10장 전처리기

전처리기

- 프로그램 개발과 유지보수를 쉽게 하고, 가독성을 높이고, 프로그램 포팅을 쉽게 함
- #으로 시작하는 행을 전처리 지시자라고 함

```
#include …
#define …
#if …
…
```

전처리기

- 전처리 지시자를 위한 구문은 C 언어의 나머지 부분과 독립적임
- 전처리 지시자가 영향을 미치는 범위는 그 파일에서 전처리 지시자가 있는 위치에서 시작하여 그 파일의 끝까지이거나, 다른 지시자에 의해서 그 지시자의 효력이 없어질 때까지 임
- 전처리기는 C를 알지 못함

매크로

- 여러 가지 종류의 매크로
  - 기호 상수
  - 문자열 대치
  - 인자가 있는 매크로
- 정의가 길어질 경우에 현재 행의 끝에 역슬래시 \를 삽입하면 다음 행에 연결해서 계속 쓸 수 있음

기호 상수

- 형태

  #define 식별자 상수

  – 프로그램의 모든 식별자는 상수로 대치됨

- 선언 예제

```
#define   N       60
#define   PI      3.14
#define   C       299792.458          // km/s
#define   EOF     (-1)
#define   NULL    0
```

기호 상수

프로그램 11.1

```
#include <stdio.h>
#define  PI  3.14                  // 3.141592
double circumference(double r){
   return 2.0 * r * PI;
}
double area(double r){
   return r * r * PI;
}
double volume(double r){
   return 4.0 / 3.0 * r * r * r * PI;
}
double surface_area(double r){
   return 4.0 * r * r * PI;
}
```

기호 상수

프로그램 11.1

```
int main(void){
    double r;
    printf("반지름을 입력 하세요 : ");
    scanf("%lf", &r);
    printf("반지름이 %.3f인\n", r);
    printf("   원의 원둘레 길이 : %.3f\n", circumference(r));
    printf("   원의 면적 : %.3f\n", area(r));
    printf("   구의 부피 : %.3f\n", volume(r));
    printf("   구의 표면적 : %.3f\n", surface_area(r));
    return 0;
}
```

프로그램 결과

```
반지름을 입력 하세요 : 5.0
반지름이 5.000인
    원의 원둘레 길이 : 31.400
    원의 면적 : 78.500
    구의 부피 : 523.333
    구의 표면적 : 314.000
```

문자열 대치

- **형태**

  #define 식별자 문자열

  – 프로그램의 모든 식별자는 문자열로 대치됨

- **예제**

```
#define   EQ     ==
if (sum EQ 100) . . .      ⇒    if (sum == 100) . . .

#define   AND   &&
#define   OR    ||
```

매개 변수를 갖는 매크로

- 함수와 비슷한 모양을 가짐
- 예

```
#define    AREA(x)   ((x) * (x) * PI)
```

  - x : 매개변수
  - 전처리기는 매개변수를 실인자로 대치함

매개 변수를 갖는 매크로

프로그램 11.2

```
#define  PI  3.14
#define  CIRCUMFERENCE(x)  (2.0 * (x) * PI)
#define  AREA(x)           ((x) * (x) * PI)
#define  VOLUME(x)         (4.0 / 3.0 * (x) * (x) * (x) * PI)
#define  SURFACE_AREA(x)   (4.0 * (x) * (x) * PI)
int main(void){
   double r;
   printf("반지름을 입력 하세요 : ");
   scanf("%lf", &r);
   printf("반지름이 %.3f인\n", r);
   printf("   원의 원둘레 길이 : %.3f\n", CIRCUMFERENCE(r));
   printf("   원의 면적 : %.3f\n", AREA(r));
   printf("   구의 부피 : %.3f\n", VOLUME(r));
   printf("   구의 표면적 : %.3f\n", SURFACE_AREA(r));
   return 0;
}
```

