14\_전처리기와 분할 컴파일

학습목표

전처리기 기능들을 이해할 수 있다.

단순 매크로를 사용할 수 있다.

함수 매크로를 사용할 수 있다.

복잡한 프로그램을 여러 개의 소스 파일로 나누어서 작성할 수 있다.

1. 전처러기란?

전처리기(preprocessor)는 본격적으로 컴파일하기에 앞서서 소스 파일을 처리하는 컴파일러의 한 부분이다. 전처리기는 보통 컴파일러에 포함되어 있고 자동으로 실행되며 컴파일러의 하나의 요소로 취급된다.

전처리기 첫 문자가 #인 문장만 처리한다.

전처리기에 사용되는 지시자들

#define 매크로 정의

#include 파일 포함

#undef 매크로 정의 해제

#if, #else, #endif 조건에 따른 컴파일

#ifdef, #endif 매크로가 정의되어 있는 경우 컴파일

#ifndef, #endif 매크로가 정의되어 있지 않는 경우 컴파일

#line 행번호 출력

1. 단순 매크로

#define 지시자를 이용하면 숫자 상수에 의미 있는 이름을 부여할 수 있다. 이것을 단순 매크로(macro)라고 한다.

#define은 반드시 숫자 상수를 기호 상수로만 바꾸는데 한정되지 않는다. 사실 어떤 텍스트도 다른 텍스트로 바꿀 수 있다. 예를 들어서 함수의 이름도 변경해서 사용할 수 있다.

#define PRINT printf //printf대신 PRINT를 사용할 수 있다.

참고: #define과 typedef의 차이점

유사해 보이지만 상당한 차이가 있다. 먼저 typedef는 컴파일러에 의해 처리되지만 #define은 전처리기에 의해 처리된다. #define은 기계적인 텍스트 변경이다. 전처리기에 의하여 전부 원래 의미로 바뀌어 컴파일된다. 하지만 typedef은 사용자가 새로운 자료형을 정의한 것이므로 컴파일러가 컴파일 내내 그 의미를 알고 있다. 따라서 가능하면 새로운 자료형을 정의할 때는 typedef을 사용하는 편이 좋다.

1. 함수 매크로

함수 매크로(function-like macro)란 매크로가 함수처럼 매개 변수를 가지는 것이다. 함수 매크로를 사용하면 함수와 유사한 매크로를 작성할 수 있다.

예 #define SQUARE(x) ( (x) \* (x) )

매크로는 두 개 이상의 매개 변수를 가질 수 있다.

예 #define SUM(x, y) ((x) + (y))

함수 매크로에서는 매개 변수의 자료형을 써주지 않는다. 따라서 어떠한 자료형에 대해서도 적용이 가능하다. SUM 매크로는 정수뿐만이 아니라 실수 덧셈도 가능한 것이다. 함수와는 다르게 자료형에 따라서 매크로를 여러 개 만들 필요가 없으니 편리하다.

!함수 매크로 정의시 주의할 점

함수 매크로에서는 매개 변수가 기계적으로 대치되기 때문에 매크로를 정의하는 경우에 반드시 매개 변수들을 괄호에 묶어주어야 한다. 괄호로 묶지 않을 경우 본래의 의도와는 다르게 값이 나올 수도 있다. 매크로를 사용할 시에 증감식은 조심해서 사용해야 한다.

함수 매크로 VS 함수

함수 매크로의 장점은 함수에 비하여 수행 속도가 빠르다는 것이다. 함수 매크로는 함수 호출이 아니라 코드가 그 위치에 삽입되는 것이기 때문에 함수 호출의 복잡한 단계를 거칠 필요가 없다. 함수 호출을 하기 위해서는 인수와 복귀 주소를 시스템 스택에 저장하는 복잡한 절차들이 필요한데 함수 매크로는 이러한 절차들이 전혀 필요 없다. 따라서 실행 속도가 빠르다.

함수 매크로의 단점은 코드의 길이를 어느 한도 이상 길게 할 수 없다. 또한 함수 매크로가 발견될 때마다 정의된 코드가 삽입되므로 전체 소스의 길이가 길어지게 된다. 만약 30개의 함수 매크로가 있다면 30개의 동일한 코드가 프로그램에 존재하는 것이다. 함수의 경우에는 단 하나의 코드만을 가지고 있다. 따라서 함수 매크로를 사용할 경우 소스 파일의 크기가 커진다.

따라서 함수를 사용할 것이냐 함수 매크로를 사용할 것이냐는 프로그램의 크기와 실행 속도 중에서 어떤 것이 더 중요한지를 따져보아야 한다. 간단한 기능들은 함수보다는 함수 매크로를 사용하는 편이 낫다.

만약 함수 매크로가 전체 프로그램을 통하여 한 번만 사용된다면 큰 효과를 기대하기 힘들다. 하지만 함수 매크로가 중첩 반복 루프 안에 위치했다면 실행 속도가 개선될 가능성이 높다.

TIP: 매크로를 멀티 라인으로 작성하는 방법

#define PRETTY\_PRINT(s) \

Printf (“Message: \”%s\” \n”, \s);

1. #ifdef, #endif

#ifdef은 조건부 컴파일을 지시하는 전처리 지시자이다. 조건부 컴파일이란 어떤 조건이 만족되는 경우에만 지정된 소스 코드 블록을 컴파일하는 것이다. #ifdef 다음에 있는 매크로를 검사하여 매크로가 정의되어 있으면 #if와 #endif 사이에 있는 모든 문장들을 컴파일한다. 그렇지 않으면 문장들은 컴파일되지 않고 없는 취급이 된다.

