C Programming - Day 6

2017.09.14.

JunGu Kang
Dept. of Cyber Security

아주대학교 Whois

전처리기

Preprocessor

컴파일 이전에 소스코드를 전처리한다.

소스코드 포함

```
#include <standard_header_name.h>
#include “user_defined_header_name.h”;
```

소스코드 포함

소스코드를 그대로 읽어서 붙여넣는다.

매크로 상수

```
#define VALUE 1
```

매크로 함수

```
#define VALUE 10

main() {
    printf(“%d”, VALUE);
}
```

매크로 함수

```
main() {
    printf(“%d”, 10);
}
```

매크로 함수

```
#define ADD(x, y) x+y
```

매크로 함수

```
#define ADD(x, y) x+y

main() {
    printf(“%d”, MUL(2, 3)*MUL(3, 4));  // 5*7?
}
```

매크로 함수

```
main() {
    printf(“%d”, 2+3*3+4);   // 전처리된 코드는 2+9+4
                             // 의도한 것은 (2+3)*(3+4)
}
```

매크로 함수

모조리 괄호를 쳐줘야 한다.

매크로 함수

```
#define ADD(x, y) ((x)+(y))

main() {
    printf(“%d”, MUL(2, 3)*MUL(3, 4));
}
```

매크로 함수

```
main() {
    printf(“%d”, ((2)+(3))*((3)+(4)));  // 원하는 대로 처리되었다.
}
```

매크로 함수

장점보다 단점이 많으니 쓰지 않는게 좋다.

~~사실 쓰기도 더럽다.~~

조건부 컴파일

```
#define OPTION 1

main() {
#if OPTION == 1
    printf(“OPTION is 1\n”);
#elif OPTION == 2
    printf(“OPTION is 2\n”);
#else
    printf(“OPTION is Not 1 or 2\n”);
#endif
}
```

조건부 컴파일

```
main() {
    printf(“OPTION is 1\n”);
}
```

조건부 컴파일

```
#define OPTION 2

main() {
#if OPTION == 1
    printf(“OPTION is 1\n”);
#elif OPTION == 2
    printf(“OPTION is 2\n”);
#else
    printf(“OPTION is Not 1 or 2\n”);
#endif
}
```

조건부 컴파일

```
main() {
    printf(“OPTION is 2\n”);
}
```

조건부 컴파일

```
#define OPTION 3

main() {
#if OPTION == 1
    printf(“OPTION is 1\n”);
#elif OPTION == 2
    printf(“OPTION is 2\n”);
#else
    printf(“OPTION is Not 1 or 2\n”);
#endif
}
```

조건부 컴파일

```
main() {
    printf(“OPTION is Not 1 or 2\n”);
}
```

조건부 컴파일

```
#define VALUE 1

main() {
#ifdef VALUE
    printf(“%d\n”, VALUE);
#ifndef VALUE
    printf(“VALUE is not defined.\n”);
#endif
}
```

조건부 컴파일

```
main() {
    printf(“%d\n”, 1);
}
```

조건부 컴파일

```
main() {
#ifdef VALUE
    printf(“%d\n”, VALUE);
#ifndef VALUE
    printf(“VALUE is not defined.\n”);
#endif
}
```

조건부 컴파일

```
main() {
    printf(“VALUE is not defined.\n”);
}
```

조건부 컴파일

```
#define STRING(A) “String is ” A

main() {
    printf(“%s\n”, STRING(A));
}
```

조건부 컴파일

```
#define STRING(A) “String is ” A

main() {
    printf(“%s\n”, STRING(“Hello!”));
}
```

조건부 컴파일

```
#define STRING(A) “String is ” A

main() {
    printf(“%s\n”, “String is ” A);  // 에러
}
```

조건부 컴파일

```
#define STRING(A) “String is ” #A

main() {
    printf(“%s\n”, “String is ” “Hello”);
}
```

조건부 컴파일

```
#define STRING(A) “String is ” #A

main() {
    printf(“%s\n”, “String is Hello”);
}
```

조건부 컴파일

```
#define CONCAT(A, B) AB

main() {
    printf(“%s\n”, CONCAT(12, 34));
}
```

조건부 컴파일

```
#define CONCAT(A, B) AB

main() {
    printf(“%s\n”, AB);
}
```

조건부 컴파일

```
#define CONCAT(A, B) A ## B

main() {
    printf(“%s\n”, CONCAT(12, 34));
}
```

조건부 컴파일

```
#define CONCAT(A, B) A ## B

main() {
    printf(“%s\n”, 1234);
}
```

여러 소스코드로 구성된 프로그램

External Variable

```
// main.c
extern int num;

main() {
    printf(“%d\n”, num);
}

// var.c
int num = 10;
```

External Function

```
// main.c
extern void say_hello(void);

main() {
    say_hello();
}

// func.c
void say_hello(void) {
    printf(“Hello!\n”);
}
```

External Function

```
// main.c
void say_hello(void);  // 함수에는 extern 키워드를 생략할 수 있다.

main() {
    say_hello();
}

// func.c
void say_hello(void) {
    printf(“Hello!\n”);
}
```