매개 변수를 갖는 매크로

- 전처리기 처리 이후의 main() 함수

함수 11.1

```
int main(void){
    double r;
    printf("반지름을 입력 하세요 : ");
    scanf("%lf", &r);
    printf("반지름이 %.3f인\n", r);
    printf("    원의 원둘레 길이 : %.3f\n", (2.0 * (r) * 3.14));
    printf("    원의 면적 : %.3f\n", ((r) * (r) * 3.14));
    printf("    구의 부피 : %.3f\n", (4.0 / 3.0 * (r) * (r) * (r)
   * 3.14));
    printf("    구의 표면적 : %.3f\n", (4.0 * (r) * (r) * 3.14));
    return 0;
}
```

매개 변수를 갖는 매크로

- inline 함수와 유사하지만 몇 가지 차이가 있음
- 매크로는 인자의 형을 검사하지 않음

```
#define  AREA(x)    ((x) * (x) * PI)
```

  - x에 대응되는 인자의 형은 고려하지 않음

- 매크로는 인자들을 여러 번 평가함

```
#define  AREA(x)    ((x) * (x) * PI)
```

```
: AREA(r++)
   ==> ((r++) * (r++) * 3.14)
```

유의 사항

- 매크로를 정의할 때 올바른 평가 순서를 유지하기 위해 괄호를 충분히 사용해야 함
- 예

```
#define  AREA(x)    ((x) * (x) * PI)
```

```
#define  AREA(x)    x * x * PI
```

: AREA(a + b)

==> a + b * a + b * PI ≠ ((a + b) * (a + b) *PI)

유의 사항

- 매크로를 정의할 때 올바른 평가 순서를 유지하기 위해 괄호를 충분히 사용해야 함
- 예

```
#define  AREA(x)    ((x) * (x) * PI)
```

```
#define  AREA(x)    (x) * (x) * PI
```

: 4 / AREA(2)

==> 4 / (2) * (2) * PI ≠ 4 / ((2) * (2) *PI)

유의 사항

- 매크로 이름과 매개변수 사이에 공백이 있으면 안됨
- 예

```
#define  AREA(x)    ((x) * (x) * PI)
```

```
#define  AREA (x)   ((x) * (x) * PI)
: AREA(5.0)
  ==> (x)   ((x) * (x) * PI) (5.0)
```

유의 사항

- 세미콜론 사용에 유의
- 예

```
#define  AREA(x)    ((x) * (x) * PI)
```

```
#define  AREA(x)    ((x) * (x) * PI);
: if (r > 0) x = AREA(r);
  else x = -1;
  ==>
  if (r > 0) x = ((r) * (r) * PI);;
  else x = -1;
```

연산자

- "문자열화" 연산자
- 예

```
#define  string(a)   #a
```

: string(주라기 공원 I)

==>

“주라기 공원 I”

연산자

- 디버깅 코드

```
printf("i = %d\n", i);
printf("j = %d\n", j);
```

- 디버깅을 위한 # 연산자 사용

```
#define  PRINT_VAR(x) printf(#x" = %d\n", x)
```

- 변수이름 출력
- 사용 예

```
PRINT_VAR(i);
PRINT_VAR(j);
```

인자 생략

- C90에서는 매개변수를 갖는 매크로를 사용할 때 인자를 명시해야 함
- C99에서는 매개변수를 갖는 매크로를 사용할 때 인자를 생략해도 됨
- 예

```
#define print_age(x) printf(#x" : %d\n", age_##x)
: print_age();
  ==>
  printf(" " " : %d\n ", age_);
```

헤더 파일

- *.h 파일
- #include, 매크로 정의, 함수 원형, 기호 상수 등 정의
- 시스템이 제공하는 헤더 파일(표준 헤더 파일)
  - stdio.h, stdlib.h, string.h, 등
- 사용자도 자신의 헤더 파일을 만들 수 있음
  - 사용자 헤더 파일은 소스 파일과 같은 디렉터리에 생성함
- #include를 통해 프로그램에 삽입함

