Féidearthachtaí as Cuimse
Infinite Possibilities

Semester 2
Week 5 - Tutorial

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TU DUBLIN
TECHNOLOGICAL UNIVERSITY DUBLIN

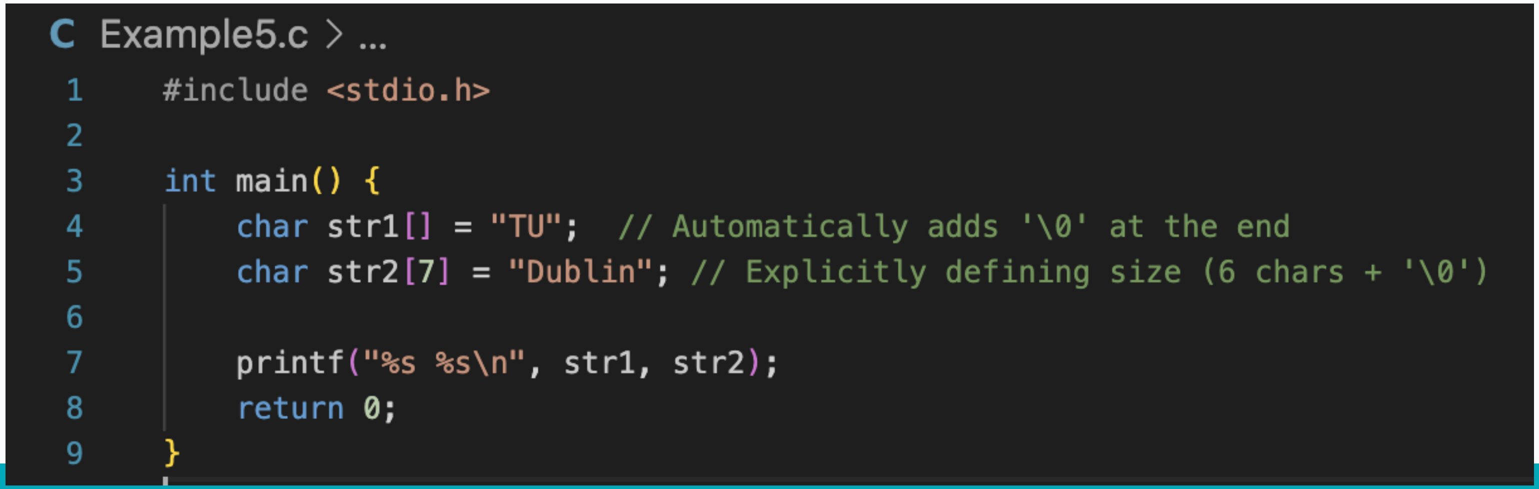
Programming - Week 5 – 24th February 2025

Overview

- Strings – revision
- Fgets()
- Strlen()
- Strcat()
- Strcmp()
- Strcpy()
- Reverse string example
- Mandatory lab question

Strings

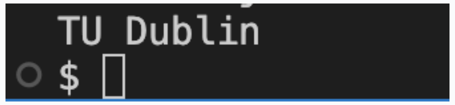
- In C programming, strings are arrays of characters ending with a null character (\0). Unlike other languages, C does not have a built-in string type, so strings are handled using character arrays or pointers. The null terminator (\0) is automatically added.
- char name[] = "Hello"; // 'H' 'e' 'l' 'l' 'o' '\0'



Example5.c > ...

```c
#include <stdio.h>

int main() {
    char str1[] = "TU";  // Automatically adds '\0' at the end
    char str2[7] = "Dublin"; // Explicitly defining size (6 chars + '\0')

    printf("%s %s\n", str1, str2);
    return 0;
}
```

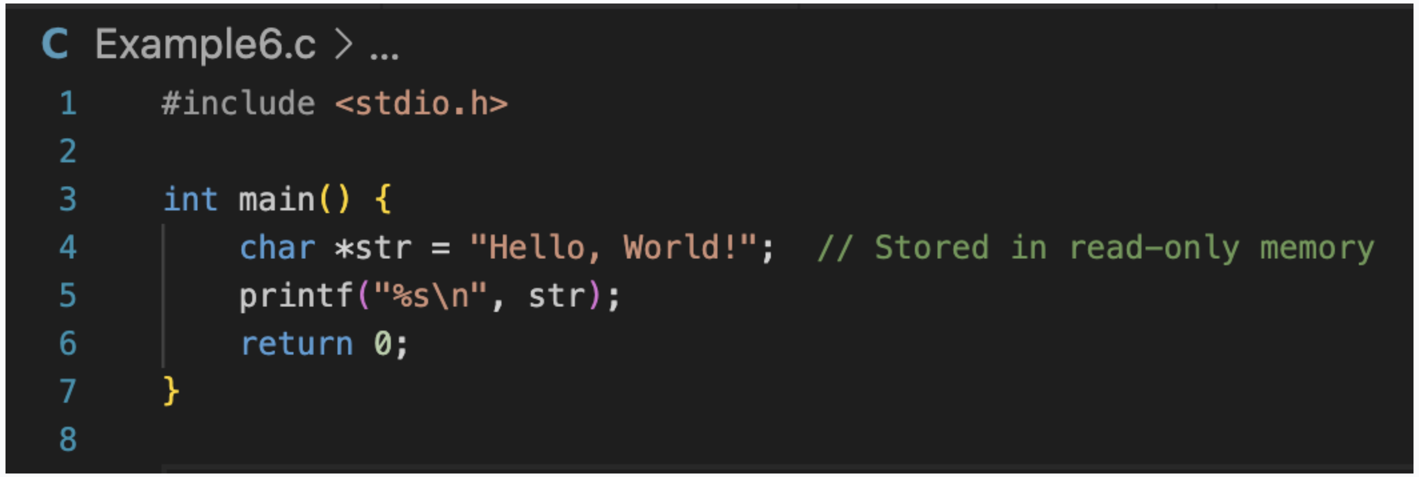
In Terminal:



```
TU Dublin
$ 
```

Character Pointers

- Strings defined with char * are stored in read-only memory, and modifying them causes an error.

