튜터링 11주차

(TUTOR: 성열암)

응용컴퓨터 프로그래밍

TUTORING — https://github.com/developersung13/cbnu-tutoring

전처리 및 다중 소스 파일 상호 간 연결 방법 익히기

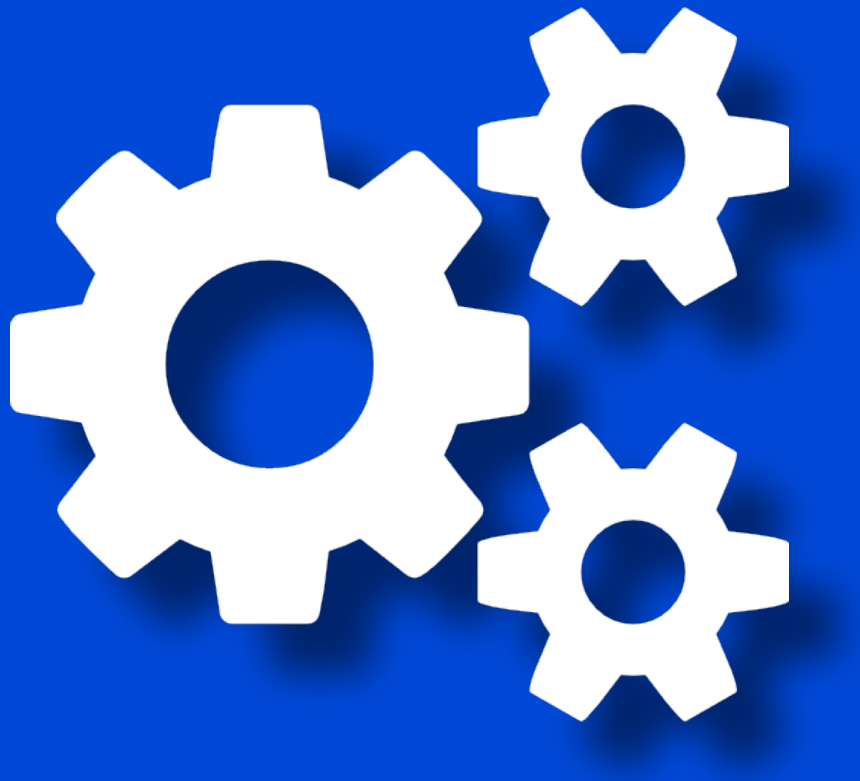
CONTENTS
INDEX

01 전처리기

02 퀴즈

03 질의응답

전처리기

