## 매크로 사용하기

C 언어에서는 컴파일러에 특정 작업을 지시하는 전처리기(preprocessor)를 제공한다.

보통 전처리기는 반복되는 값이나 작업을 미리 정의할 때 사용하며 컴파일 옵션 설정이나 조건부 컴파일도 가능하다.

## 매크로 정의

#define으로 정의한 매크로는 다른 이름으로 다시 정의할 수 있습니다.

#define ARRAY\_SIZE 10 // 10을 ARRAY\_SIZE로 정의

#define DEFAULT\_ARRAY\_SIZE ARRAY\_SIZE // ARRAY\_SIZE를 DEFAULT\_ARRAY\_SIZE로 정의

이렇게 정의하면 결과적으로 DEFAULT\_ARRAY\_SIZE는 10입니다.

## 함수 형태의 매크로 정의

#define은 함수 모양의 매크로도 정의할 수 있다.

* **#define 매크로이름(x) 함수(x)**
* **#define 매크로이름(x) 코드조합**

|  |
| --- |
| #include <stdio.h>  **#define PRINT\_NUM(x) printf("%d\n", x)** // printf("%d\n", x)를 PRINT\_NUM(x)로 정의  int main()  {  PRINT\_NUM(10); // 10: printf("%d\n", 10)  PRINT\_NUM(20); // 20: printf("%d\n", 20)  return 0;  }  **실행 결과**  10  20 |

**참고 |** **함수를 사용하지 못하도록 만들려면?**

#define을 사용하면 함수가 아무 동작도 하지 않도록 만들 수 있습니다. 다음과 같이 함수 이름으로 된 빈 매크로를 지정하면 이후에 해당 이름으로 함수를 호출해도 사용할 수 없게 됩니다.

#define printf // printf를 빈 매크로로 정의

printf("Hello, world!"); // 아무것도 출력되지 않음

## 여러줄 묶어서 매크로 정의

* #define 매크로이름 코드1 \

코드2 \

코드3

|  |
| --- |
| **//macro\_function\_multiline.c**  #include <stdio.h>  // printf 세 줄을 PRINT\_NUM3으로 정의  #define PRINT\_NUM3(x) printf("%d\n", x); \  printf("%d\n", x + 1); \  printf("%d\n", x + 2);  int main()  {  PRINT\_NUM3(10); // 10  // 11  // 12  PRINT\_NUM3(20); // 20  // 21  // 22  return 0;  }  **실행 결과**  10  11  12  20  21  22 |

여러줄로 된 매크로 사용시, 조건문/반복문 사용에 주의!

|  |
| --- |
| **macro\_function\_multiline\_if.c**  #include <stdio.h>  #define PRINT\_NUM3(x) printf("%d\n", x); \  printf("%d\n", x + 1); \  printf("%d\n", x + 2);  int main()  {  int num1 = 1;  **if (num1 == 2)**  PRINT\_NUM3(10); // 11  // 12  return 0;  }  **실행 결과**  11  12 |

소스 코드를 실행했을 때 num1은 1이므로 다음 if 조건문은 실행되지 않아야 하지만

11, 12가 출력되었다.

전처리를 거치면서 최초 첫번째 printf문만 영향을 주기때문에 아래 두줄은 영향을 받지 않아 출력되었다.

중괄호를 통해 이러한 문제를 회피할 수 있다.

## SWAP매크로 정의하기

일반적인 swap함수는 다음과 같다.

|  |
| --- |
| void swap(int \*a, int \*b)  {  int temp;  temp = \*a;  \*a = \*b;  \*b = temp;  } |

#define로 매크로를 만들 때 do while을 사용하면 모든 자료형을 만족하면서 두 변수의 값을 바꿀 수 있다.