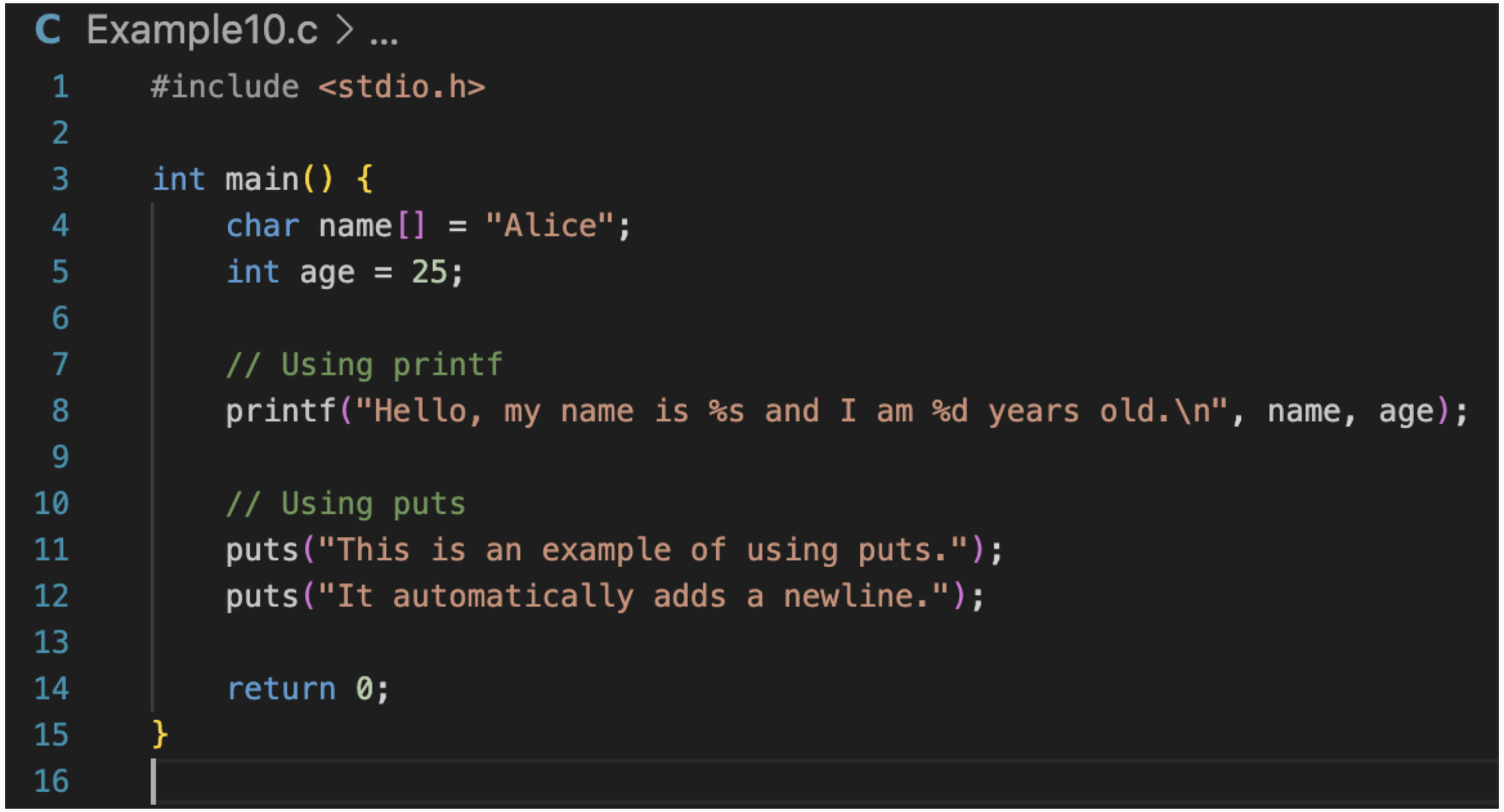


Example6.c > ...

```c
#include <stdio.h>

int main() {
    char *str = "Hello, World!";  // Stored in read-only memory
    printf("%s\n", str);
    return 0;
}