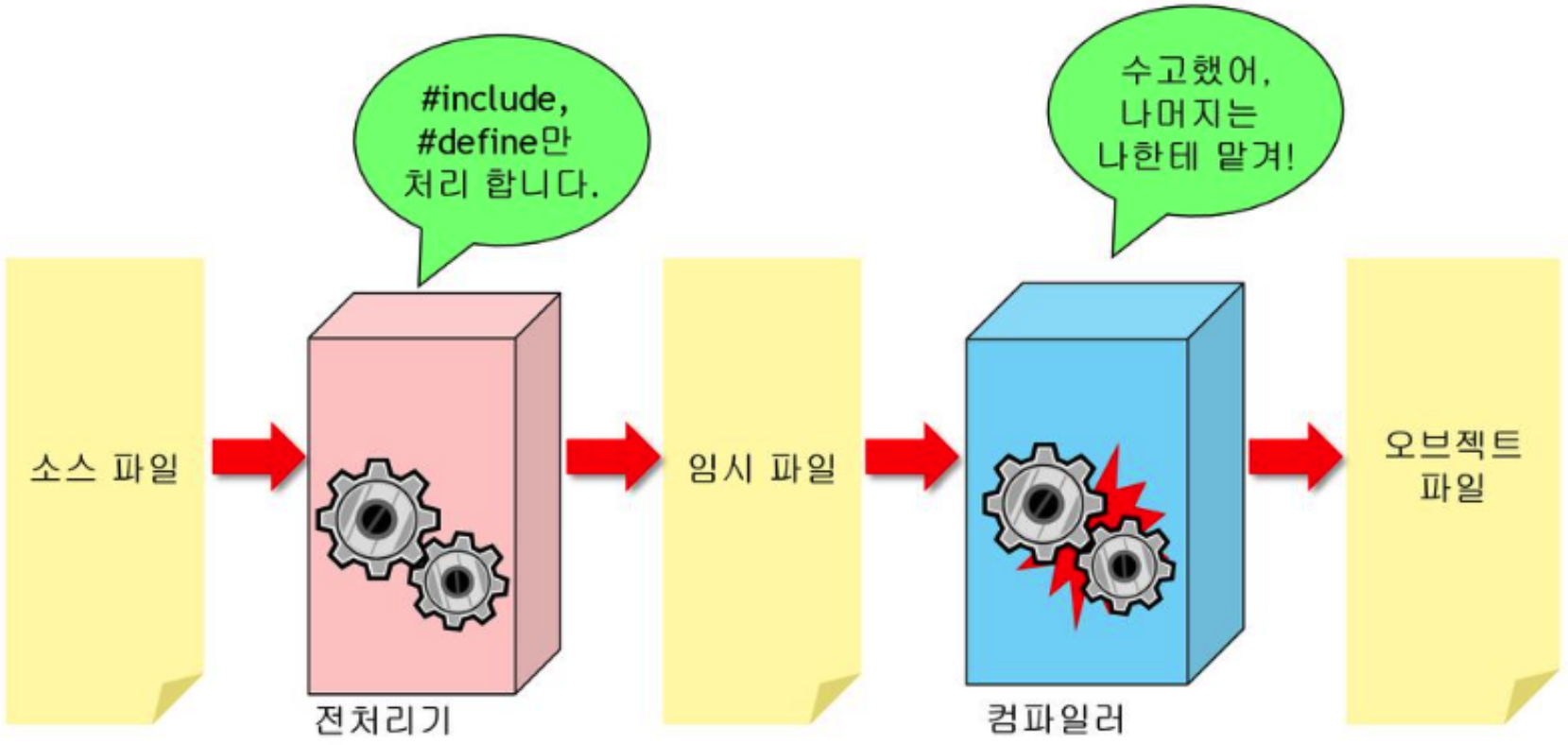


코드를 컴파일하기에 앞서서 소스 파일을 처리하는
컴파일러의 한 부분이다.

01

01 전처리기 (1/17)

❑ 전처리기의 개념



전처리기는 소스 파일을 처리하여
수정된 소스 파일을 생산한다.

❑ 전처리기의 사용

전처리기는 몇 가지의 전처리기 지시자들을
처리한다. 이들 지시자들은 #기호로 시작한다.

