

double to IEEE 754 Converter

본 문서는 Static Linking, Dynamic Linking, Runtime Linking에 관한 구현 내용을 포함합니다.

C언어에서 IEEE 754 표준에 따라, 부동소수점 수를 2진수 표현으로 변환하는 함수를 디자인하고 링크하는 과정을 설명합니다.

■ 개발환경

OS : Ubuntu 20.04 LTS

Compiler : GCC 9.4.0

♣ 리포지토리 구조

```
└── static
    ├── main.c
    ├── double_to_ieee754.c
    ├── double_to_ieee754.h
    ├── double_to_ieee754.o
    └── libdouble.a
        └── main_static
            ├── dynamic
            │   ├── main.c
            │   ├── double_to_ieee754.c
            │   ├── double_to_ieee754.h
            │   ├── double_to_ieee754.o
            │   └── libdouble.so
            └── main_dynamic
                └── runtime
                    ├── main_runtime.c
                    ├── double_to_ieee754.c
                    ├── double_to_ieee754.h
                    ├── double_to_ieee754.o
                    └── libdouble_to_ieee754.so
                    └── main_runtime
└── README.md
```

❖ 코드 설명

double_to_ieee754.h

다음의 헤더 파일은 double_to_ieee754() 함수 프로토타입을 선언합니다.

```
#ifndef DOUBLE_TO_IEEE754_H
#define DOUBLE_TO_IEEE754_H

void double_to_ieee754(double num, char *binary);

#endif
```

double_to_ieee754.c

다음의 C 파일은 헤더 파일을 포함하고, 함수 본문을 구현합니다.

부동소수점 수를 2진수 표현으로 변환하는 함수를 디자인하였습니다.

```
#include "double_to_ieee754.h"
#include <stdint.h>

void double_to_ieee754(double num, char *binary) {
    uint64_t bits = *(uint64_t *)&num;
    for (int i = 63; i >= 0; i--) {
        binary[63 - i] = (bits & ((uint64_t)1 << i)) ? '1' : '0';
    }
    binary[64] = '\0';
}
```

main.c

라이브러리를 사용하는 메인 프로그램입니다.

double_to_ieee754() 함수를 사용하여 부동소수점 수를 2진수로 출력합니다.

```
#include <stdio.h>
#include "double_to_ieee754.h"

int main() {
    char binary[65];
    double num = 3.14;

    double_to_ieee754(num, binary);
    printf("IEEE 754 Representation: %s\n", binary);

    return 0;
}
```

main_runtime.c

Runtime Linking에서 사용되는 main 코드입니다.

dlopen과 dlsym을 사용해 런타임에 동적으로 라이브러리를 로드하고 double_to_ieee754 함수를 호출합니다.

```
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>

int main() {
    void *handle;
    void (*double_to_ieee754)(double, char *);
    char *error;

    handle = dlopen("./libdouble_to_ieee754.so", RTLD_LAZY);
    if (!handle) {
        fprintf(stderr, "%s\n", dlerror());
        exit(1);
    }

    double_to_ieee754 = dlsym(handle, "double_to_ieee754");
    if ((error = dlerror()) != NULL) {
        fprintf(stderr, "%s\n", error);
        exit(1);
    }

    char binary[65];
    double num = 3.14;
    double_to_ieee754(num, binary);
    printf("IEEE 754 Representation: %s\n", binary);

    if (fclose(handle) < 0) {
        fprintf(stderr, "%s\n", dlerror());
        exit(1);
    }
    return 0;
}
```

Static Linking

해당 과정의 실행은 static 폴더에서 이루어집니다.

