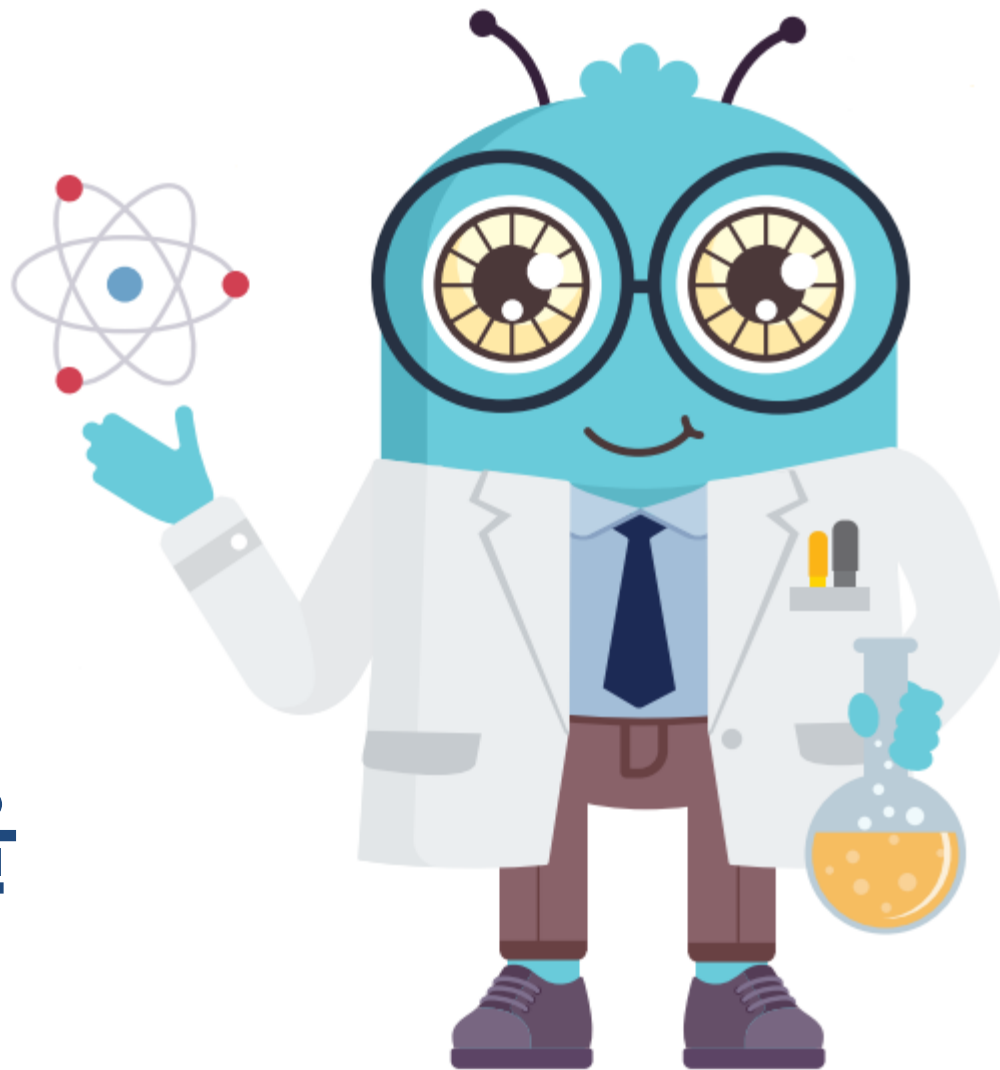


Chapter 14

고급 프로그램을 만들기 위한 C



목차

1. `main()` 함수
2. 헤더 파일
3. 전처리문과 예약어

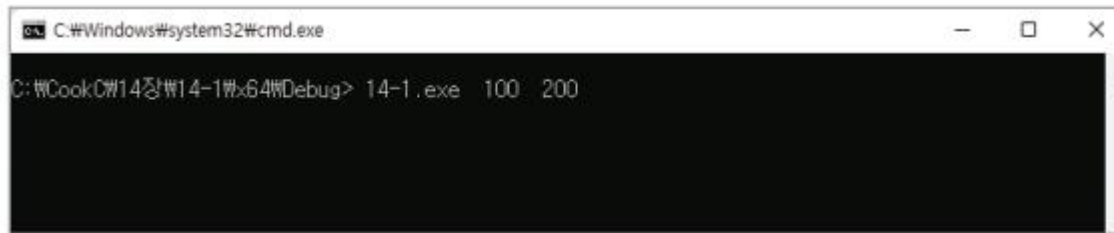
01

main() 함수

1. main() 함수

1. main() 함수의 매개변수

- 매개변수를 받아서 main() 함수를 만들고 빌드한 후 실행하는 형식



```
C:\Windows\system32\cmd.exe
C:\CookC\14장\14-1\64\Debug> 14-1.exe 100 200
```