#ifdef DEBUG

Printf(“Yes!”);

#endif //ifdef.c 참조

위의 문장은 DEBUG라는 매크로가 정의되어 있으면 컴파일된다. DEBUG라는 매크로는 어디서 정의하는가? 보통은 소스 코드의 첫 부분에서 정의하거나 컴파일할 때 옵션으로 제공하기도 한다.

아마 조건부 컴파일이 많이 사용되는 경우는 디버깅일 것이다. 디버깅은 디버거 등을 이용하여 할 수도 있지만 아직도 가장 많이 사용되는 전통적인 방법은 printf()를 이용하여 중간 중간 중요한 값들을 화면에 출력해보는 것이다. 문제는 이러한 디버깅을 위한 출력문은 제품이 출시될 때는 소스 코드에서 삭제되어야 한다. 이러한 경우에 조건부 컴파일이 제격이다. 매크로 정의만 삭제하면 디버깅용 출력문을 컴파일에서 제외시킬 수 있다.

#else도 있다.

1. #if와 #endif

앞 절에서 학습한 #ifdef은 매크로의 값에는 상관하지 않는다. 즉 매크로가 정의만 되어 있으면 된다. 하지만 #if는 매크로의 값에 따라서 컴파일 여부를 결정한다. If.c참조

아주 많이 사용되는 것이 #if 0를 이용한 주석 만들기이다. /\*와 \*/을 사용하여 전체를 주석으로 만들 수도 있으나 중간에 다른 주석이 있는 경우 사용이 까다롭다. 이때 #if 0 을 사용하면 손쉽게 주석을 만들 수 있다.

#elif는 #else if를 줄인 것이다.

헤더 파일 중복 포함 막기

#progma once 사용!(#include 두번 쓰는 실수를 방지)

1. 다중 소스 파일

지금까지 실습한 C프로그램은 모두 하나의 소스 파일로만 되어 있었다. 복잡한 프로그램의 경우, 하나의 파일에 모든 코드를 저장한다면 파일의 크기가 너무 커질 것이다. 파일을 편집하는 것도 쉽지 않을 것이다. C에서는 하나의 프로그램이 여러 소스 파일로 이루어 질 수 있다.

그렇다면 왜 소스 파일을 여러 개 만드는 것일까? 서로 관련된 함수들을 모아서 독립적인 소스 파일에 저장시켜 놓으면 다음에 재사용할 수 있기 때문이다.

필요한 파일들 main.c power.c power.h >> power.exe

참고

사용자가 만든 헤더 파일을 포함할 때는 “와 “을 사용한다

반면에 컴파일러가 제공하는 헤더 파일을 포함할 때는 <와 >을 사용한다.

1. 헤더 파일을 사용하는 이유

헤더 파일을 사용하지 않으려면 다른 소스 파일에서 제공하는 함수를 사용하기 전에 함수 원형을 소스 파일 첫 부분에서 선언해야 한다. 만약 소스 파일이 여러 개라면 동일한 내용이 복사되어서 들어가게 된다.

예를 들어서 그래픽 기능을 제공하는 함수들이 graphics.c에 모여 있다고 가저하자. 이 함수들을 사용하려면 다른 소스 파일에서는 이들 함수의 원형을 소스 파일 첫 부분에서 선언해야 한다. 하지만 이것은 상당히 번거로운 일이고 소스 파일이 많다면 같은 내용이 중복된다. 이런 경우에 헤더 파일을 작성해서 여기에 함수들의 원형을 넣어두고 다른 소스 파일에서는 이 헤더 파일을 포함하는 것이 좋다. 헤더 파일에는 일반적으로 함수의 원형 또는 구조체 정의. 매크로 정의, typedef의 정의를 넣어주면 좋다.

1. extern 키워드

다중 소스 파일의 경우, 하나의 프로그램에 여러 개의 소스 파일이 존재한다. 그렇다면 하나의 소스 파일에 정의되어 있는 변수를 다른 소스 파일에서 사용할 수 있을까? 지역 변수의 경우 어차피 정의된 함수를 벗어나면 사용이 불가능하다. 전역 변수의 경우, 함수의 외부에서 선언된 변수로서 그 소스 파일 안에서는 사용이 가능하다. 전역 변수가 정의된 소스 파일이외의 다른 소스 파일에서도 사용할 수 있을까? 외부 변수를 선언하면 가능하다.

외부 변수 선언은 다른 소스 파일에서 정의된 전역 변수를 사용하기 위해 extern이라는 키워드를 사용하여 그 변수를 외부 변수로 선언하는 것이다. 예를 들어서 main.c에 다음과 같이 전역 변수가 선언되었다고 가정하자.

Double gx, gy;

만약 power.c 파일에서 이 파일을 사용하려면 먼저 다음과 같이 소스 파일의 처음 부분에서 외부 변수로 선언을 해야 한다.

extern double gx, gy;

extern키워드는 변수가 외부에 선언되어 있다는 것을 컴파일러에게 알려주는 역할을 한다. extern으로 선언된 변수는 전역 변수처럼 소스 파일의 모든 함수에서 사용할 수 있다.

1. 프로그램 인수

Main()도 함수이므로 매개 변수와 반환값을 가질 수 있다.

예 int main(int argc, char \*argv[]) { … }

위의 형태에서는 두 개의 매개 변수가 선언되어 있다. Argc는 프로그램 실행 시 전달되는 인수의 개수를 의미한다. Argv는 문자열의 형태로 명령어 안에 있는 단어들을 전달한다. 여기서 argv는 문자형 포인터의 배열임을 유의하라