return 0;
}
```

- This creates an array with 100 elements
- Only the characters before the first '\0' in the array are considered part of the string
  - Only the first 8 will be used to store the string.
- The remaining characters will be uninitialized
  - will be ignored by any functions dealing with strings
  - everything after '\0' is ignored

Oversized String

```c
#include <stdio.h>

int main() {

    char oversized_string[100] = "Goodbye";

    printf("Here is the oversized string: %s\n", oversized_string);

    oversized_string[50] = 'Z';

    printf("Oversized string after modification: %s\n", oversized_string);

    return 0;
}
```

- This will change one of the elements of the array,
- It will not change the string, because it is inserted after the null character

strlen()

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

int main() {

    char string1[] = {'H', 'e', 'l', 'l', 'o', '\0'};
    char string2[] = "Hello";

    char oversized_string[100] = "Goodbye";

    printf("Here is the oversized string: %s\n", oversized_string);

    oversized_string[50] = 'Z';

    printf("Oversized string after modification: %s\n", oversized_string);

    size_t string1_length = strlen(string1);

    printf("The length of string1 is %zu\n", string1_length);
    printf("The length of string2 is %zu\n", strlen(string2));
    printf("The length of oversized_string is %zu\n", strlen(oversized_string));

    return 0;
}
```

- strlen() will get the length of a string up to, but not including, the
- first null character in the array

String Input/Output

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

int main() {



    char user_input[100];
    printf("Enter a string: ");

    scanf("%s", user_input);

    printf("You entered \"%s\"\n", user_input);

    return 0;
}
```

- No need to use & when calling scanf() with strings, because
- scanf() can modify the array that you pass it
- **scanf() will read up to the first time it encounters whitespace (spaces, tabs, newlines)**

Repeat String

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

int main() {


    char user_input[100];
    printf("Enter a string: ");

    scanf("%s", user_input);

    printf("You entered \"%s\"\n", user_input);

    repeat_string(user_input);

    printf("Your string repeated is \"%s\"\n", user_input);

    return 0;
}
```

```c
void repeat_string(char string[]) {
    size_t original_length = strlen(string);

    for (size_t i = 0; i < original_length; i++) {
        string[i + original_length] = string[i];
    }

    string[original_length * 2] = '\0';
}
```

Repeat String

```c
void repeat_string(char string[]) {
    size_t original_length = strlen(string);

    for (size_t i = 0; i < original_length; i++) {
        string[i + original_length] = string[i];
    }

    string[original_length * 2] = '\0';
}
```

c	h	a	r	\0					
0	1	2	3	4	5	6	7	8	.....

c	h	a	r	c	h	a	r	\0	
0	1	2	3	4	5	6	7	8	.....

Reverse String

Separate string

original

b	i	r	d	\0
0	1	2	3	4

reversed

In-place

b	i	r	d	\0
0	1	2	3	4