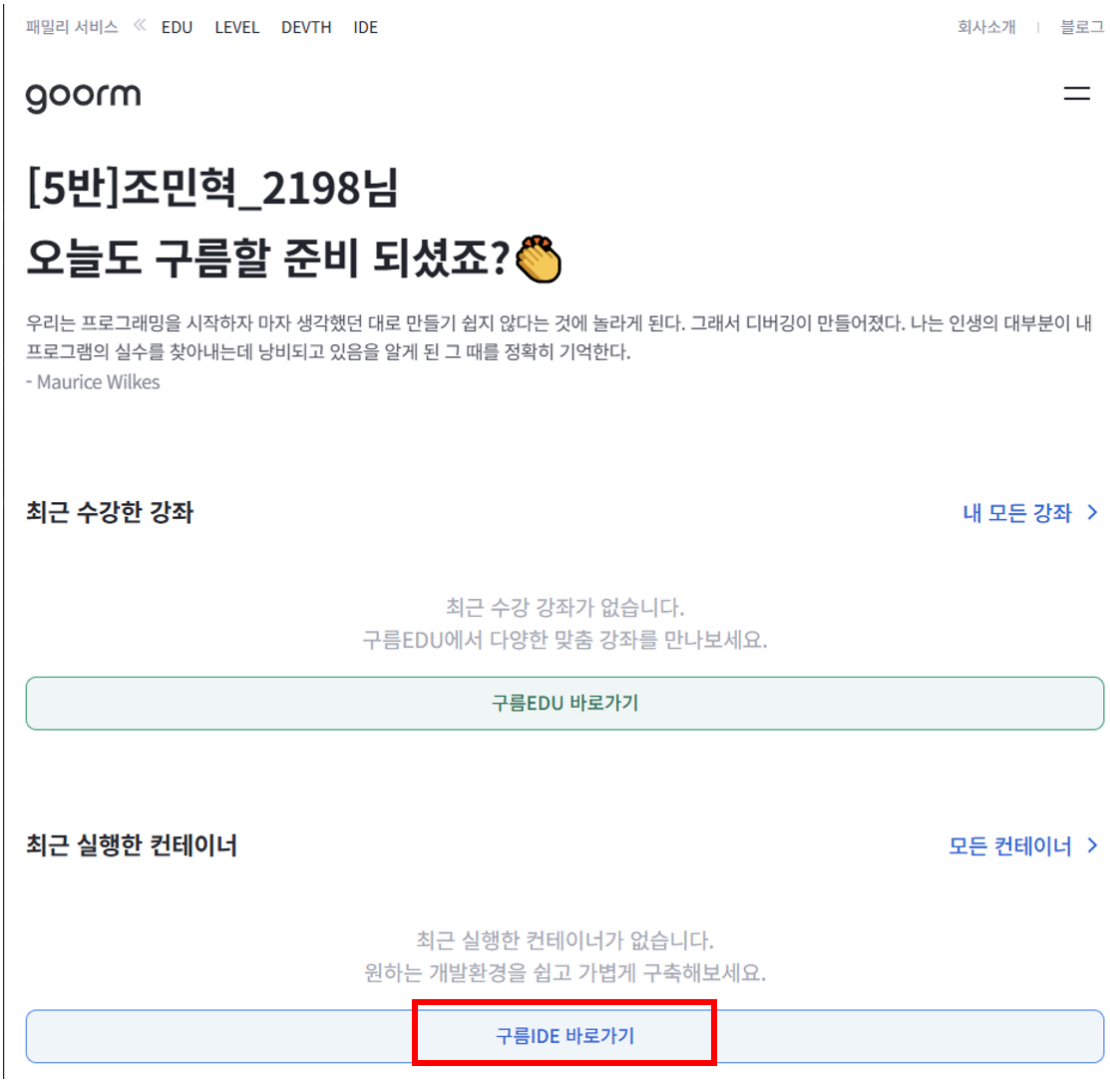
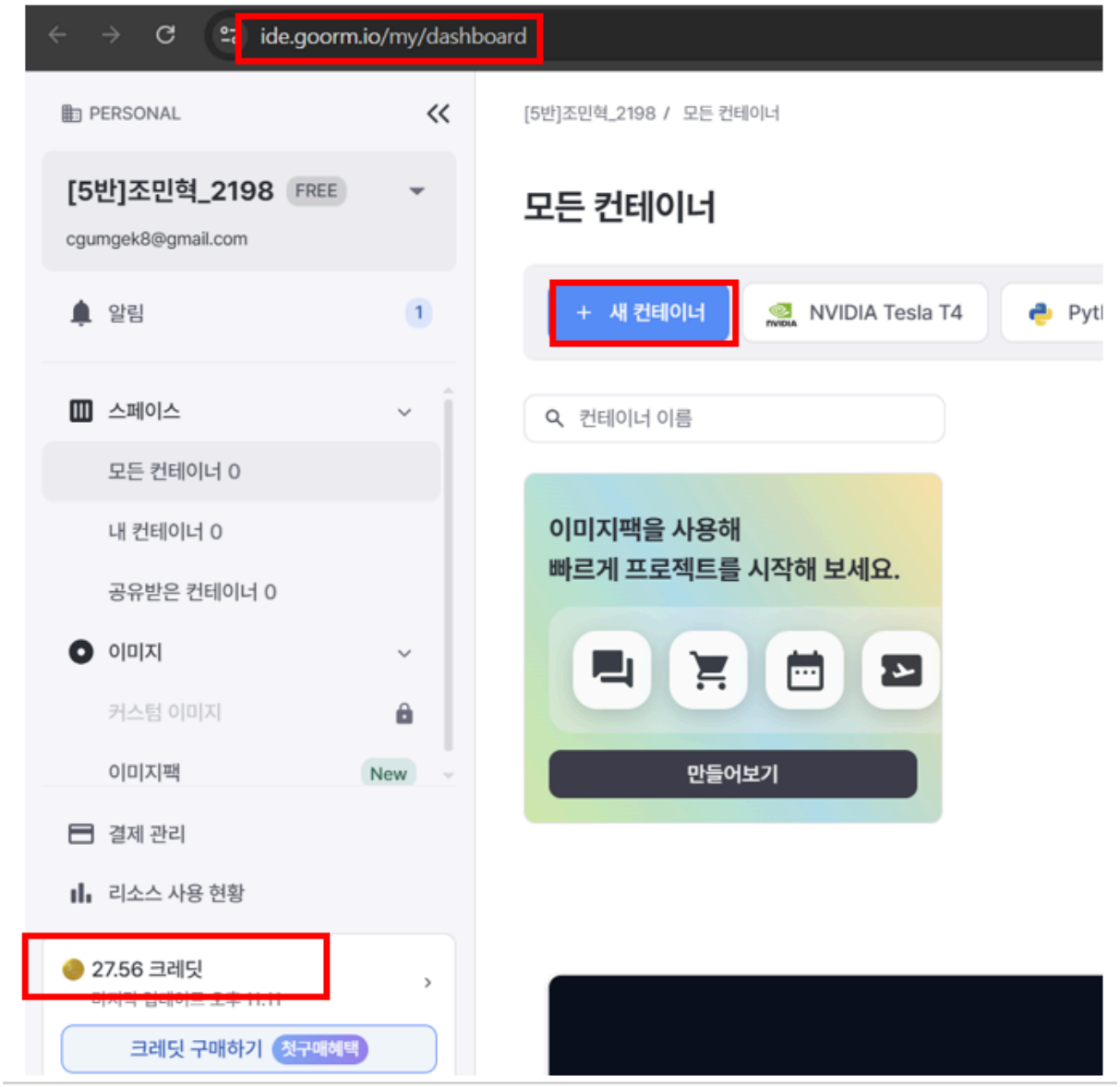
[WHS 3기 컴퓨터구조1] 조민혁 (2198) - 2025/03/15 과제 실습 보고서