그림 14-1 main() 함수에서의 매개변수 사용

1. main() 함수

1. main() 함수의 매개변수

기본 14-1 main() 함수에서의 매개변수 사용 예 1

14-1.c

```
01 #include <stdio.h>
02
03 void main(int argc, char* argv[]) — 매개변수를 지정한다.
04 {
05     int i;
06
07     printf(" argc 매개변수 ==> %d \n", argc); — 매개변수의 개수를 출력한다.
08
09     for(i=0; i < argc; i++) — 매개변수의 개수만큼 반복하여
10         printf(" argv[%d] 매개변수 ==> %s \n", i, argv[i]); 매개변수의 내용을 출력한다.
11 }
```

실행 결과 ▼

```
C:\Windows\system32\cmd.exe
C:\CookC\14장\14-1\Win64\Debug> 14-1.exe 100 200
argc 매개변수 ==> 3
argv[0] 매개변수 ==> 14-1.exe
argv[1] 매개변수 ==> 100
argv[2] 매개변수 ==> 200

C:\CookC\14장\14-1\Win64\Debug> 14-1.exe AAA BBB CCC
argc 매개변수 ==> 4
argv[0] 매개변수 ==> 14-1.exe
argv[1] 매개변수 ==> AAA
argv[2] 매개변수 ==> BBB
argv[3] 매개변수 ==> CCC

C:\CookC\14장\14-1\Win64\Debug>
```

1. main() 함수

1. main() 함수의 매개변수

응용 14-2 main() 함수에서의 매개변수 사용 예 2

14-2.c

```
01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03 void main(int argc, char* argv[])
04 {
05     char str[200];
06     FILE *rfp;
07
08     if(__1__ != 2) ----- 매개변수가 하나가 아니면 메시지를 출력한 후
09     { ----- 프로그램을 종료한다.
10         printf("\n -- 매개변수를 1개 사용하세요 --\n");
11         return;
12     }
13
14     rfp=fopen(__2__, "r"); ----- 첫 번째 매개변수로 넘어온 것을 읽기 모드로 연다.
15
16     for( ; ; ) ----- 무한 루프이다.
17     {
18         fgets(str, 199, rfp); ----- 파일의 내용을 한 줄 읽어들인다.
19
20         iffeof(rfp) ----- 파일의 끝이면 무한 루프를 종료한다.
```

1. main() 함수

1. main() 함수의 매개변수

```
21      break;
22
23      printf("%s", str);
24  }
25
26  fclose(rfp);
27 }
```

읽을 내용을 출력한다.

파일을 닫는다.

읽을 1 argc 2 argv[1]

실행 결과 ▼

```
C:\Windows\system32\cmd.exe
C:\CookC\14장\14-2\14-2\Debug> 14-2.exe
-- 매개변수를 1개 사용하세요 --
C:\CookC\14장\14-2\14-2\Debug> 14-2.exe AAA BBB
-- 매개변수를 1개 사용하세요 --
C:\CookC\14장\14-2\14-2\Debug> 14-2.exe C:\windows\win.ini
: for 16-bit app support
[fonts]
[extensions]
[mci extensions]
[files]
[Mail]
MAPI=1
C:\CookC\14장\14-2\14-2\Debug>
```

02

헤더 파일

2. 헤더 파일

1. 헤더 파일의 이해

- *.h라는 확장자를 사용하는 파일

- 파일에는 함수의 프로토 타입이 선언되어 있으며 구조체 등의 데이터 구조가 정의되어 있음
- 예) printf() 함수 : 프로그램의 맨 첫 줄에 "#include <stdio.h>"를 썼기 때문에 printf() 함수를 만든 적이 없지만 화면에 무언가를 출력해주는 역할을 수행해옴

