

The background of the slide features a photograph of the Alma Mater statue at the University of Illinois, which depicts a woman in classical robes holding a book and a torch. The entire image is overlaid with a semi-transparent red filter. The text is positioned in the upper left and lower left areas of the slide.

MP1 Overview Session

CS 240 - The University of Illinois

Eunice Zhou

January 24, 2022

Goals

In this MP, you will get comfortable with programming in C and learn how to:

- work with strings, memory allocation, pointers
- create data structures
- read files
- manipulate data at the bit- and byte-level

Programming in C

A photograph of a crowd of people gathered around a statue of Alma Mater, overlaid with a semi-transparent orange filter. The text "Programming in C" is centered in white. The background shows a large group of people, some looking towards the camera and others looking towards the statue. The statue is a large, classical-style figure, possibly representing a personification of wisdom or knowledge. The overall scene is outdoors, with trees and foliage visible in the background.

Memory Allocation in C

- *void* malloc (size_t size)* - allocate heap memory
- *void* calloc (size_t num, size_t size)* - allocate heap memory and initialize all bits to zero
- *sizeof()* - return the size of the object or the type
 - E.g. *sizeof(char) == 1*
- *void free (void* ptr)* - deallocate previously allocated memory

String in C

A string in C is a sequence of characters with a terminating null byte

- There is no string type in C (*string s*; ❌)
- A C-string is represented by a character pointer
 - *char s[3] = "HI";*

H	I	\0
---	---	----
- Useful functions: *strlen*, *strcpy* / *strncpy*, *strcmp* / *strncmp*, *strcat* / *strncat*

Working with File in C

Useful functions:

- *fopen*, *fclose* - open/close a file
- *fread*, *fwrite* - read from/write to file
- *fseek* - set the position of the file pointer
- *ftell* - return the current position of the file pointer

Remember to check for errors!

Emoji & UTF-8

The background of the slide features a photograph of a large, classical-style statue of a woman in a long, flowing dress, standing on a pedestal. The statue is surrounded by a crowd of people, some of whom are looking towards the camera. The entire image is overlaid with a semi-transparent orange color, which serves as a background for the white text.

UTF-8

- Unicode Transformation Format
- Variable-length character encoding
- Encode character using 1 - 4 bytes
- Can encode a lot more characters than ASCII
- In this MP, you will work with UTF-8 encoded string and manipulate data at byte-level

Emoji

In UTF-8, an emoji is encoded using either 3 bytes ($U+203C$ - $U+3299$) or 4 bytes ($U+1F000$ - $U+1FAFF$)

For this MP, we will only consider emojis in the inclusive range of $U+1F000$ to $U+1FAFF$

For example,  ($U+1F499$) has the byte sequence $0xF0$ $0x9F$ $0x92$ $0x99$

MP - Part 1



Simple Emoji Functions

Implement six functions in *emoji.c*:

- *emoji_favorite* - return your favorite emoji
 - “❤️”
 - “\u1F499”
 - “\xF0\x9F\x92\x99”

Simple Emoji Functions

Implement six functions in *emoji.c*:

- *emoji_count* - count the number of emojis in a UTF-8 string

Simple Emoji Functions

Implement six functions in *emoji.c*:

- *emoji_random_alloc* - generate a random emoji
 - rand() for generating random number
 - allocate heap memory for the emoji
 - make sure to return a valid C-string

Simple Emoji Functions

Implement six functions in *emoji.c*:

- *emoji_invertChar* - invert an emoji
 - invert only the first character in the string
 - invert 😊 to some sad face
 - invert five more emojis of your choice

Simple Emoji Functions

Implement six functions in *emoji.c*:

- *emoji_invertAll* - invert all the emojis in a string using *emoji_invertChar*

Simple Emoji Functions

Implement six functions in *emoji.c*:

- *emoji_invertFile_alloc* - read the contents of a file, invert all the emojis, and return the inverted string

MP - Part 2



An Emoji Translator

Implement four functions in *emoji-translate.c*:

- *emoji_init* - initialize an *emoji_t* object
 - add any member variables an *emoji_t* object might have in *emoji-translate.h*

An Emoji Translator

Implement four functions in *emoji-translate.c*:

- *emoji_add_translation* - add a translation to the *emoji_t* object
 - E.g. “😊” → “happy”

An Emoji Translator

Implement four functions in *emoji-translate.c*:

- *emoji_translate_file_alloc* - translate the content of a file using all the translation rules added so far
 - when multiple rules match, choose the one with the longest emoji string
 - E.g. “😊😊” → “very happy” >> “😊” → “happy”

An Emoji Translator

Implement four functions in *emoji-translate.c*:

- *emoji_destroy* - destroy an *emoji_t* object and deallocate any memory associated with the object

An Emoji Translator

Example Usage

```
emoji_t emoji;
emoji_init(&emoji);

emoji_add_translation(&emoji, "❤️", "heart");

// The file on disk contains: "I ❤️💙 Illinois!"
unsigned char *translation = emoji_translate_file_alloc(&emoji, "tests/txt/simple.txt");

// Translation Output: "I heart💙 Illinois!"
printf("%s\n");

emoji_destroy(&emoji);
```

Memory Correctness

Your code need to run “valgrind clean”:

- Zero memory error, no memory leak
- *free()* any memory allocated with *malloc/calloc*
- *fclose()* any file opened with *fopen*