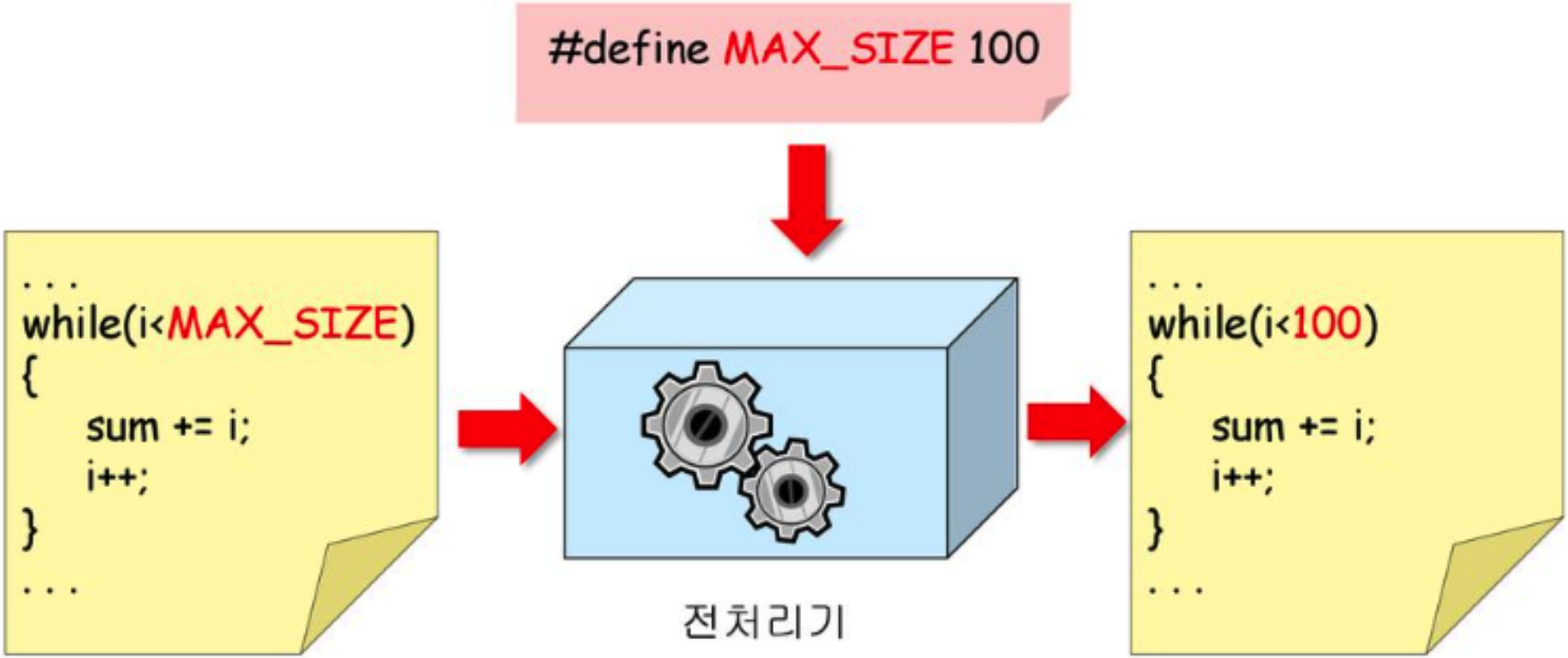
지시어(1)	의미	지시어(2)	의미
#define	매크로 정의	#endif	조건 처리 문장 종료
#include	파일 포함	#ifdef	매크로가 정의되어 있는 경우
#undef	매크로 정의 해제	#ifndef	매크로가 정의되어 있지 않은 경우
#if	조건이 참일 경우	#line	행번호 출력
#else	조건이 거짓일 경우	#pragma	시스템에 따라 의미가 다름

❑ 단순 매크로(1)

#define 문을 이용하여 숫자 상수를 기호
상수로 만든 것을 단순 매크로(macro)라고 한다.

```
1   #include <stdio.h>
2
3   // 기호상수 MAX_SIZE를 100으로 정의한다.
4   #define MAX_SIZE 100
```

❑ 단순 매크로(2)



단순 매크로 사용의 장점은 프로그램의
가독성이 높아지고, 상수의 변경이 용이하다.