#include

- 뒤에 명시된 파일을 #include 문장 대신에 삽입함
- 어떤 종류의 파일도 명시해도 되지만 보통 헤더 파일(*.h)을 명시함
- 파일을 명시하는 두 가지 방법

```
#include <filename>
#include “filename”
```

#include

#include <filename>

- 표준 헤더 파일을 삽입할 때 사용
- 시스템이 정의한 디렉토리에서 filename 파일을 찾아 삽입함
- 일반적으로, 표준 헤더 파일이 저장된 장소는 시스템에 따라 다름

#include "filename"

- 사용자 헤더 파일을 삽입할 때 사용
- 먼저 현재 디렉토리에서 검색하고, 거기에 없다면 시스템이 정의한 디렉토리에서 검색하여 삽입

#include

헤더 파일 11.1 (pi.h) 프로그램 10.2 참조

```
#include <stdio.h>
#define  PI  3.14
#define  CIRCUMFERENCE(x)  (2.0 * (x) * PI)
#define  AREA(x)           ((x) * (x) * PI)
#define  VOLUME(x)         (4.0 / 3.0 * (x) * (x) * (x) * PI)
#define  SURFACE_AREA(x)    (4.0 * (x) * (x) * PI)
```

프로그램 11.4

```
#include "pi.h"
int main(void){
    double r;

    printf("반지름을 입력 하세요 : ");
    scanf("%lf", &r);
    printf("반지름이 %.3f인\n", r);
    printf("    원의 원둘레 길이 : %.3f\n", CIRCUMFERENCE(r));
    printf("    원의 면적 : %.3f\n", AREA(r));
    printf("    구의 부피 : %.3f\n", VOLUME(r));
    printf("    구의 표면적 : %.3f\n", SURFACE_AREA(r));

    return 0;
}
```

조건부 컴파일

- 프로그램의 특정 부분을 선택적으로 컴파일 할 수 있게 하는 기능
- 이식성이 높은 프로그램을 만들 수 있게 함
- 관련 전처리 지시자

  #if, #ifdef, #else, #elif, #endif, #undef

#undef

- 앞에서 정의한 매크로를 무효화 함

  ```
  #undef PI
  ```

  – 이 문장 이후로는 PI 매크로를 사용할 수 없음

- 매크로를 다시 정의할 때 주로 사용됨

#ifdef, #ifndef, #else, #endif

- #ifdef, #ifndef 뒤에는 식별자(매크로 이름)가 옴
- 뒤에 오는 식별자가 #define에 의해 정의되어 있는지 검사
- 조건이 참이면 #endif 사이의 문장이 적용되고, 거짓이면 #endif 사이의 문장은 무시됨
- 예

```
#ifdef PI
#undef PI
#endif
#define PI 3.1415926535
```

  - PI가 정의되어 있건 없건 간에 PI를 3.1415926535로 정의함

#ifdef, #ifndef, #else, #endif

- #ifdef, #ifndef 는 #endif로 끝나기 때문에 들여쓰기를 하여 문장들이 구분되게 하는 것이 좋음
- 예

```
#ifdef PI
#    undef PI
#endif
#define PI 3.1415926535

#ifdef PI
    #undef PI
#endif
#define PI 3.1415926535
```

#ifdef, #ifndef, #else, #endif

- #else는 #ifdef, #ifndef가 거짓일 때 선택 됨
- 예

```
#ifdef   PI
   area = r * r * PI;
#else
   area = r * r * 3.14;
#endif

#ifndef   PI
   area = r * r * 3.14;
#else
   area = r * r * PI;
#endif
```

– PI가 선언되어 있지 않으면 PI대신 3.14를 곱함

#if, #elif, #else, #endif

- #if와 #elif 뒤에는 정수 수식이 옴
- 예

```
#if  N < 10
    int  num[100];
#elif N > 2000
    int  num[2000];
#else
    int  num[N];
#endif
```

