#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <string.h>

#include <math.h>

#define EXPO\_BIT 11

#define SIG\_BIT 52

typedef struct t\_stack

{

    char string;

    struct t\_stack \*next;

} t\_stack;

void push(t\_stack \*\*top, char stream);

void pop(t\_stack \*\*top);

char peek(t\_stack \*\*top);

void decimalToBinary(double decimal, t\_stack \*\*stack);

bool SignBitIsNegative(char number);

void pause(void);

int main()

{

    t\_stack \*stackForward = NULL, \*stackBackward = NULL;

    char buffer[10000] = {'\0'};

    char \*token;

    printf("Input number (decimal): ");

    if (fgets(buffer, sizeof(buffer), stdin) == NULL) {

        printf("Error reading input.\n");

        return 1;

    }

    if (buffer[strlen(buffer) - 1] == '\n') {

        buffer[strlen(buffer) - 1] = '\0';

    }

    double number = atof(buffer);

    // Append Sign Bit

    char signBit = SignBitIsNegative(buffer[0]) ? '1' : '0';

    push(&stackBackward, signBit);

    if (isnan(number)) // Handle NaN case

    {

        for (int i = 0; i < EXPO\_BIT; i++)

        {

            push(&stackBackward, '1');

        }

        for (int i = 0; i < SIG\_BIT; i++)

        {

            push(&stackBackward, '1');

        }

    }

    else if (isinf(number)) // Handle infinity case

    {

        // Set exponent bits to all 1

        for (int i = 0; i < EXPO\_BIT; i++)

        {

            push(&stackBackward, '1');

        }

        // Set mantissa bits to all 0

        for (int i = 0; i < SIG\_BIT; i++)

        {

            push(&stackBackward, '0');

        }

    }

    else if (number == 0.0) // Handle 0 case

    {

        push(&stackBackward, '0'); // Sign bit

        for (int i = 0; i < EXPO\_BIT; i++)

        {

            push(&stackBackward, '0'); // Exponent bits (all set to 0)

        }

        for (int i = 0; i < SIG\_BIT; i++)

        {

            push(&stackBackward, '0'); // Push trailing zero

        }

    }

    else

    {

    decimalToBinary(number, &stackBackward);

}

    // Reverse stack

    while (stackBackward != NULL)

    {

        push(&stackForward, peek(&stackBackward));

        pop(&stackBackward);

    }

    // Print the IEEE 754 representation from the forward stack

    printf("Binary Representation: ");

    while (stackForward != NULL)

    {

        printf("%c", peek(&stackForward));

        pop(&stackForward);

    }

    printf("\n");

    pause();

    return 0;

}

void push(t\_stack \*\*top, char stream)

{

    t\_stack \*newNode = (t\_stack \*)calloc(1, sizeof(t\_stack));

    newNode->string = stream;

    if (\*top == NULL)

    {

        newNode->next = NULL;

        \*top = newNode;

    }

    else

    {

        newNode->next = \*top;

        \*top = newNode;

    }

}

void pop(t\_stack \*\*top)

{

    if (\*top == NULL)

    {

        printf("STACK UNDERFLOW\n");

        return;

    }

    t\_stack \*node\_to\_free = \*top;

    \*top = (\*top)->next;

    free(node\_to\_free);

}

char peek(t\_stack \*\*top)

{

    if (\*top != NULL)

    {

        return (\*top)->string;

    }

    return '\0';

}

void decimalToBinary(double decimal, t\_stack \*\*stack) {

    unsigned char \*binaryBytes = (unsigned char \*)&decimal;

    for (int byteIndex = sizeof(double) - 1; byteIndex >= 0; byteIndex--) {

        unsigned char byte = binaryBytes[byteIndex];

        for (int bitIndex = 7; bitIndex >= 0; bitIndex--) {

            char bit = (byte >> bitIndex) & 1;

            push(stack, bit + '0');

        }

    }

}

bool SignBitIsNegative(char number)

{

    return number == '-';

}

void pause(void)

{

    printf("Press enter to continue...");

    fflush(stdout);

    getchar();

}