❑ 단순 매크로 예제

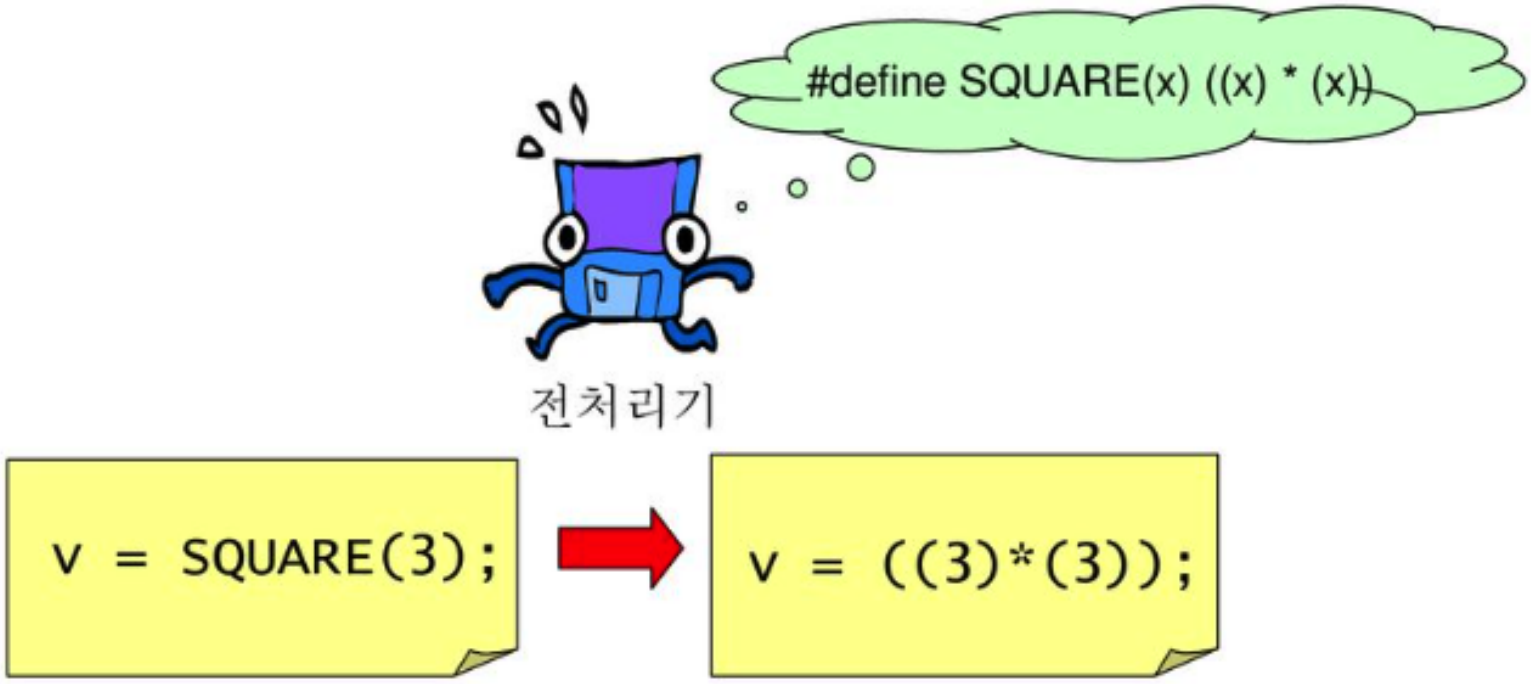
```c
#include <stdio.h>

#define AND &&
#define OR ||
#define NOT !
#define IS ==
#define ISNOT !=
#define KEY 300

int main() {
  int k = 0;
  int numArr[] = { 100, 200, 300, 400, 500 };
  int arrSize = sizeof(numArr) / sizeof(numArr[0]);

  while (k < arrSize AND numArr[k] ISNOT KEY) k++;
  if (k IS arrSize) printf("KEY값에 해당하는 데이터를 조회하지 못 하였습니다.\n");
  else printf("배열 내 %d값의 위치: %d\n", KEY, k);
  return 0;
}
```

```
> make -s
> ./main
배열 내 300값의 위치: 2
>
```

❑ 함수 매크로(1)

함수 매크로(function-like macro)란
매크로가 함수처럼 매개 변수를 가지는 것이다.

(예) #define SQUARE(x) ((x) * (x))



❑ 함수 매크로(2)

```c
#include <stdio.h>
#define SQUARE(x) x * x

int main() {
    int a = 1, b = 2;
    int num = SQUARE(a + b);
    printf("%d\n", num);
    return 0;
}
```

What's wrong?

❑ 함수 매크로의 장점

함수 매크로에서는 매개 변수의 자료형을 써주지 않는다.
따라서 어떠한 자료형에 대해서도 적용이 가능하다. (Overloading)

```
1  #include <stdio.h>
2
3  #define SUM(x, y) ((x)+(y))
4  #define AVG(x, y, z) (((x)+(y)+(z)) / 3)
5  #define MAX(x, y) ((x) > (y)) ? (x) : (y)
6  #define MIN(x, y) ((x) < (y)) ? (x) : (y)
```

❑ # 연산자

#은 문자열 변환 연산자(Stringiing Operator)라고 불린다.
매크로 정의에서 매개 변수 앞에 #가 위치하면 매크로 호출에
의하여 전달되는 실제 인수는 큰따옴표로 감싸지고 문자열로 변환된다.

```c
#include <stdio.h>

#define PRINT(temp) printf(#temp"=%d\n", temp);

int main() {
    int x = 5;
    PRINT(x);
    return 0;
}
```

```
> make -s
> ./main
x=5
>
```

❑ 내장 매크로

내장 매크로란 컴파일러가 프로그래머들이 유용하게
사용하도록 제공하는 몇 개의 미리 정의된 매크로이다.