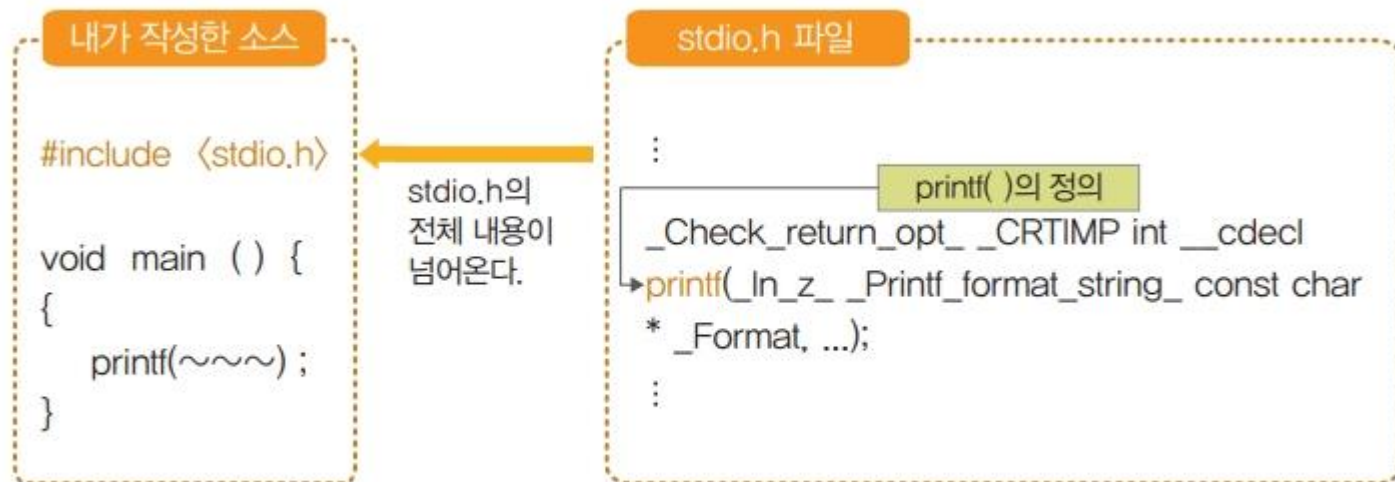


그림 14-2 #include의 개념

2. 헤더 파일

- 헤더 파일과 관련된 함수의 종류

표 14-1 주요 헤더 파일과 관련 함수

헤더 파일	설명	관련 함수
stdio.h	표준 입출력 관련	printf() scanf() puts() gets() fopen() fclose() ...
string.h	문자열 관련	strcat() strcmp() strlen() strcpy() ...
math.h	수학 관련	sin() cos() abs() pow() sqrt() log10() ...
malloc.h	메모리 관련	malloc() realloc() calloc() free() ...
stdlib.h	C 표준 라이브러리 관련	exit() rand() system() ...
time.h	시간 관련	clock() time() localtime() ...

2. 사용자가 만드는 헤더 파일

- 사용자가 만든 헤더 파일의 이름이 'myheader.h' 일 경우

```
#include "myheader.h"
```

2. 헤더 파일

■ 헤더 파일을 만드는 방법

- 01 Visual Studio를 실행한 뒤 '14_3'이라는 이름으로 프로젝트를 생성하고 '14_3.c' 파일 생성
- 02 왼쪽 솔루션 탐색기의 [14_3]-[헤더 파일]에서 마우스 오른쪽 버튼 클릭 후 [추가 (D)]-[새 항목]을 선택



그림 14-3 헤더 파일 추가 1

2. 헤더 파일

- 03 [헤더 파일]을 선택하고 이름에 "myHeader.h"를 입력한 후 <추가>를 클릭

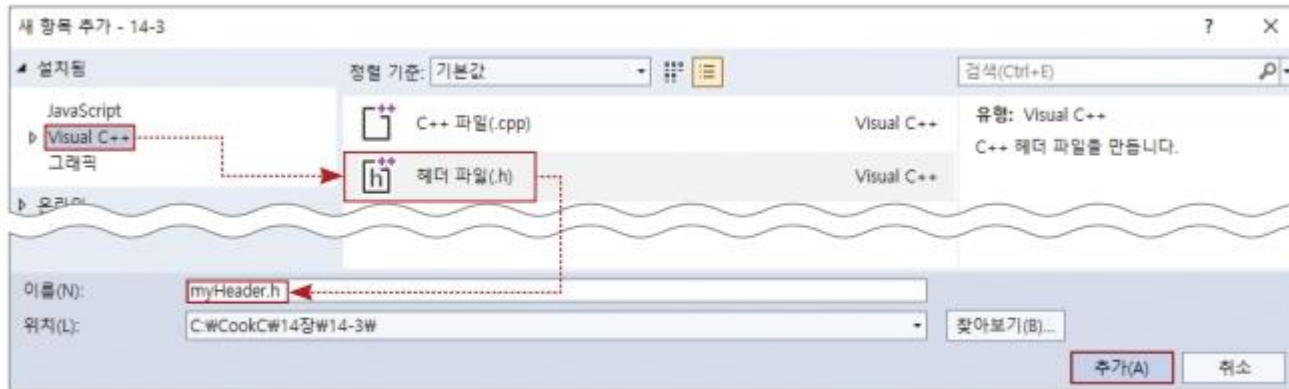


그림 14-4 헤더 파일 추가 2

- 04 헤더 파일의 내용에는 다음 한 줄을 입력

```
void func(int);
```

2. 헤더 파일

- 05 C 소스 파일(14_3.c)은 다음과 같이 코딩

기본 14-3 사용자 정의 헤더 파일 사용 예

14-3.c

```
01 #include <stdio.h>
02
03 #include "myHeader.h"  ——— 사용자가 만든 헤더 파일을 포함한다.
04
05 void main( )
06 {
07     func(10);  ——— 함수를 호출한다.
08 }
09
10 void func(int num)  ——— func( ) 함수의 내용을 정의한다.
11 {
12     printf("%d\n", num);
13 }
```