```

String Output

- We use printf and puts to output strings to terminal



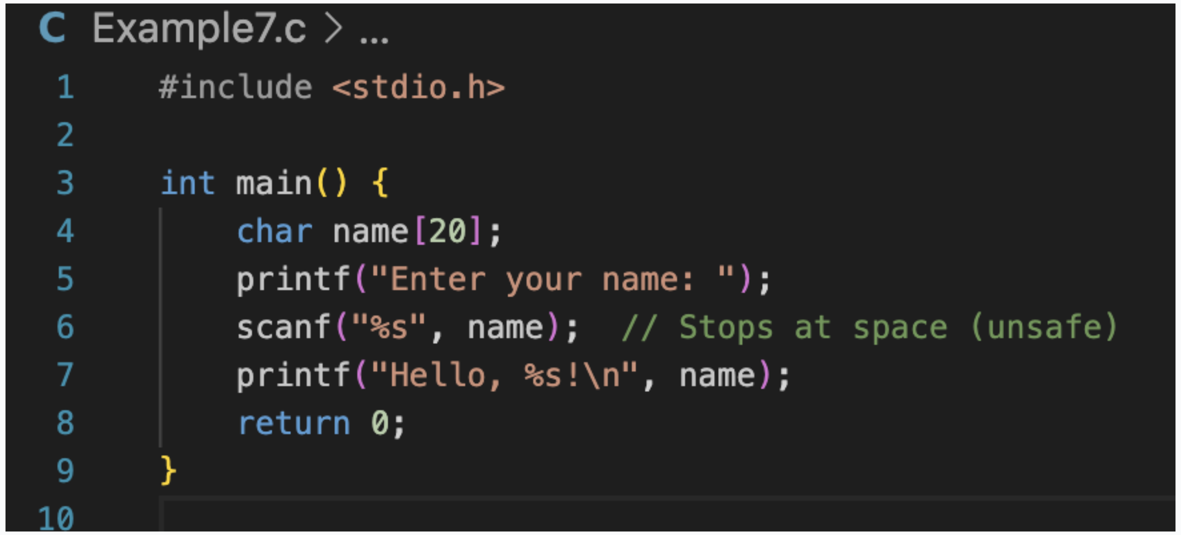
Example10.c > ...

```c
#include <stdio.h>

int main() {
    char name[] = "Alice";
    int age = 25;

    // Using printf
    printf("Hello, my name is %s and I am %d years old.\n", name, age);

    // Using puts
    puts("This is an example of using puts.");
    puts("It automatically adds a newline.");

    return 0;
}
```

Use printf when you need to format output with variables.

Use puts when printing a simple string (it's faster and adds a newline automatically).

String Input: scanf

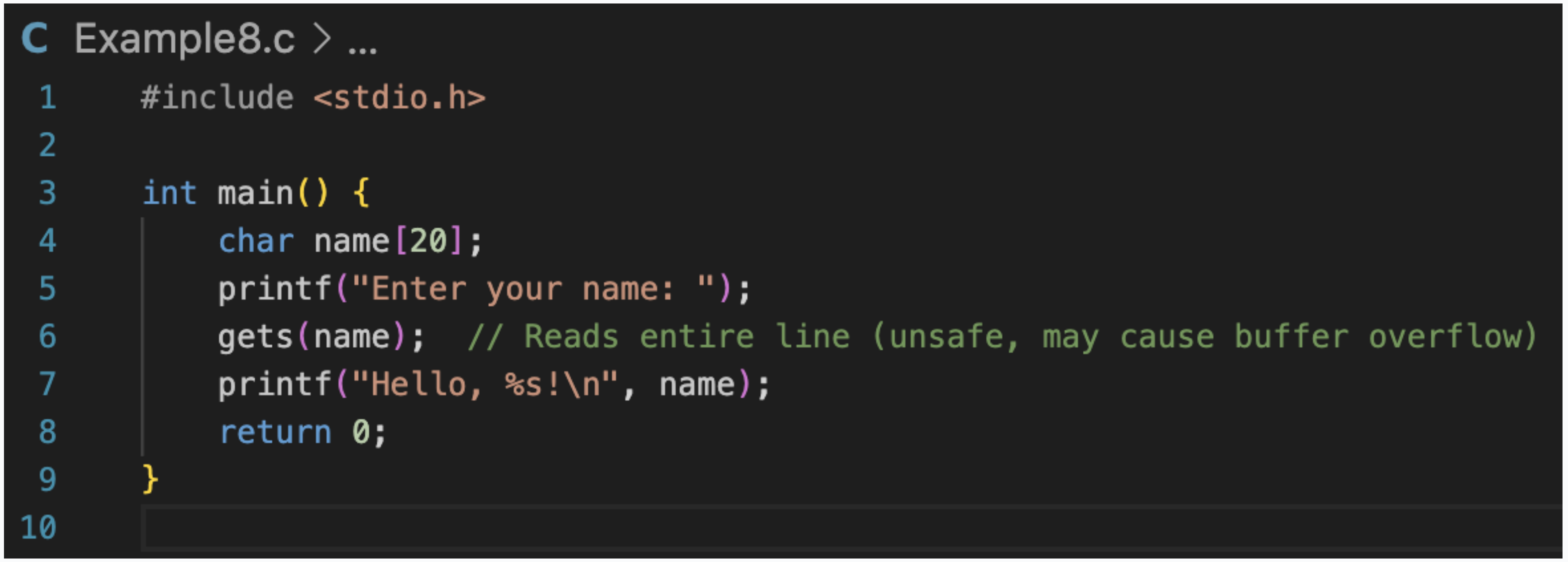


Example7.c > ...

```c
#include <stdio.h>

int main() {
    char name[20];
    printf("Enter your name: ");
    scanf("%s", name);  // Stops at space (unsafe)
    printf("Hello, %s!\n", name);
    return 0;
}
```

- **Problem:** If input is ”Diana Prince", only ”Diana" is stored.
- It only reads up to the space.
- scanf does **not** handle spaces

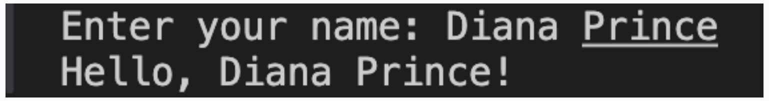
String Input: gets()



Example8.c > ...