1. 구름 IDE로 리눅스 개발환경 만들기

STEP 1) goorm.io/dashboard에 접속 후 구름IDE 바로가기 클릭

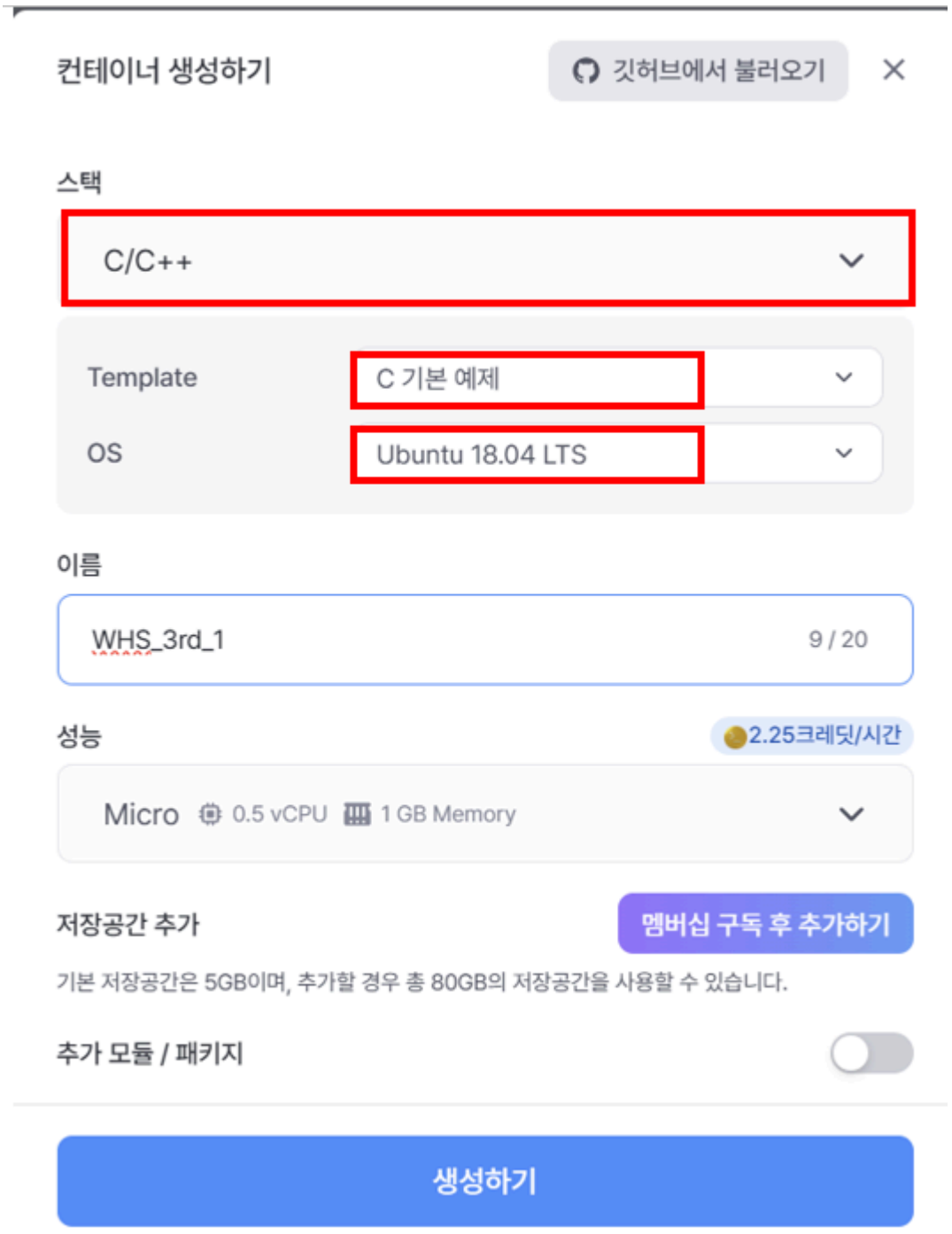


STEP 2) 아래 사진과 같이 '사용가능한 크레딧'을 확인 후 '새 컨테이너'를 클릭한다.