실행 결과

10

2. 헤더 파일

3. 하나의 파일을 여러 개의 파일로 분리하기

■ 파일 분리 과정

- 01 Visual Studio를 실행 → '14_4'라는 이름으로 프로젝트 생성 → 솔루션 탐색기의 [14_4]-[헤더 파일]에서 마우스 오른쪽 버튼을 클릭 → [추가]-[새 항목] 누름
- 02 중간 창에서 [헤더 파일]을 선택 → 이름에 'func.h'를 입력 → <추가> 클릭
- 03 [14_4]-[소스 파일]에서 마우스 오른쪽 버튼 클릭 → [추가]-[새 항목] 선택
- 04 중간 창에서 [C++ 파일]을 선택 → 이름에 'main.c'를 입력 → <추가> 클릭
- 05 'func1.c'와 'func2.c' 추가 → 최종적으로 다음과 같이 파일 4개가 추가됨

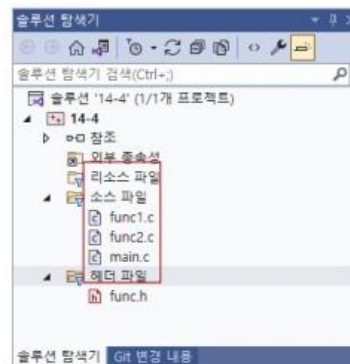


그림 14-6 파일 분리

2. 헤더 파일

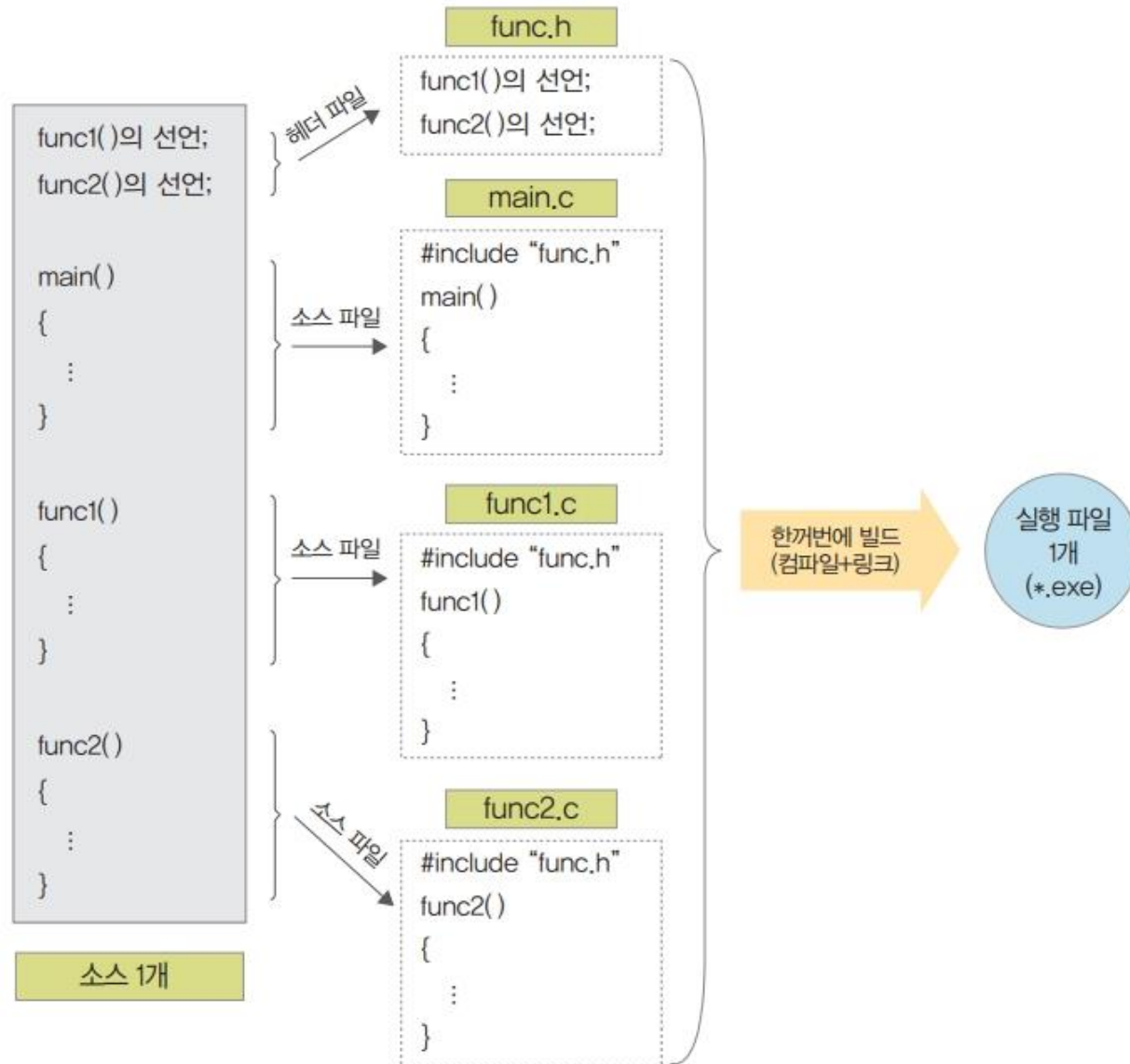


그림 14-5 파일 분리의 개념

2. 헤더 파일

3. 하나의 파일을 여러 개의 파일로 분리하기

기본 14-4 함수별로 파일을 분리하는 예

14-4.c

func.h

```
01 #include <stdio.h>
```

```
02
```

```
03 void func1( );
```

```
04 void func2( );
```

----- func1(), func2() 함수의 프로토타입을 선언한다.

main.c

```
01 #include "func.h"
```

```
02
```

```
03 void main( )
```

```
04 {
```

```
05     func1( );
```

```
06     func2( );
```

```
07 }
```

----- 직접 만든 헤더 파일을 포함한다.

----- func1(), func2() 함수를 호출한다.

2. 헤더 파일

3. 하나의 파일을 여러 개의 파일로 분리하기

func1.c

```
01 #include "func.h"
```

—— 직접 만든 헤더 파일을 포함한다.

```
02
```

```
03 void func1( )
```

—— func1() 함수의 내용을 정의한다.

```
04 {
```

```
05     printf("func1( ) 함수를 실행합니다. \n");
