2.7 EXERCISES

giovedì 29 febbraio 2024 06:18

Exercise: Analyzing Floating-Point Representation in C

Objective: Write a C program to dissect and print the components of a single-precision floating-point number according to the IEEE 754 standard. Your program should extract and display the sign bit, the exponent, and the fraction (mantissa) of the given floating-point number.

Background: In the IEEE 754 standard for floating-point arithmetic, a single-precision floating-point number (float in C) is represented using 32 bits, divided into three parts:

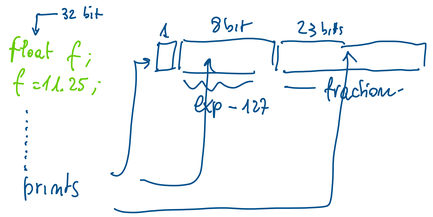
- 1 bit for the sign (0 for positive, 1 for negative),
- 8 bits for the exponent, biased by 127,
- 23 bits for the fraction (mantissa), representing the significant digits of the number.

Requirements:

1. Your program should accept a single floating-point number as input. You may hard-code this number into your program for simplicity.
2. Implement a function named printFloatParts that takes a float as an argument and prints its sign, exponent (in biased form), and fraction (mantissa).
   - The sign should be printed as an unsigned integer (0 or 1).
   - The exponent should be printed as an unsigned integer in its biased form.
   - The fraction should be printed as a hexadecimal number to represent the exact bits of the mantissa.
3. You must use pointer casting to access the binary representation of the floating-point number as an unsigned 32-bit integer. Do not use unions or any form of array manipulation for this task.
4. Ensure your program properly handles both positive and negative floating-point values.

Hints:

- Remember that the sign bit is the most significant bit (MSB), the exponent follows the sign bit, and the fraction (mantissa) occupies the least significant 23 bits.
- Use bitwise operations to extract each component from the 32-bit representation.
- To access the binary representation, cast the address of the floating-point number to a pointer of type uint32_t and then dereference it.



Exercise: Stack Heap and Static Area

Write a C program that demonstrates the different memory areas used in a C application: the static area, the heap, and the stack. Your program will declare variables in each of these areas and print their addresses to show where each type of variable is located in memory by tracking heap and stack grow direction.