  - N 값에 따라 num 배열 크기를 다르게 함

defined

- defined 뒤에 명시된 식별자가 정의되어 있는지 확인
- 정의되어 있으면 1, 아니면 0의 값을 가짐
- #if와 같이 사용하면 #ifdef와 같은 결과를 얻음
- 예

```
#if defined(PI)
    #undef  PI
#endif
```

디버깅 코드

- 조건부 컴파일 기법은 디버깅 코드를 위해서 많이 사용됨
- 예

```
#define   DEBUG   1       // 0
. . .
#if DEBUG
   printf("debug: a = %d\n", a);  // 디버깅 코드
#endif
```

  - 디버깅이 필요하면 DEBUG를 1로 아니면 0으로 정의함
  - DEBUG 값에 따라 디버깅 코드가 삽입되거나 무시됨

디버깅 코드

- #define 대신 컴파일 옵션을 사용할 수도 있음
  - gcc 경우 -D 옵션
- 예

```
$ gcc -D DEBUG test.c
$ gcc -D DEBUG=1 test.c
```

  - 이것은 test.c 파일의 첫 행에 다음과 같은 행이 있는 것과 같음

```
#define DEBUG
#define DEBUG 1
```

조건부 컴파일

프로그램 11.5

```
#define   N   10
int main(void){
   int grade[N];
   int i, j, sum, tmp;
   float average;
   // 성적 입력
   for (i = 0; i < N; i++) {
      printf("%d 번째 성적을 입력하세요 : ", i);
      scanf("%d", &grade[i]);
   }
#ifdef DEBUG
   {
      int j;
      printf("입력 데이터 : ");
      for (j = 0; j < N; j++)
         printf("%d%s", grade[j], j == N-1 ? "\n" : ", ");
   }
#endif
```

조건부 컴파일

프로그램 11.5

```
   // 평균 계산
   . . .
   // 성적 정렬 : 버블 정렬 사용
   for (i = 0; i < N-1; i++){
      for (j = N-1; j > i; j--)
         if (grade[j - 1] > grade[j]){
            tmp = grade[j - 1];
            grade[j - 1] = grade[j];
            grade[j] = tmp;
         }
#ifdef DEBUG
      {
         int j;
         printf("%d 단계 : ", i);
         for (j = 0; j < N; j++)
            printf("%d%s", grade[j], j == N-1 ? "\n" : ", ");
      }
#endif
   }
   . . .
```

프로그램 결과

```
$ gcc grade.c -o grade -D DEBUG
$ grade
0 번째 성적을 입력하세요 : 80
1 번째 성적을 입력하세요 : 90
2 번째 성적을 입력하세요 : 99
3 번째 성적을 입력하세요 : 77
4 번째 성적을 입력하세요 : 89
5 번째 성적을 입력하세요 : 60
6 번째 성적을 입력하세요 : 88
7 번째 성적을 입력하세요 : 83
8 번째 성적을 입력하세요 : 94
9 번째 성적을 입력하세요 : 97
입력 데이터 : 80, 90, 99, 77, 89, 60, 88, 83, 94, 97
성적 평균 : 85.699997
0 단계 : 60, 80, 90, 99, 77, 89, 83, 88, 94, 97
1 단계 : 60, 77, 80, 90, 99, 83, 89, 88, 94, 97
2 단계 : 60, 77, 80, 83, 90, 99, 88, 89, 94, 97
3 단계 : 60, 77, 80, 83, 88, 90, 99, 89, 94, 97
4 단계 : 60, 77, 80, 83, 88, 89, 90, 99, 94, 97
5 단계 : 60, 77, 80, 83, 88, 89, 90, 94, 99, 97
6 단계 : 60, 77, 80, 83, 88, 89, 90, 94, 97, 99
7 단계 : 60, 77, 80, 83, 88, 89, 90, 94, 97, 99
8 단계 : 60, 77, 80, 83, 88, 89, 90, 94, 97, 99
성적순 : 60 77 80 83 88 89 90 94 97 99
```