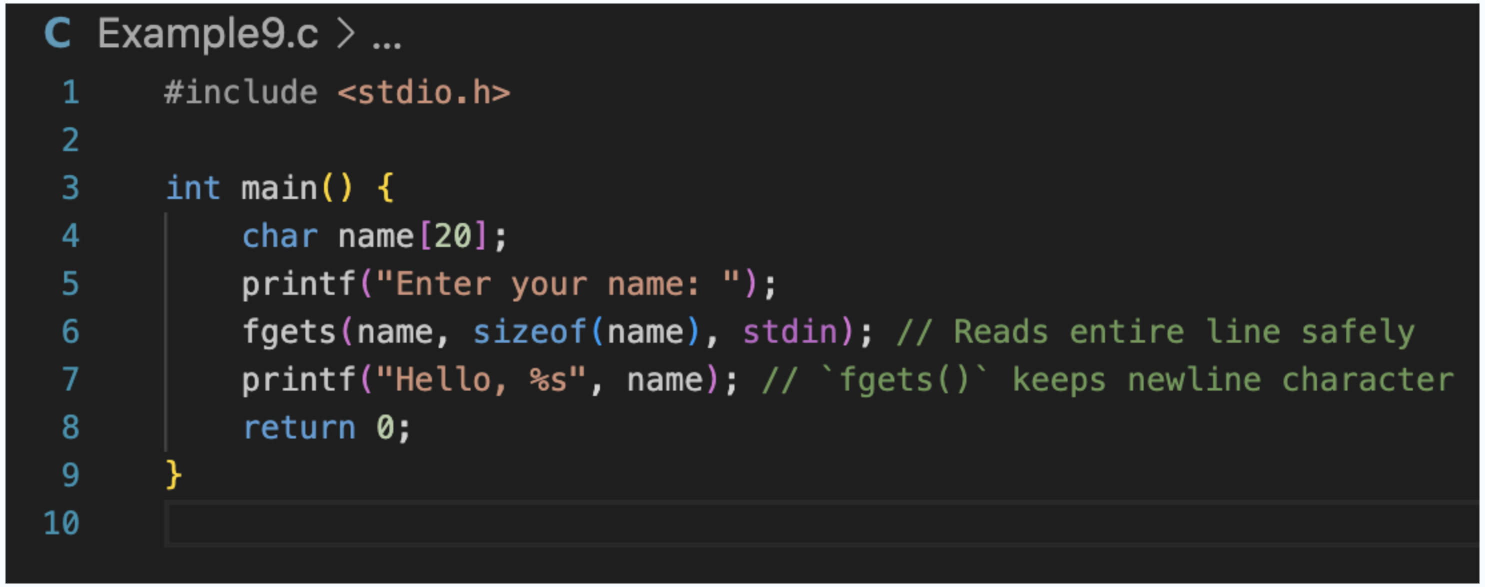
```c
#include <stdio.h>

int main() {
    char name[20];
    printf("Enter your name: ");
    gets(name);  // Reads entire line (unsafe, may cause buffer overflow)
    printf("Hello, %s!\n", name);
    return 0;
}
```

- gets() is unsafe because it does not check for buffer overflow.



```
Enter your name: Diana Prince
Hello, Diana Prince!
```

String Input: fgets()

