```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>
#define EXPO_BIT 11
#define SIG_BIT 52
#define EXPO_BIAS 1023
#define EXPO_BI_MAX 2047

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);
double abs_double(double num);
double round(double num);
void push_binary(double Num, t_stack **stack, int *zero_count, int *index_first);
void push_frac(double Num, t_stack **stack, int *zero_count, int *index_first);
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): ");
    fgets(buffer, sizeof(buffer), stdin);
    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';
    char space = ' ';
    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))
    {
        // Set exponent bits to all 1s
        for (int i = 0; i < EXPO_BIT; i++)
        {
            push(&stackBackward, '1');
        }
        // Set mantissa bits to all 0s
        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'); // Mantissa bits (all set to 0)
        }
    }
    else
    {
        // Exponent part
        int exponentPart = (int)round(log2(abs_double(number)));
        int biasedExponent = exponentPart + EXPO_BIAS;

        // Convert exponent to binary and push to stack
        for (int i = EXPO_BIT - 1; i >= 0; i--)
        {
            char bit = ((biasedExponent >> i) & 1) + '0';
            push(&stackBackward, bit);
        }
        // Mantissa part
        int trailing_zeros;
        int bin_count = 0;
        int index_first = 0;
        push_binary(number, &stackBackward, &bin_count, &index_first);
        push_frac(number, &stackBackward, &bin_count, &index_first);
        trailing_zeros = 52 - bin_count;
    }
    // Reverse stack
    while (stackBackward != NULL)
    {
        push(&stackForward, peek(&stackBackward));
        pop(&stackBackward);
    }
    char temp[64];
    int j = 0;
    // Print the IEEE 754 representation from the forward stack
    printf("Binary Representation: ");
    while (stackForward != NULL)
    {
        printf("%c", peek(&stackForward));
        temp[j] = peek(&stackForward);
        j++;
        pop(&stackForward);
    }
    temp[j] = '\0';
    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);
}
```

```c
char peek(t_stack **top)
{
    if (*top != NULL)
    {
        return (*top)->string;
    }
    return '\0';
}

double abs_double(double num)
{
    return fabs(num);
}

double round(double num)
{
    return floor(num);
}

void push_binary(double Num, t_stack **stack, int *zero_count, int *index_first)
{
    int num = (int)Num;
    if (num > 1)
    {
        (*zero_count)++;
        push_binary(num / 2, stack, zero_count, index_first);
    }
    if (*index_first == 0) // Skip pushing first index
    {
        (*index_first)++;
    }
    else
    {
        char bit = (num % 2) + '0';
        push(stack, bit);
    }
}
void push_frac(double Num, t_stack **stack, int *zero_count, int *index_first)
{
    double fraction = Num - (int)Num; // Remove int part from double.
    for (int i = 0; i < SIG_BIT; i++) //  Push till all bit filled.
    {
        fraction *= 2;
        int bit = (int)fraction;
        char bit_char = bit + '0';
        (*zero_count)++;
        push(stack, bit_char);
        fraction -= bit;
        if (fraction == 0) // If no fraction
        {
            break;
        }
    }
    // Push zero to all empty bit.
    while (*zero_count < SIG_BIT)
    {
        push(stack, '0');
        (*zero_count)++;
    }
}

bool SignBitIsNegative(char number)
{
    return number == '-';
}

void pause(void)
{
    printf("Press enter to continue...");
    fflush(stdout);
    getchar();
}
```