내장 매크로	설명
__DATE__	이 매크로를 만나면 소스가 컴파일된 날짜(월 일 년)로 치환된다.
__TIME__	이 매크로를 만나면 소스가 컴파일된 시간(시:분:초)으로 치환된다.
__LINE__	이 매크로를 만나면 소스 파일에서의 현재의 라인 번호로 치환된다.
__FILE__	이 매크로를 만나면 소스 파일 이름으로 치환된다.

❑ 내장 매크로 예제

```c
#include <stdio.h>

int main() {
    printf("Current Date: %s\n", __DATE__);
    printf("오류 발생 파일: %s, 라인 번호: %d\n", __FILE__, __LINE__);
    return 0;
}
```

프로그래머가 위 매크로를 사용하면 전처리기에
의하여 이들 매크로는 매크로의 정의된 코드로 치환된다.

❑ #ifdef, #else, #endif

어떤 조건이 만족되었을 경우에만
컴파일하는 조건부 컴파일 지시한다.

```
#ifdef 매크로
문장1              // 매크로가 정의되었을 경우
...
#else
문장2              // 매크로가 정의되지 않았을 경우
...
#endif
```

01 전처리기 (13/17)

❑ #ifdef, #else, #endif 예제

```
1  #include <stdio.h>
2
3  #define DEBUG
4
5  int main() {
6      int x = 5, y = 10;
7
8      printf("x = %d, y = %d\n", x, y);
9  #ifdef DEBUG
10     x++;
11     y++;
12 #else
13     x--;
14     y--;
15 #endif
16     printf("x = %d, y = %d\n", x, y);
17     return 0;
18 }
```

Result?

❑ ifndef, undef

```
1  #ifndef LIMIT
2  #define LIMIT 1000
3  #endif
```

#ifndef는 #ifdef의 반대의 의미로 사용

```
1  #define SIZE 1000
2  ...
3  #undef SIZE
4  #define SIZE 2000
```

#undef은 매크로의 정의를 취소한다.

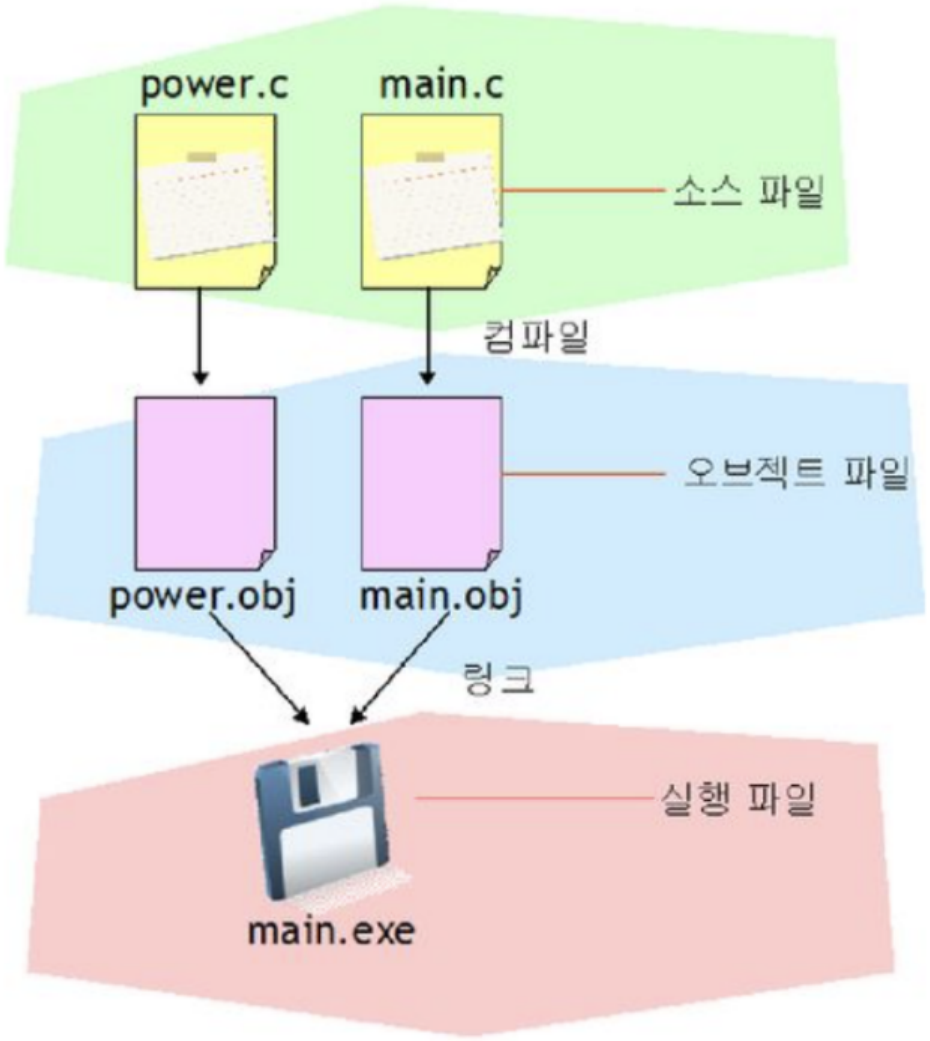
❑ #if, #else, #endif

```c
#include <stdio.h>

#define DEBUG 1

int main() {
#if DEBUG == 1
  printf("Entered(1)");
#elif DEBUG == 2
  printf("Entered(2)");
#else
  printf("Entered(3)");
#endif
  return 0;
}
```

```
> make -s
> ./main
Entered(1)
>
```