```

```
06 }
```

func2.c

```
01 #include "func.h"
```

—— 직접 만든 헤더 파일을 포함한다.

```
02
```

```
03 void func2( )
```

—— func2() 함수의 내용을 정의한다.

```
04 {
```

```
05     printf("func2( ) 함수를 실행합니다. \n");
```

```
06 }
```

실행 결과

func1() 함수를 실행합니다.

func2() 함수를 실행합니다.

03

전처리문과 예약어

3. 전처리문과 예약어

1. 전처리문

- 실제 컴파일 전에 미리 처리되는 문장
- 기존의 방대한 소스를 건드리지 않은 상태에서 부분적인 컴파일 수행
- 소스의 시작부분에 위치하며, #으로 시작
- #include, #define, #ifdef, #undef 등

■ #define 문

- 소스 코드에 사용할 숫자나 문자열, 함수의 이름이 너무 길거나 복잡할 때 한 눈에 파악하도록 쉬운 기호로 표현

#define [기호] [숫자 또는 문자열 또는 함수]

- 예) 원주율 표시 - 3.1415926535

```
void main( ){  
...  
    area = rad * rad * 3.1415926535;  
...  
}
```



```
#define PI 3.1415926535  
void main( ){  
...  
    area = rad * rad * PI;  
...  
}
```

3. 전처리문과 예약어

1. 전처리문

- #define 문

기본 14-5 #define문 사용 예

14-5.c

```
01 #include <stdio.h>
02
03 #define PI 3.1415926535
04 #define STR "원의 면적을 계산했습니다.\n"
05 #define END_MSG printf("프로그램이 종료되었습니다. \n\n")
06
07 void main( )
08 {
09     printf("반지름이 10인 원의 면적은 ==> %10.5f \n", 10*10*PI);
10
11     printf(STR);
12
13     END_MSG;
14 }
```

실행 결과

반지름이 10인 원의 면적은 ==> 314.15927
원의 면적을 계산했습니다.
프로그램이 종료되었습니다.