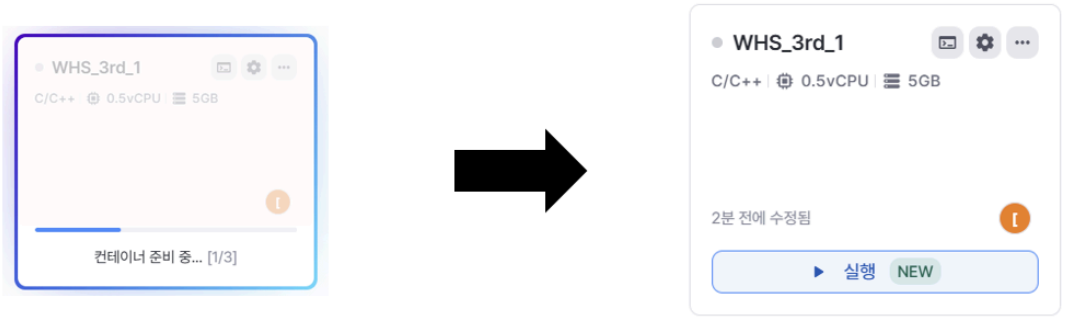


STEP 3) 아래 사진과 같이 C/C++ 스택 설정 후 Template: C 기본 예제, OS: Ubuntu 18.04 LTS로 설정

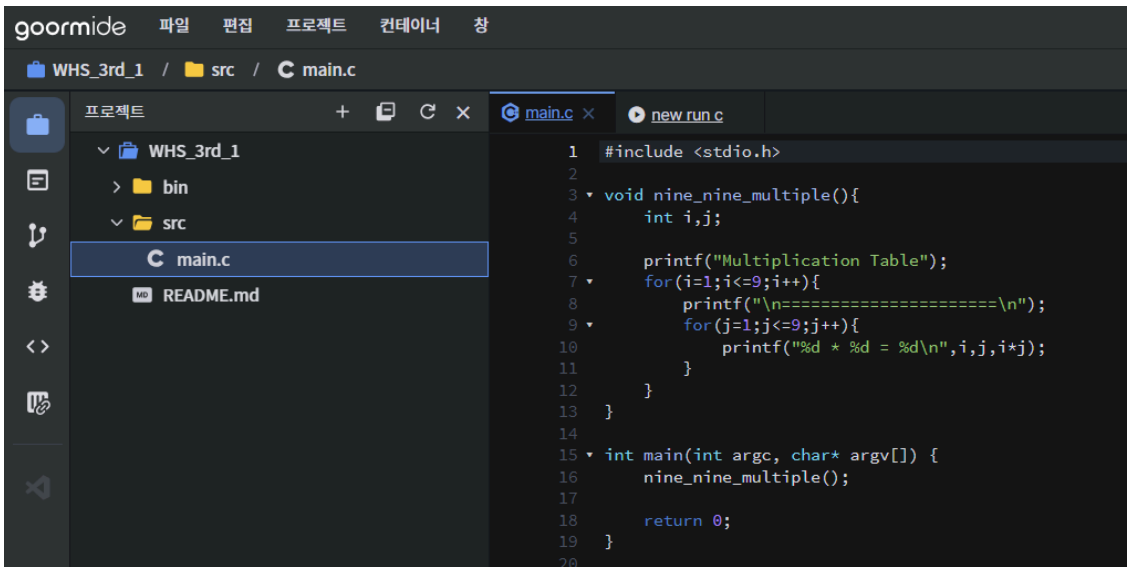
- 이후 컨테이너 이름 설정을 마친 후 '생성하기' 버튼을 클릭한다.



STEP 4) 아래와 같이 '컨테이너 준비 중...' 을 기다린 후 오른쪽과 같이 변하면 '실행' 버튼을 클릭한다.



STEP 5) '실행' 버튼을 클릭하면 아래와 같은 화면을 확인할 수 있다.



STEP 6) 최종적으로 프로그램 실행 시 아래와 같은 구구단 프로그램이 실행되는 것을 확인할 수 있다.

```
root@goorm:/workspace/WHS_3rd_1/src# gcc -o main main.c
root@goorm:/workspace/WHS_3rd_1/src# ./main
Multiplication Table
====================
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
```

2. Sizeof 연산 타이핑해보기

- sizeof 연산은 자료형의 바이트 크기를 알려주는 연산

STEP 1) WHS_Sizeof_Typing.c 코드 작성

WHS_Sizeof_Typing.c > main()

```c
#include <stdio.h>

int main(){
    printf("Size of char: %zu bytes\n", sizeof(char));
    printf("Size of short: %zu bytes\n", sizeof(short));
    printf("Size of int: %zu bytes\n", sizeof(int));
    printf("Size of long: %zu bytes\n", sizeof(long));
    printf("Size of long long: %zu bytes\n", sizeof(long long));
    printf("Size of float: %zu bytes\n", sizeof(float));
    printf("Size of double: %zu bytes\n", sizeof(double));
    printf("Size of long double: %zu bytes\n", sizeof(long double));
    printf("Size of pointer: %zu bytes\n", sizeof(void*));

    return 0;
}
```

STEP 2) 해당 코드를 컴파일후 실행 시 아래와 같은 결과를 확인할 수 있다.

