#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);

}

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();

}