3. 전처리문과 예약어

1. 전처리문

- #define 문

❶ #define을 사용하지 않은 경우

```
#include <stdio.h>

void main( )
{
    printf("반지름이 10인 원의 면적은 ==>
           %10.5f \n", 10*10*3.1415926535)
    printf("원의 면적을 계산했습니다. \n");
    printf("원의 면적을 계산했습니다. \n");
    printf("원의 면적을 계산했습니다. \n");
    printf("원의 면적을 계산했습니다. \n");
    printf("프로그램이 종료되었습니다. \n\n")
}
```

❷ #define을 사용한 경우

```
#include <stdio.h>
#define STR "원의 면적을 계산했습니다.\n"
void main( )
{
    printf("반지름이 10인 원의 면적은 ==>
           %10.5f \n", 10*10*3.1415926535)
    printf(STR);
    printf(STR);
    printf(STR);
    printf(STR);
    printf(STR);
    printf("프로그램이 종료되었습니다. \n\n")
}
```

3. 전처리문과 예약어

2. 예약어

- 특별한 기능을 수행하도록 프로그래밍 언어에서 미리 정의한 것
- 변수나 함수 이름에 사용할 수 없음
- 컴파일러는 예약어를 변수 키워드로 인식하므로 변수 역할을 하지 못함

```
auto, case, cdecl, const, char, continue, default, do, double, else, enum, extern,  
float, for, goto, if, int, long, register, return, short, signed, sizeof, static,  
struct, switch, typedef, union, unsigned, void, volatile, while
```

▪ const 예약어

- #define과 비슷한 기능
- const로 변수를 선언하면 해당변수는 변수로서의 역할을 하는 것이 아니라 상수 역할을 함

```
const 변수형 변수_이름 = 변수값;
```

3. 전처리문과 예약어

- const의 사용 방법

```
const int a = 100;    ⇒ a를 100으로 고정한다.  
int b;
```

```
b = a + 200;          ⇒ (○) a는 100이므로 b는 300이 된다.
```

```
a = 200;              ⇒ (×) a는 더 이상 변수가 아니며 고정된 값으로만 사용해야 한다.
```

- const를 사용한 위의 예를 #define으로 바꿈

```
#define a 100          ⇒ 100을 기호 a로 정의한다.  
int b;
```

```
b = a + 200;          ⇒ (○) a는 100과 동일하므로 b는 300이 된다.
```

```
a = 200;              ⇒ (×) 100=200이라고 쓴 것이므로 문법적으로 틀린 문장이다.
```

▪ static 예약어

- 처음 설정된 값을 초기화하지 않고 계속 유지하게 함

```
static 변수형 변수_이름 = 변수값;
```

3. 전처리문과 예약어

- static 예약어

기본 14-6 static 예약어 사용 예

14-6.c

```
01 #include <stdio.h>
02
03 void myfunc( );           —— myfunc( ) 함수의 프로토타입을 선언한다.
04
05 void main( )
06 {
07     myfunc( );           —— myfunc( ) 함수를 두 번 호출한다.
08     myfunc( );
09 }
10
11 void myfunc( )
12 {
13     static int a = 0;     —— static 예약어로 a 변수를 선언하고 초기화한다.
14
15     a = a + 100;         —— a에 100을 증가시킨 후 출력한다.
16     printf("a의 값 ==> %d\n", a);
17 }
```