- long과 long long은 64bit 시스템에서는 두 자료형 모두 8Bytes를 출력하는 것을 알 수 있다.

- pointer는 64bit 시스템에서 8Bytes인 것을 알 수 있다.

```
minhyuk@DESKTOP-9KUSLQP:~/WhiteHatSchool/Computer_Architecture$ ./WHS_Sizeof_Typing
Size of char: 1 bytes
Size of short: 2 bytes
Size of int: 4 bytes
Size of long: 8 bytes
Size of long long: 8 bytes
Size of float: 4 bytes
Size of double: 8 bytes
Size of long double: 16 bytes
Size of pointer: 8 bytes
```

3. 오버플로 예제를 언더플로로 바꿔서 해보기 - CHAR_MIN - 1 하기

STEP 1) 아래와 같이 코드를 작성한다.

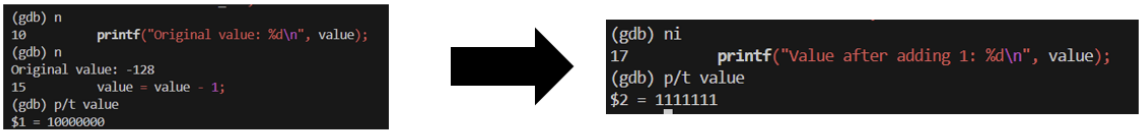
Underflow_Practice.c > main()

```c
#include <stdio.h>
#include <limits.h>

int main(){
    // CHAR_MIN == -128
    // Bit 표현 시 1000 0000
    char value = CHAR_MIN;


    printf("Original value: %d\n", value);

    // 1000 0000에 -1을 한다면
    // 1000 0000 + 1111 1111 = 1 0111 1111 (char은 1Byte 이므로) 최종적으로 0111 1111
    // 즉, 127이 된다.
    value = value - 1;

    printf("Value after adding 1: %d\n", value);

    return 0;
}
```

STEP 2) 최종적으로 아래와 같은 결과가 보여진다.

```
minhyuk@DESKTOP-9KUSLQP:~/WhiteHatSchool/Computer_Architecture$ ./Underflow_Practice
Original value: -128
Value after adding 1: 127
```

STEP 3) gdb를 통해 동적 분석 해보면 아래와 같은 비트 값이 변수에 저장된다.



4. 비트 연산 프로그램 바꿔보기

- 특정 위치의 비트를 끄는 함수 구현

- 사용자의 입력 (특정 위치 -int 값)을 받도록 수정

STEP 1) 코드를 아래와 같이 작성한다.

```c
#include <stdio.h>

unsigned char clear_bit(unsigned char value, int position){
    return ~(1 << position) & value;
}

int main(){
    int clear_position;
    printf("Clear Position: ");
    scanf("%d", &clear_position);
    unsigned char before_value = 127; // 0111 1111

    for(int i = 7; i >= 0; i--) {
        printf("%d", (before_value >> i) & 1);
    }puts("\n");

    unsigned char after_value = clear_bit(before_value, clear_position);

    for(int i = 7; i >= 0; i--) {
        printf("%d", (after_value >> i) & 1);
    }puts("\n");

    return 0;
}
```

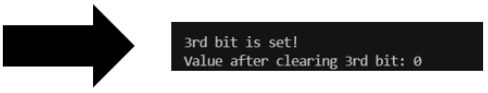
STEP 2) 해당 프로그램을 실행 후 4번째 위치를 끄도록 하면 아래와 같이 결과가 정상적으로 보여진다.

```
Clear Position: 4
01111111

01101111
```

STEP 3) 예제 프로그램에 해당 함수를 추가하여 실행해보면 아래와 같이 결과가 보여진다.

- 아래 코드는 3번째 bit를 clear한 결과이다.

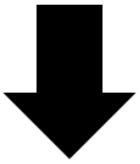
```c
#include <stdio.h>

unsigned char clear_bit(unsigned char value, int position){
    return ~(1 << position) & value;
}

int is_bit_set(unsigned char value, int position){
    return (value & (1 << position)) != 0;
}

unsigned char set_bit(unsigned char value, int position){
    return value | (1 << position);
}

int main(){
    unsigned char value = 0b00001000;

    if(is_bit_set(value, 3)) {
        printf("3rd bit is set!\n");
    } else {
        printf("3rd bit is not set!\n");
    }

    // value = set_bit(value, 2);
    // printf("Value after setting 2nd bit: %d\n", value);

    value = clear_bit(value, 3);
    printf("Value after clearing 3rd bit: %d\n", value);

    return 0;
}
```



5. C언어가 기계어가 되는 과정 직접 해보기

STEP 1) Hello World 프로그램을 전처리까지 컴파일을 진행하면 아래와 같은 결과가 나온다.