1. 객체 파일 생성

double_to_ieee754.c 파일을 gcc를 통해 컴파일하여 객체 파일을 생성합니다.

```
$ gcc -Og -c double_to_ieee754.c -o double_to_ieee754.o
```

2. 아카이브 파일 생성

정적 라이브러리를 생성하는 ar -rcs 명령어를 입력합니다.

```
$ ar -rcs libdouble.a double_to_ieee754.o
```

3. 컴파일

정적 라이브러리를 사용해서 실행 파일인 main_static을 생성합니다.

```
$ gcc -static main.c -L . -ldouble -o main_static
```

4. main_static 파일을 실행합니다.

```
fos@inha:~/CSE3209/static$ gcc -Og -c double_to_ieee754.c -o double_to_ieee754.o
```

```
fos@inha:~/CSE3209/static$ ar -rcs libdouble.a double_to_ieee754.o
```

```
fos@inha:~/CSE3209/static$ gcc -static main.c -L . -ldouble -o main_static
```

```
fos@inha:~/CSE3209/static$ ls
```

```
double_to_ieee754.c double_to_ieee754.h double_to_ieee754.o libdouble.a main.c main_static
```

```
fos@inha:~/CSE3209/static$ ./main_static
```

```
IEEE 754 Representation: 010000000001001000111101011100001010001111010111000010100011111
```

Dynamic Linking

해당 과정의 실행은 dynamic 폴더에서 이루어집니다.

1. 객체 파일 생성

double_to_ieee754.c를 위치독립코드(pic)로 컴파일하여 객체 파일을 생성합니다.

```
$ gcc -fPIC -c double_to_ieee754.c -o double_to_ieee754.o
```

2. 동적 라이브러리를 생성하는 코드를 입력합니다.

```
$ gcc -shared -L . -ldouble -o libdouble.so double_to_ieee754.o
```

3. 현재 디렉토리의 동적 라이브러리를 링크하고, 메인 파일을 컴파일합니다.

```
$ gcc main.c -L . -ldouble -o main_dynamic
```

4. 현재 디렉토리를 동적 라이브러리 검색 경로로 만들어 환경변수를 설정합니다.

```
$ export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
```

5. main_dynamic 파일을 실행합니다.

```
fos@inha:~/CSE3209/dynamic$ gcc -Og -c double_to_ieee754.c -o double_to_ieee754.o
```

```
fos@inha:~/CSE3209/dynamic$ gcc -shared -L . -ldouble -o libdouble.so double_to_ieee754.o
```

```
fos@inha:~/CSE3209/dynamic$ gcc main.c -L . -ldouble -o main_dynamic
```

```
fos@inha:~/CSE3209/dynamic$ export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
```

```
fos@inha:~/CSE3209/dynamic$ ls
```

```
double_to_ieee754.c double_to_ieee754.h double_to_ieee754.o libdouble.so main.c main_dynamic
```

```
fos@inha:~/CSE3209/dynamic$ ./main_dynamic
```

```
IEEE 754 Representation: 010000000001001000111101011100001010001111010111000010100011111
```

Runtime Linking

해당 과정은 Runtime 폴더에서 이루어집니다.

1. 객체 파일 생성

double_to_ieee754.c를 위치독립코드(pic)로 컴파일하여 객체 파일을 생성합니다.

```
$ gcc -fPIC -c double_to_ieee754.c -o double_to_ieee754.o
```

2. 동적 라이브러리를 생성하는 코드를 입력합니다.

```
$ gcc -shared -L . -ldouble -o libdouble.so double_to_ieee754.o
```

3. 런타임 링크를 사용하도록 설정된 main_dynamic.c를 컴파일합니다.

```
$ gcc main.c -L . -ldouble -o main_runtime
```

4. 현재 디렉토리를 동적 라이브러리 검색 경로로 만들어 환경변수를 설정합니다.

```
$ export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
```

5. main_runtime 파일을 실행합니다.

```
fos@inha:~/CSE3209/runtime$ gcc -fPIC -c double_to_ieee754.c -o double_to_ieee754.o
```

```
fos@inha:~/CSE3209/runtime$ gcc -shared -L . -ldouble -o libdouble.so double_to_ieee754.o
```

```
fos@inha:~/CSE3209/runtime$ gcc main.c -L . -ldouble -o main_runtime
```

```
fos@inha:~/CSE3209/runtime$ export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
```

```
fos@inha:~/CSE3209/runtime$ ls
```

```
double_to_ieee754.c double_to_ieee754.h double_to_ieee754.o libdouble.so main.c main_runtime
```

```
fos@inha:~/CSE3209/runtime$ ./main_runtime
```

```
IEEE 754 Representation: 010000000001001000111101011100001010001111010111000010100011111
```