실행 결과

a의 값 ==> 100

a의 값 ==> 200

3. 전처리문과 예약어

▪ extern 예약어

- 다른 소스 파일에 선언된 전역 변수를 현재의 소스 파일에 가져와서 사용하고 싶을 때 사용함

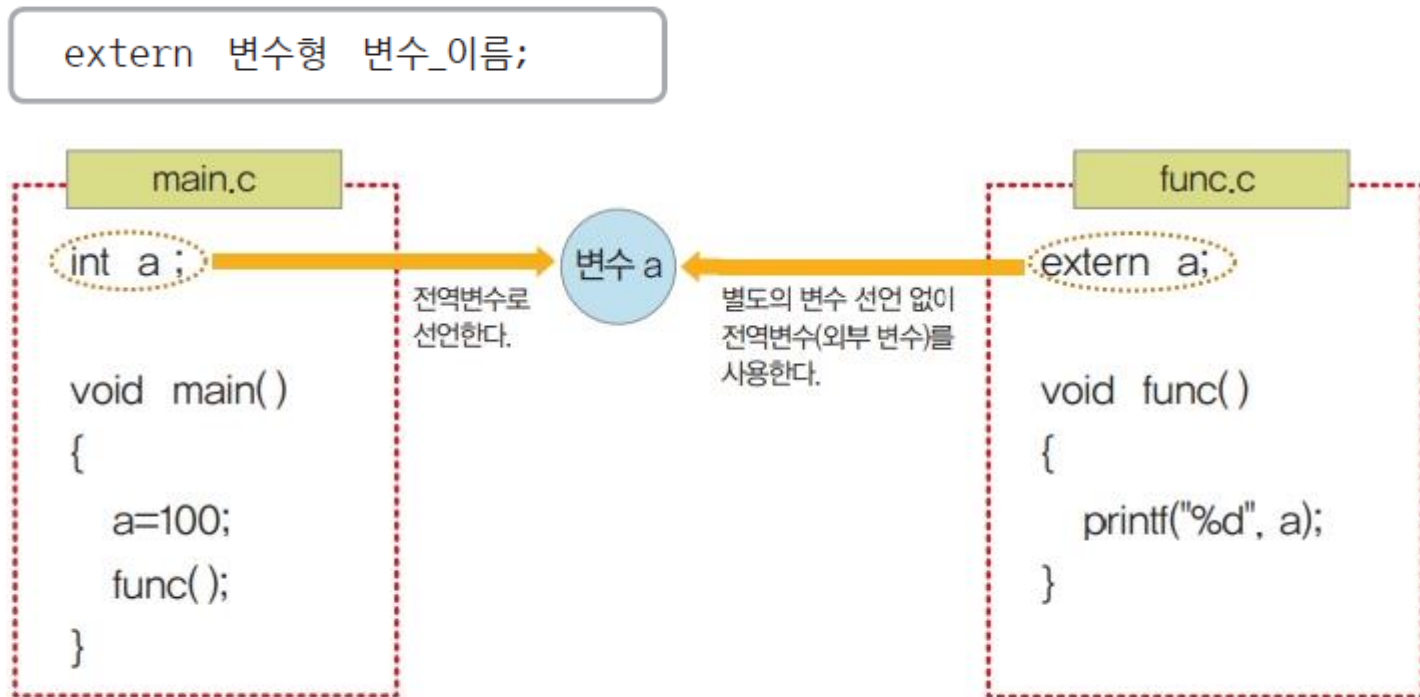


그림 14-7 extern 예약어 사용

3. 전처리문과 예약어

- extern 예약어

기본 14-7 extern 예약어 사용 예 1

14-7.c

main.c

```
01 #include <stdio.h>
```

```
02
```

```
03 void func( );
```

----- func() 함수의 프로토타입을 선언한다.

```
04
```

```
05 int a;
```

----- 전역변수 a를 선언한다.

```
06
```

```
07 void main( )
```

```
08 {
```

```
09     a = 100;
```

----- 전역변수에 값을 대입한다.

```
10
```

```
11     func( );
```

----- 함수를 호출한다.

```
12 }
```

3. 전처리문과 예약어

- extern 예약어

func.c

```
01 #include <stdio.h>
```

```
02
```

```
03 extern int a; ----- 변수 a를 외부 변수로 선언한다.
```

```
04
```

```
05 void func( )
```

```
06 {
```

```
07     printf("extern int a값 ==> %d\n", a); ----- 변수 a의 값을 출력한다.
```

```
08 }
```

실행 결과

```
extern int a값 ==> 100
```

3. 전처리문과 예약어

- extern 예약어

응용 14-8 extern 예약어 사용 예 2

14-8.c

main.c

```
01 #define _CRT_SECURE_NO_WARNINGS
```

```
02 #include <stdio.h>
```

```
03 void exchange( );
```

—— exchange() 함수의 프로토타입을 선언한다.

```
04
```

```
05 int a, b;
```

—— 전역변수 a와 b를 선언한다.

```
06
```

```
07 void main( )
```

```
08 {
```

```
09     printf("a의 값을 입력 : ");
```

```
10     scanf("%d", &a);
```

—— 전역변수 a에 값을 입력한다.

```
11     printf("b의 값을 입력 : ");
```

```
12     scanf("%d", &b);
```

—— 전역변수 b에 값을 입력한다.

```
13
```

```
14     __1__( );
```

—— 함수를 호출한다.

```
15
```

```
16     printf("\n바뀐 값 a는 %d, b는 %d \n", a, b);
```

—— 결과를 출력한다.

```
17 }
```

3. 전처리문과 예약어

- extern 예약어

exchange.c

```
01  2 ----- 변수 a, b를 외부 변수로 선언한다.  
02  
03  void exchange( )  
04  {  
05      int tmp;  
06  
07      tmp = a; ----- 두 값을 교환한다.  
08      a = b;  
09      b = tmp;  
10  }
```