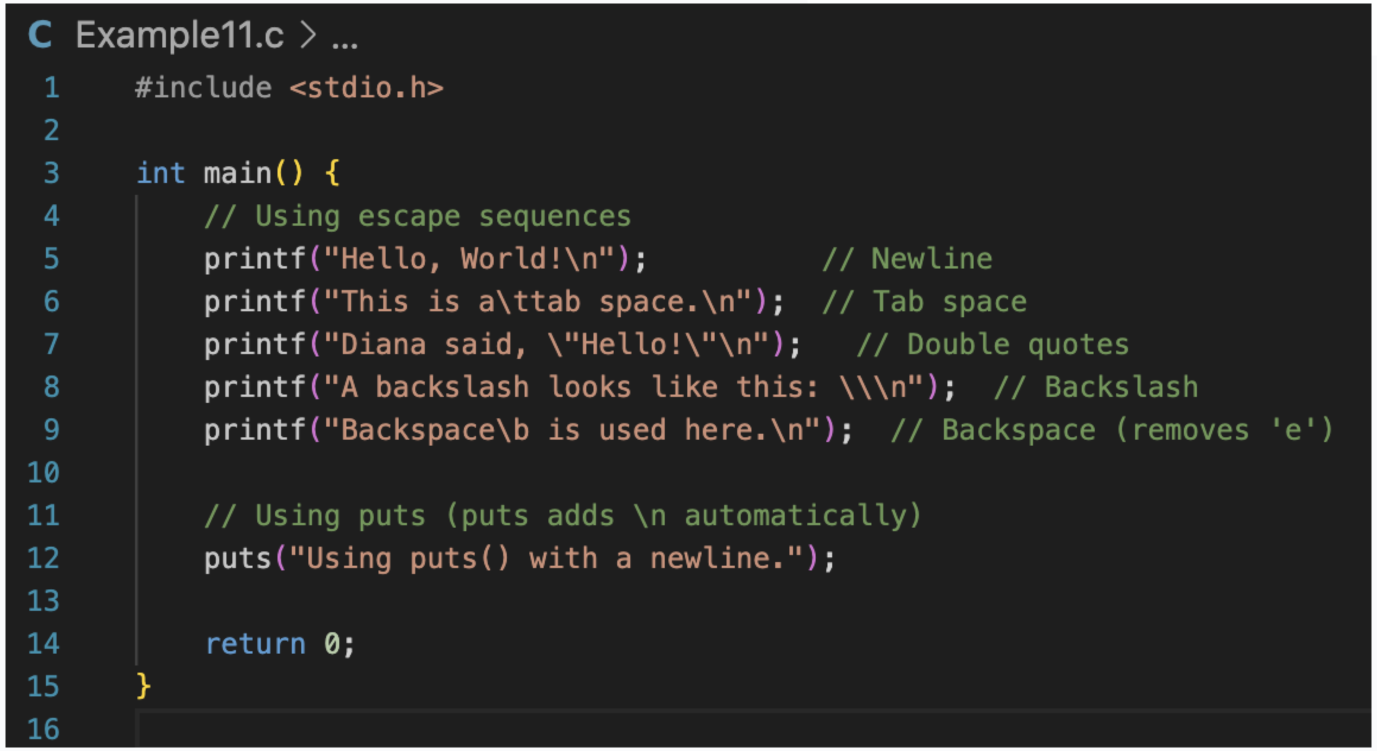


```
C Example9.c > ...
#include <stdio.h>

int main() {
    char name[20];
    printf("Enter your name: ");
    fgets(name, sizeof(name), stdin); // Reads entire line safely
    printf("Hello, %s", name); // `fgets()` keeps newline character
    return 0;
}
```

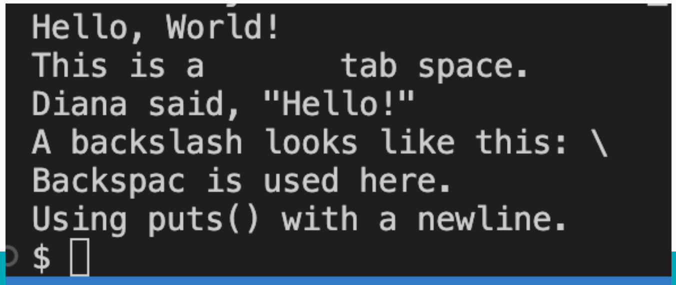
- fgets() prevents buffer overflow and **reads spaces** correctly.

Escape Characters

Character	Meaning
\n	new line
\"	display a double quote
\'	display a single quote
\0	NULL character
\t	display a tab
\b	remove a space, backspace
\a	alert, ping, chime noise

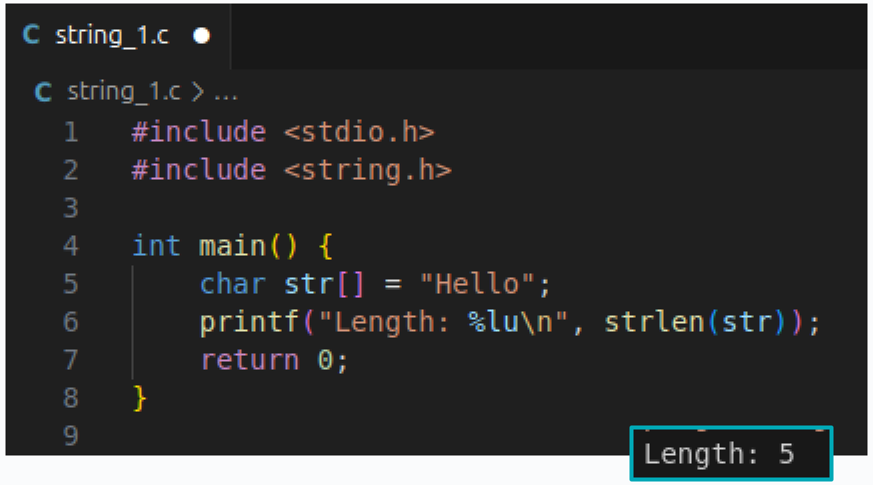

Example11.c > ...

```c
#include <stdio.h>

int main() {
    // Using escape sequences
    printf("Hello, World!\n");            // Newline
    printf("This is a\ttab space.\n");  // Tab space
    printf("Diana said, \"Hello!\"\n");   // Double quotes
    printf("A backslash looks like this: \\\n");  // Backslash
    printf("Backspace\b is used here.\n");  // Backspace (removes 'e')

    // Using puts (puts adds \n automatically)
    puts("Using puts() with a newline.");

    return 0;
}
```



```
Hello, World!
This is a       tab space.
Diana said, "Hello!"
A backslash looks like this: \
Backspac is used here.
Using puts() with a newline.
$ 
```