|  |
| --- |
| **//swap\_macro.c**  #include <stdio.h>  // 매크로 안에서 변수를 선언할 수 있도록 do while 사용  // a와 b의 값을 서로 바꿈  #define SWAP(a, b, type) do { \  type temp; \  temp = a; \  a = b; \  b = temp; \  } while (0)  int main()  {  int num1 = 10;  int num2 = 20;  SWAP(num1, num2, int); // 값을 바꿀 자료형으로 int를 지정  printf("%d %d\n", num1, num2); // 20 10: 두 변수의 값이 바뀜  float num3 = 1.5f;  float num4 = 3.8f;  SWAP(num3, num4, float); // 값을 바꿀 자료형으로 float를 지정  printf("%f %f\n", num3, num4); // 3.800000 1.500000: 두 변수의 값이 바뀜  return 0;  }  **실행 결과**  20 10  3.800000 1.500000  #define으로 SWAP을 정의할 때 바꿀 변수가 들어갈 a와 b 그리고 자료형이 들어갈 type을 지정해준다. 함수에서는 자료형을 매개변수에 그대로 전달할 수 없지만 매크로는 전달하는 것이 아니라 지정한 코드 자체가 바뀌게 되므로 자료형도 지정할 수 있다.  // 매크로 안에서 변수를 선언할 수 있도록 do while 사용  // a와 b의 값을 서로 바꿈  #define SWAP(a, b, type) do { \  type temp; \  temp = a; \  a = b; \  b = temp; \  **} while (0)**  다음은 SWAP(num1, num2, int);와 같이 int형 변수 num1, num2와 자료형 int를 지정한 뒤 전처리기를 거쳤을 때 바뀐 코드를 나타낸다. 즉, type 부분이 자료형 int로 바뀌면서 temp 변수를 선언하게 된다. a와 b도 지정된 변수로 바뀌므로 자료형은 따지지 않아도 된다.  do {  // ↓ type 부분이 자료형으로 바뀌므로 지정한 자료형으로 변수를 선언할 수 있음  int temp;  temp = num1;  num1 = num2;  num2 = temp;  } while (0); |

## 매크로와 연산자 우선순위

매크로를 정의할 때는 연산자 우선순위를 주의해야 합니다. 먼저 두 수를 곱하는 매크로와 더하는 매크로를 정의해보겠습니다.

**// macro\_operator\_precedence\_wrong.c**

#include <stdio.h>

// a와 b를 곱함

#define MUL(a, b) a \* b

// a와 b를 더함

#define ADD(a, b) a + b

int main()

{

printf("%d\n", MUL(10, 20)); // 200: 10 \* 20

printf("%d\n", MUL(1 + 2, 3 + 4)); // 11: 1 + 6 + 4, 2 \* 3이 먼저 계산됨

printf("%d\n", ADD(1, 2)); // 3: 1 + 2

printf("%d\n", ADD(1, 2) \* 3); // 7: 1 + 6, 2 \* 3이 먼저 계산됨

return 0;

}

**실행 결과**

200

11

3

7

MUL은 a와 b를 곱하고, ADD는 a와 b를 더하도록 정의되어 있습니다.

// a와 b를 곱함

#define MUL(a, b) a \* b

// a와 b를 더함

#define ADD(a, b) a + b

이제 매크로를 사용해보면 MUL(10, 20)은 200이 잘 나오지만 MUL(1 + 2, 3 + 4)은 3 \* 7의 결과인 21이 나오지 않고 11이 나온다. 마찬가지로 ADD(1, 2)는 결과가 잘 나오지만 ADD(1, 2) \* 3은 9가 나와야 하는데 7이 나왔다.

printf("%d\n", MUL(10, 20)); // 200: 10 \* 20

printf("%d\n", MUL(1 + **2, 3** + 4)); // 11: 1 + 6 + 4, 2 \* 3이 먼저 계산됨

printf("%d\n", ADD(1, 2)); // 3: 1 + 2

printf("%d\n", ADD(1, **2) \* 3**); // 7: 1 + 6, 2 \* 3이 먼저 계산됨

왜냐하면 MUL이 전처리기를 거쳤을 때 다음과 같은 모양이 되므로 연산자 우선순위에 따라 2 \* 3이 먼저 계산됨. 즉, 1 + 6 + 4라서 11이 나온다.

/\*

a b

↓ ↓ \*/

1 + 2 \* 3 + 4

// ↑ 먼저 계산됨

마찬가지로 ADD도 전처리기를 거치면 숫자와 연산자가 그냥 나열된다. 여기서도 연산자 우선순위에 따라 덧셈보다 곱셈이 먼저 계산되므로 2 \* 3이 먼저 계산됩니다. 1 + 6이라서 7이 된다.

/\*

a b

↓ ↓ \*/

1 + 2 \* 3

// ↑ 먼저 계산됨

이렇게 연산자 우선순위에 따라서 의도치 않게 계산 결과가 바뀌는 문제를 해결하려면 매크로의 인수와 결과를 모두 ( ) 괄호로 묶어주면 된다.

**//macro\_operator\_precedence.c**

#include <stdio.h>

// a와 b, 결과를 모두 괄호로 묶어줌

**#define MUL(a, b) ((a) \* (b))**

// a와 b, 결과를 모두 괄호로 묶어줌

**#define ADD(a, b) ((a) + (b))**

int main()

{

printf("%d\n", MUL(10, 20)); // 200: 10 \* 20

printf("%d\n", MUL(1 + 2, 3 + 4)); // 21: 3 \* 7

printf("%d\n", ADD(1, 2)); // 3: 1 + 2

printf("%d\n", ADD(1, 2) \* 3); // 9: 3 \* 3

return 0;

}

**실행 결과**

200

21

3

9