실행 결과 1 exchange 2 extern int a, b

실행 결과

a의 값을 입력 : 111

b의 값을 입력 : 222

바뀐 값 a는 222, b는 111

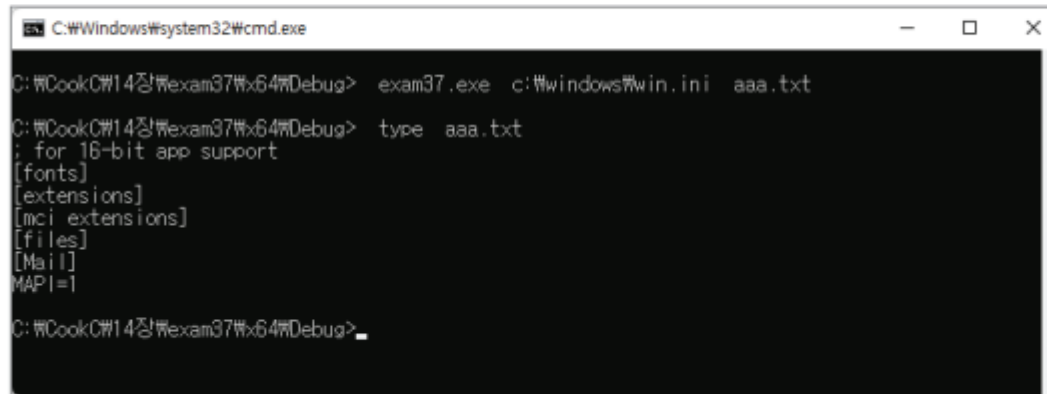
*

예제 모음

[예제모음 37] 텍스트 파일 복사

예제 설명 도스 명령어 중 copy 명령어와 동일하게 텍스트 파일을 복사하는 프로그램이다.

실행 결과



```
C:\Windows\system32\cmd.exe
C:\CookC\14장\exam37\64\Debug> exam37.exe c:\windows\win.ini aaa.txt
C:\CookC\14장\exam37\64\Debug> type aaa.txt
; for 16-bit app support
[fonts]
[extensions]
[mci_extensions]
[files]
[Mail]
MAP1=1
C:\CookC\14장\exam37\64\Debug> _
```

[예제모음 37] 텍스트 파일 복사

```
01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03 void main(int argc, char* argv[])
04 {
05
06     char str[200];          — 한 행을 최대 199자로 제한한다.
07     FILE *rfp;              — 읽기용 포인터와 쓰기용 포인터 변수를 선언한다.
08     FILE *wfp;
09
10     if(argc != 3)           — 매개변수가 2개가 아닌 경우
11     {                       프로그램을 종료한다.
12         printf("\n — 매개변수를 2개 사용하세요 —\n");
13         return;
14     }
15
16     rfp=fopen(argv[1], "r"); — 읽기 모드의 파일을 연다.
17     wfp=fopen(argv[2], "w"); — 쓰기 모드의 파일을 연다.
18
19     for( ; ; )              — 무한 루프이다.
20     {
21         fgets(str, 199, rfp); — 원본 파일에서 한 행을 읽어온다.
22     }
```


[예제모음 37] 텍스트 파일 복사

```
23     if(feof(rfp))           ----- 파일의 끝이면(더 읽을 행이 없으면) for문을 종료한다.
24         break;
25
26     fputs(str, wfp);         ----- 읽은 행을 대상 파일에 쓴다.
27 }
28
29 fclose(rfp);                ----- 파일을 닫는다.
30 fclose(wfp);
31 }
```

[예제모음 38] static 예약어를 활용한 구구단 출력

예제 설명 static 예약어를 이용하여 구구단을 출력하는 프로그램이다.

실행 결과

**** 2 단 ****

2 X 1 = 2

2 X 2 = 4

2 X 3 = 6

2 X 4 = 8

2 X 5 = 10

2 X 6 = 12

2 X 7 = 14

2 X 8 = 16

⋮

9 X 1 = 9

9 X 2 = 18

9 X 3 = 27

9 X 4 = 36

9 X 5 = 45

9 X 6 = 54

9 X 7 = 63

9 X 8 = 72

9 X 9 = 81

[예제모음 38] static 예약어를 활용한 구구단 출력

```
01 #include <stdio.h>
02
03 void gugu( )
04 {
05     static int dan=1;
06     int i;
07
08     dan++;
09
10     printf("\n\n ** %d 단 **\n", dan);
11     for(i=1; i <= 9; i++)
12     {
13         printf("%2d X %2d= %2d  \n", dan, i, dan*i);
14     }
15 }
16
17 void main( )
18 {
19     int i;
20
21     for(i=0; i < 8; i++)
22         gugu( );
23 }
```

—— static 예약어로 dan 변수를 선언하고 초기화한다.

—— 단을 하나씩 증가시킨다.

—— 제목을 출력한다.

—— 각 단의 구구단을 출력한다.

—— gugu() 함수를 여덟 번 호출한다.

감사합니다!