#if 다음에 있는 기호를 검사하여 기호가 참으로 계산되면 #if와 #endif 사이에 있는 모든 코드를 컴파일한다. 조건은 상수 수식이어야 하고 관계 연산자나 논리 연산자는 사용할 수 있다.

❑ 다중 소스 파일


power.c
main.c
소스 파일
컴파일
power.obj
main.obj
오브젝트 파일
링크
main.exe
실행 파일

01 전처리기 (17/17)

❑ 다중 소스 파일 예제

main.c

```c
#include "sum.h"
#include <stdio.h>

int main() {
    int x = 1, y = 2;
    printf("x + y = %d\n", sum(x, y));
    return 0;
}
```

sum.h

```c
#ifndef SUM_H
#define SUM_H
int sum(int x, int y);
#endif
```

서로 관련된 코드만을 모아서
하나의 소스 파일로 만들 수 있다.

sum.c

```c
#include "sum.h"

int sum(int x, int y) { return x + y; }
```

```
> make -s
> ./main
x + y = 3
>
```

QUIZ

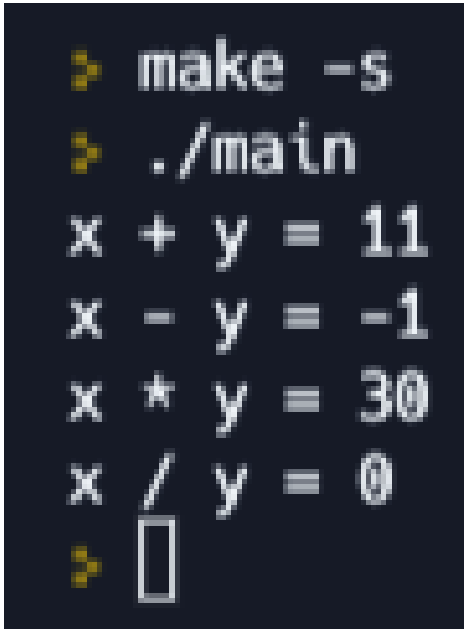
퀴즈

간단한 문제를 통하여 이번 튜터링 시간에
익힌 내용을 실습을 통해 확인하는 시간입니다.

02

02 퀴즈 (1/1)

아래와 같은 사칙연산 결과가 나타나도록
기존 main.c 소스코드 이외에 calc.c와 calc.h
파일을 추가로 생성하여 밑의 결과를 만들 수
있도록 코드를 작성하시오.



질의응답

금일 튜터링을 진행하며 이해가 어려운 부분이 있었거나,
교과목과 관련하여 궁금한 내용을 질문하고 답변드리는
시간입니다.

03

THANKYOU

TUTORING

https://github.com/developersung13/cbnu-tutoring