strlen() – Get String Length

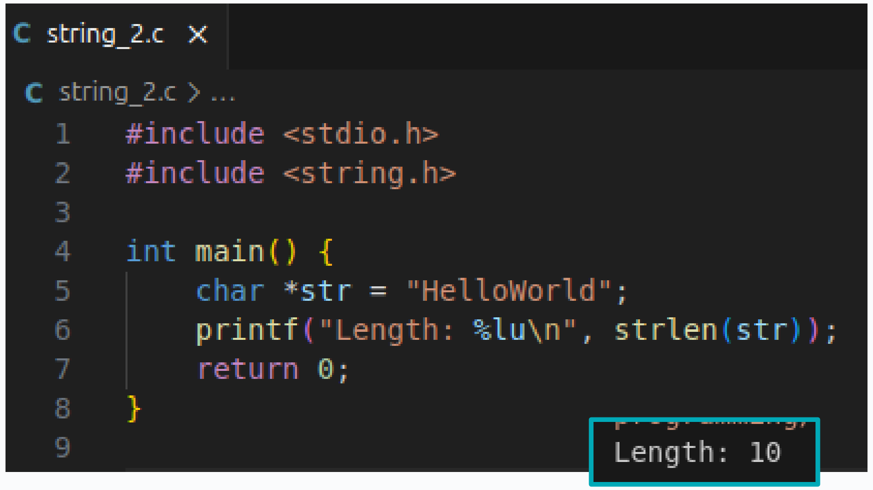


string_1.c

string_1.c > ...

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

int main() {
    char str[] = "Hello";
    printf("Length: %lu\n", strlen(str));
    return 0;
}
```

```
Length: 5
```



string_2.c

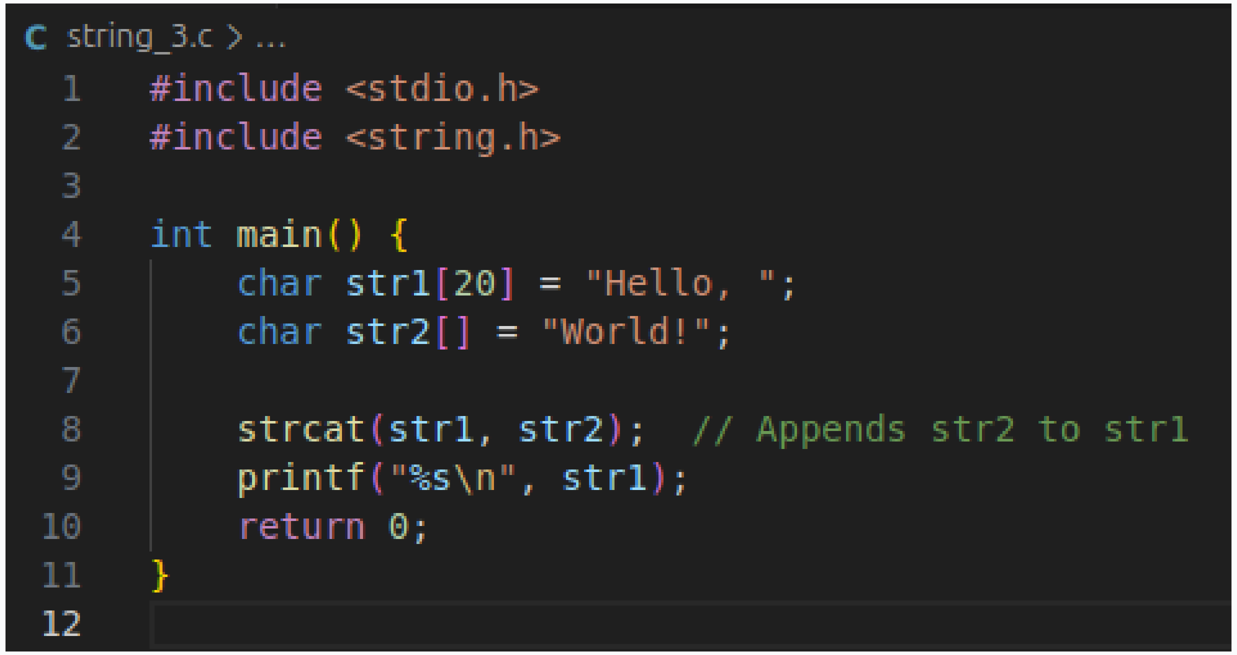
string_2.c > ...

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

int main() {
    char *str = "HelloWorld";
    printf("Length: %lu\n", strlen(str));
    return 0;
}
```

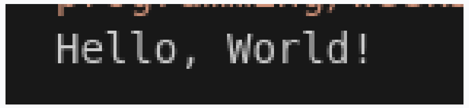
```
Length: 10
```

- The strlen() function will return the length of a string.
- We include this in the program via string.h