Static Variable

```
// main.c
extern int num;  // static 변수에 접근할 수 없다.

main() {
    printf(“%d\n”, num);
}

// var.c
static int num = 10;  // var.c 내부에서만 접근할 수 있다.
```

Static Function

```
// main.c
extern void say_hello(void);  // static 함수에 접근할 수 없다.

main() {
    say_hello();
}

// func.c
static void say_hello(void) {  // func.c 내부에서만 접근할 수 있다.
    printf("Hello!\n");
}
```

여러 파일을 컴파일

```
$ gcc -c source1.c
$ gcc -c source2.c
$ gcc -c source3.c
$ ls
source1.c source2.c source3.c
$ gcc -o output source1.o source2.o source3.o
```

여러 파일을 컴파일

```
$ gcc -o output source1.c source2.c source3.c
```

가변 인자 함수

가변 인자 함수

다양한 갯수의 인자를 전달받는 함수.

가변 인자 함수

```
void print_numbers(int n, …) {
    va_list numbers;
    for(int i = 0; i < n; i++) printf(“%d \n”, numbers[i]);
    printf(“\n”);
}

main() {
    print_numbers(2, 1, 2);  // 갯수, 인자1, 인자2
    print_numbers(4, 3, 4, 5, 6;  // 갯수, 인자1, 인자2, 인자3, 인자4
}
```

C언어 표준 비교

Traditional C → ANSI C

함수는 호출하기 전에 선언되어야 한다.

Traditional C → ANSI C

```
main() {
    a_function();  // a_function이 위에 선언되지 않았으므로 오류
}

a_function() {
    // do_something
}
```

Traditional C → ANSI C

```
a_function() {
    // do_something
}

main() {
    a_function();
}
```

Traditional C → ANSI C

```
a_function();  // 또는 이렇게 선언해야 한다.

main() {
    a_function();
}

a_function() {
    // do_something
}
```

Traditional C → ANSI C

열거형과 void형의 추가

Traditional C → ANSI C

표준 라이브러리 및 함수의 변경

Traditional C → ANSI C

새로운 전처리기 명령

`#elif` `#error` `#pragma`

Traditional C → ANSI C

구조체를 함수의 인자로 전달하고, 반환할 수 있게 됨.

Traditional C → ANSI C

```
typedef struct {
    // some variables
} Struct;

Struct a_function(Struct);

main() {
    Struct strct1, strct2;
    strct2 = a_function(strct1);
}
```

Traditional C → ANSI C

새로운 자료형 수식자

```
const   signed   volatile
```

Traditional C → ANSI C

float형이 항상 double로 캐스팅되지는 않음.

Traditional C → ANSI C

```
main() {
    float a = 7.42;
    float b = 3.14;
    a + b;  // (double) a + (double) b
}
```

Traditional C → ANSI C

```
main() {
    float a = 7.42;
    float b = 3.14;
    a + b;  // (float) a + (float) b
}
```

Traditional C → ANSI C

변수나 함수 등 식별자의 유효길이가 31글자로 확장

Traditional C → ANSI C

```
int abcdefghi1;
int abcdefghi2;

// 두 변수는 같은 변수(유효 식별 길이가 9이므로)
```

Traditional C → ANSI C

```
int abcdefghi1;
int abcdefghi2;

// ANSI C에서는 두 변수를 다른 변수로 인식
```

ANSI C → C95

연산자를 대체할 수 있는 매크로(iso646.h)

ANSI C → C95

연산자	매크로
&&	and
&=	and_eq
&	bitand
| |	or
|=	or_eq
|	bitor
!	not
!=	not_eq
~	compl
^	xor
^=	xor_eq

C95 → C99

인라인 함수

C95 → C99

```
int abcdefghi1;
int abcdefghi2;

// ANSI C에서는 두 변수를 다른 변수로 인식
```

C95 → C99

변수의 선언을 꼭 맨 위에 하지 않아도 됨.

C95 → C99

새로운 자료형

```
long long   _Bool   _Complex   _Imaginary
```

C95 → C99

자료형과 관계없이 연산을 해주는 `tgmath.h`

C95 → C99

복소수 연산을 지원하는 `complex.h`

C95 → C99

가변 길이 배열

C95 → C99

가변 길이 배열
배열의 길이를 변수로 선언할 수 있다.

C95 → C99

// 로 시작하는 주석

restrict 키워드

C99 → C11

자료형에 따른 함수 실행을 지원하는 `_Generic` 키워드

C99 → C11

```
#define func(x) _Generic((x), long double: func_ld, \
                              double: func_d, \
                              float: func_f, \
                              long: func_l, \
                              int: func_i, \
                              default: func)(x)
```

C99 → C11

크로스플랫폼 멀티 스레딩 API `threading.h`

C99 → C11

유니코드 지원 개선

C99 → C11

취약한 함수 gets()의 제거,
안전한 함수 gets_s()의 추가

C Programming - Day 6

2017.09.14.

JunGu Kang
Dept. of Cyber Security

아주대학교 Whois