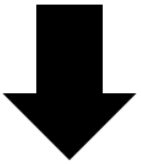
```
minhyuk@DESKTOP-9KUSLQP:~/WhiteHatSchool/Computer_Architecture$ gcc -E helloworld.c -o helloworld.i
```



```
# 0 "helloworld.c"
# 0 "<built-in>"
# 0 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 0 "<command-line>" 2
# 1 "helloworld.c"
# 1 "/usr/include/stdio.h" 1 3 4
# 27 "/usr/include/stdio.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 1 3 4
# 33 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 3 4
# 1 "/usr/include/features.h" 1 3 4
# 392 "/usr/include/features.h" 3 4
# 1 "/usr/include/features-time64.h" 1 3 4
# 20 "/usr/include/features-time64.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
# 21 "/usr/include/features-time64.h" 2 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/timesize.h" 1 3 4
# 19 "/usr/include/x86_64-linux-gnu/bits/timesize.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
# 20 "/usr/include/x86_64-linux-gnu/bits/timesize.h" 2 3 4
# 22 "/usr/include/features-time64.h" 2 3 4
# 393 "/usr/include/features.h" 2 3 4
# 486 "/usr/include/features.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 1 3 4
# 559 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 3 4
```

STEP 2) helloworld 프로그램을 어셈블리 파일까지 컴파일을 진행하면 아래와 같은 결과가 나온다.

```
minhyuk@DESKTOP-9KUSLQP:~/WhiteHatSchool/Computer_Architecture$ gcc -S helloworld.c -o helloworld.s
```

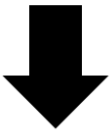


helloworld.s

```
    .file   "helloworld.c"
    .text
    .section    .rodata
.LC0:
    .string "Hello World!"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    endbr64
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    leaq    .LC0(%rip), %rax
    movq    %rax, %rdi
    call    puts@PLT
    movl    $0, %eax
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
```

STEP 3) helloworld 프로그램을 목적 파일까지 컴파일을 진행하면 아래와 같은 결과가 나온다.

```
minhyuk@DESKTOP-9KUSLQP:~/WhiteHatSchool/Computer_Architecture$ gcc -c helloworld.c -o helloworld.o
```



```
minhyuk@DESKTOP-9KUSLQP:~/WhiteHatSchool/Computer_Architecture$ readelf -a helloworld.o
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          600 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         14
  Section header string table index: 13
```

```
minhyuk@DESKTOP-9KUSLQP:~/WhiteHatSchool/Computer_Architecture$ objdump -M intel -d helloworld.o

helloworld.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <main>:
   0:   f3 0f 1e fa             endbr64
   4:   55                      push   rbp
   5:   48 89 e5                mov    rbp,rsp
   8:   48 8d 05 00 00 00 00    lea    rax,[rip+0x0]        # f <main+0xf>
   f:   48 89 c7                mov    rdi,rax
  12:   e8 00 00 00 00          call   17 <main+0x17>
  17:   b8 00 00 00 00          mov    eax,0x0
  1c:   5d                      pop    rbp
  1d:   c3                      ret
```

- 그러나 아래와 같이 아직 링킹 단계를 하지는 않았기에 프로그램이 실행되지는 않는다.

```
minhyuk@DESKTOP-9KUSLQP:~/WhiteHatSchool/Computer_Architecture$ ./helloworld.o
bash: ./helloworld.o: Permission denied
```

STEP 4) helloworld 프로그램을 링킹 까지 진행하여 컴파일을 진행하면 아래와 같이 정상적으로 프로그램이 실행된다.

```
minhyuk@DESKTOP-9KUSLQP:~/WhiteHatSchool/Computer_Architecture$ gcc -o helloworld helloworld.c
minhyuk@DESKTOP-9KUSLQP:~/WhiteHatSchool/Computer_Architecture$ ./helloworld
Hello World!
```