strcat() – join two strings



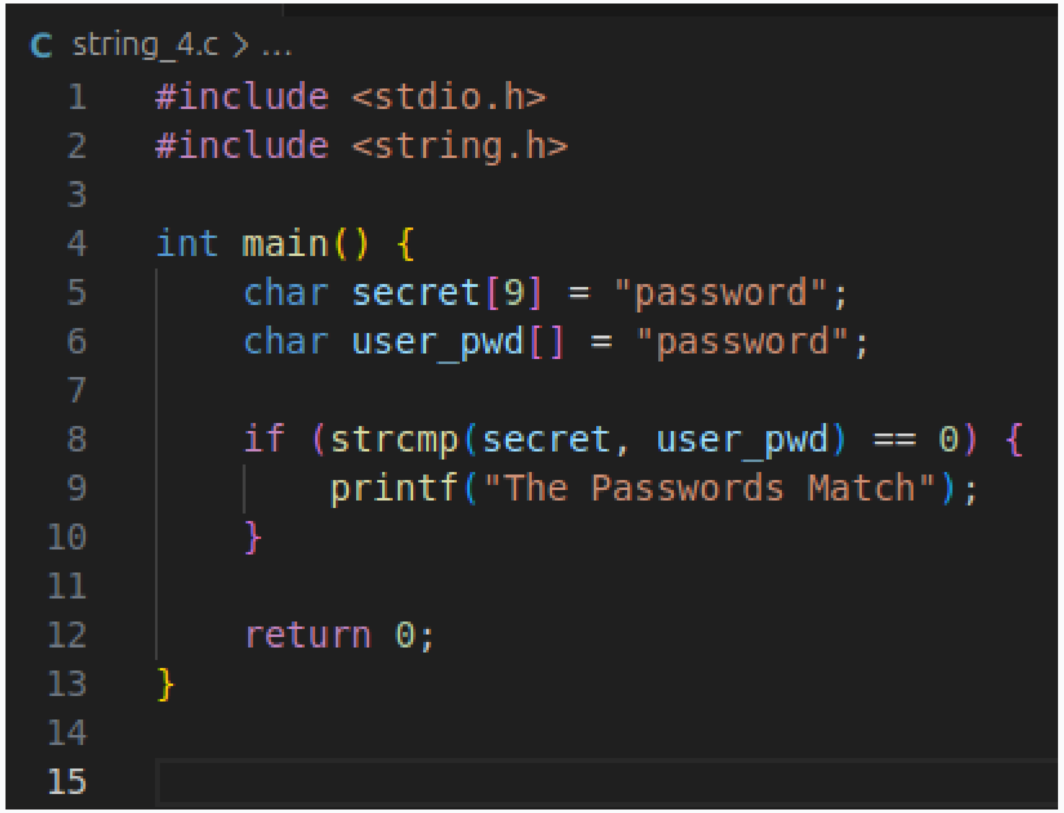
```c
string_3.c > ...
#include <stdio.h>
#include <string.h>

int main() {
    char str1[20] = "Hello, ";
    char str2[] = "World!";

    strcat(str1, str2);  // Appends str2 to str1
    printf("%s\n", str1);
    return 0;
}
```



```
Hello, World!
```

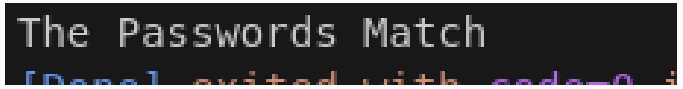
- The strcat() function will append (join) two strings together.
- We include this in the program via string.h
- Ensure str1 has enough space to append the new content

strcmp() – compare strings



string_4.c > ...

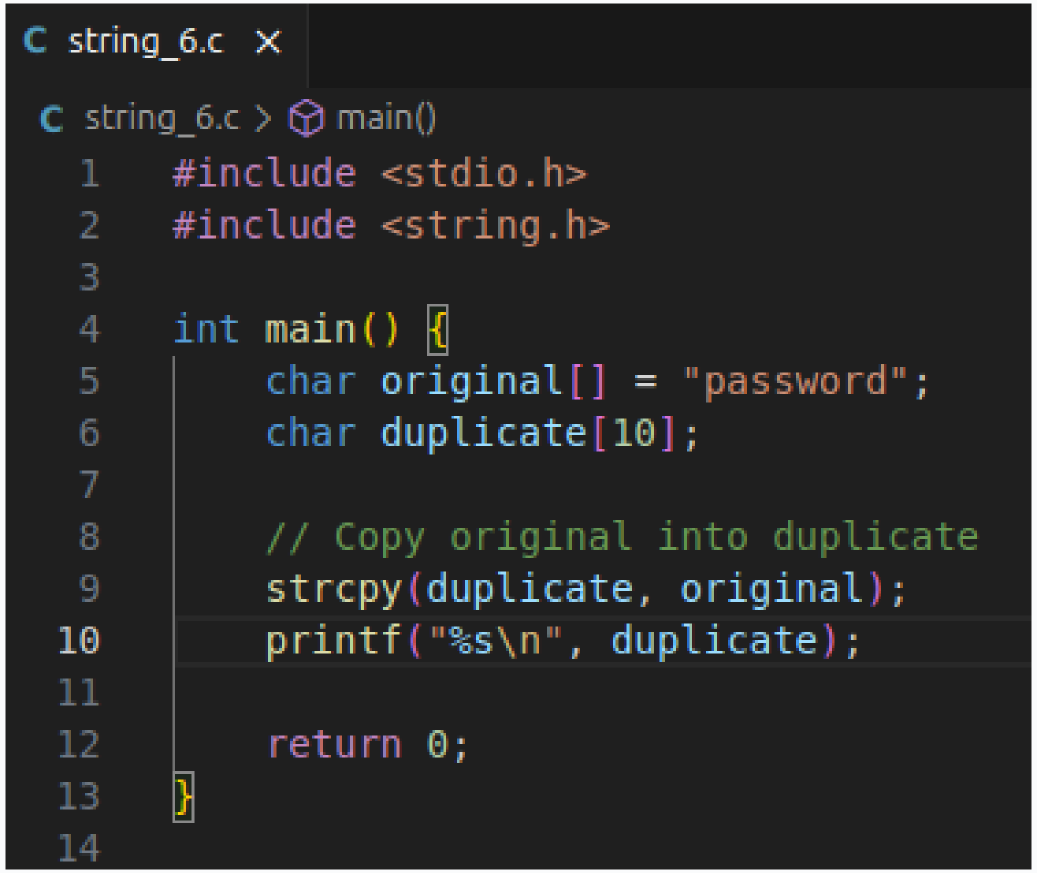
```c
#include <stdio.h>
#include <string.h>

int main() {
    char secret[9] = "password";
    char user_pwd[] = "password";

    if (strcmp(secret, user_pwd) == 0) {
        printf("The Passwords Match");
    }

    return 0;
}

