Spark Session: ft_printf

Project description:

> *Recode printf*

Topics

1. Variadic Arguments
2. Function Pointers

Variadic Arguments

1. Variadic functions add flexibility to your code by allowing an unknown number of arguments. (30 mins)
    - What would its prototype look like? (5 mins)
    - Identify the 4 macros that allow you to access these arguments. (25 mins)
        - What are the argument types?
        - What are default argument promotions?

2. Let's practice accessing and carrying out operations on a variable argument list! (30 mins)
    - Write a variadic function that:
        - has a prototype of `function(const int n, ...)` **n** being the number of arguments in the list,
        - returns the **sum** of the integers in that list.
    - Write the accompanying main to test your function. Example test: does `yourfunction(3, 40, 5, -3)` return `42`?

Break (5 mins)

Function Pointers

1. Just as we can have pointers to data (char *, int *), we can have pointers to functions. (35 mins)
    - How do we declare a pointer to a function? Pay attention to bracket placement! (15 mins)
        - Let's break down the syntax. What does each part of the declaration mean?
        - What is the value stored in the function pointer? What is the type here?
        - Is there a difference between `void (*fn)` and `void *fn`?
    - When can function pointers come in handy? (10 mins)
    - Like normal pointers, we can also have an array of function pointers. What is their syntax? (10 mins)
2. Let's practice using a function pointer! (30 mins)
    - Write a function that: (10 mins)
        - takes an integer **n** as argument,
        - prints "Hello" **n** times,
        - returns nothing.
    - Now write an accompanying main that: (20 mins)

        - declares a pointer to a function that takes an int and returns nothing,
        - initialises that pointer to the Hello function you just wrote,
        - calls that function 3 times **using the function pointer**.

Break (5 mins)

3. Now let's try doing something cooler with an array of function pointers. (20 mins)
    - Here's some code to get your started:

      ```
      enum    e_op
      {
          PLUS = 0, MINUS
      };

      void    operation_add(int a, int b)
      {
          printf("%d + %d = %d\n", a, b, a + b);
      }

      void    operation_minus(int a, int b)
      {
          printf("%d - %d = %d\n", a, b, a - b);
      }
      ```
    - Write a main that:
        - declares an array of 2 function pointers, taking 2 ints and returning nothing,
        - assigns the first array element to `operation_add` and the second element to `operation_minus`,
        - calls each function at least once through the array. *Hint: enums can make indexing easier.*