```



```
The Passwords Match
```

- strcmp() lets us compare two strings.
- strcmp returns a value:
- Returns:
    - 0 if equal
    - < 0 if first < second
    - > 0 if first > second

strcpy() – copy a string

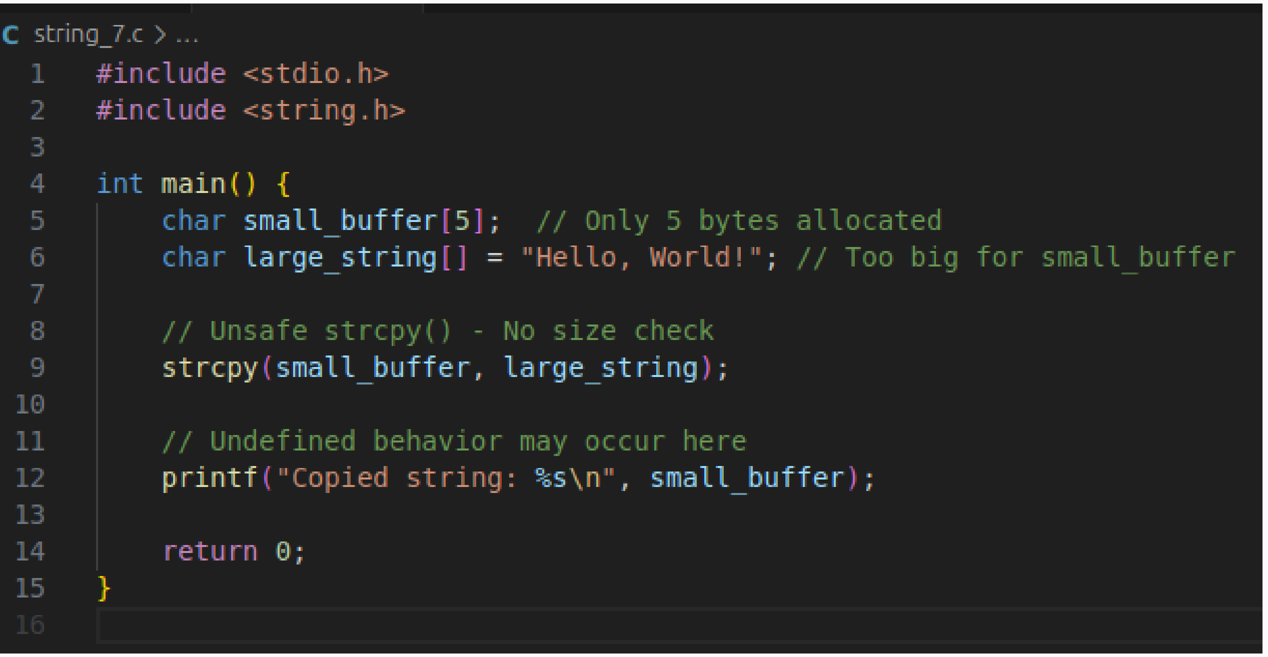

string_6.c

string_6.c > main()

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

int main() {
    char original[] = "password";
    char duplicate[10];

    // Copy original into duplicate
    strcpy(duplicate, original);
    printf("%s\n", duplicate);

    return 0;
}

```

- strcpy() function copies a string to a different location
- strcpy() does not check destination size. Use strncpy() for safety.

Buffer sizes



string_7.c > ...

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

int main() {
    char small_buffer[5];  // Only 5 bytes allocated
    char large_string[] = "Hello, World!"; // Too big for small_buffer

    // Unsafe strcpy() - No size check
    strcpy(small_buffer, large_string);

    // Undefined behavior may occur here
    printf("Copied string: %s\n", small_buffer);

    return 0;
}
```

- **What happens if the destination is smaller than the source?**
- We may experience issues with the program execution.
- If the destination buffer is too small to hold the source string, strcpy() will overwrite adjacent memory, causing buffer overflow. This leads to undefined behavior, which can cause:
  - **Crashes** (Segmentation Fault)
  - **Data Corruption**
  - Security Vulnerabilities (Exploitable by hackers)

Summary

Function	Description
strlen(str)	Get length of string
strcpy(dest, src)	Copy src to dest
strcat(str1, str2)	Append str2 to str1
strcmp(str1, str2)	Compare two strings
fgets(str, size, stdin)	Read multi-word input

Problem Solving

- Create a function reverse a string
- General operation
- Pass a string to a function
- The function reverses the string
- Display the reversed string in main

Mandatory Question – in class solution

- Write a program to read in your name and display it with a space between each letter.
- E.g.
- Diana
- D i a n a

Questions