IT 백두대간 C 언어 펀더멘탈
:견고한 프로그램을 위한 기본 원리
한빛미디어

필드 테스트 모범 답안 (4장)

최종 수정일: 2003년 4월 30일

- 이곳을 통해 제공되는 필드 테스트의 답안은 말 그대로 정답이 아닌 모범 답안입니다. 물론 수학 문제처럼 다른 가능성의 여지가 없는 문제도 있지만, 다수의 문제는 다양한 생각이 모두 올바른 답이 될 수 있는 것들입니다. 제시된 답에 대해 의문이 있거나 부족하다고 판단되는 경우, 망설임 없이 저자의 메일 (mycoboco@hanmail.net) 이나 게시판 (http://c-expert.uos.ac.kr) 을 통해 알려주시기 바랍니다. 적절한 의견은 추후 답안에 반영하도록 하겠습니다.

- 답이 생략된 문제는 이곳을 통해 충분한 답을 제공하기 어렵거나, 이미 필요한 방법 등을 본문에서 보이고 독자 여러분이 자신의 환경에서 직접 실습해보길 추천하는 수준의 문제입니다. 다양한 환경에 대한 정보를 나누기 위해, 독자 여러분이 직접 경험한 사실을 메일이나 게시판을 통해 제공해주시면, 이 역시 검토 후 답안에 반영하도록 하겠습니다.

1. 137쪽의 [예제 4-1]을 공백만으로 토큰 단위로 구분하여 다시 쓰면 다음과 같다.

```
# include <stdio.h> # define rparen ) int main ( void ) { printf ( "Hello\n" rparen ; return 0 ; )
```

하지만, 이 프로그램은 정상적으로 번역, 실행되지 않는다. 이유는 #include, #define 으로 시작하는 전처리기 지시자는 개행문자에 특별한 의미를 두기 때문이다 (이와 관련된 보다 자세한 내용은 14장에서 다룬다). 따라서, 운 좋게도 45쪽에서 보인 전처리기 지시자를 사용하지 않는 예제를 선택했다면 원하는 결과를 얻겠지만, 대부분의 예제는 전처리기 지시자를 사용하고 있기에 번역 과정에서 의도하지 않은 진단 메시지를 얻게 될 것이다. 물론, 처음 선택했던 [예제 4-1] 역시 전처리기 지시자만을 개행문자로 분리해 준다면 공백만으로 토큰을 분리해 정상적으로 번역, 실행됨을 확인할 수 있다.

```
#include <stdio.h>
# define rparen )
```

```
int main ( void ) { printf ( "Hello\n" rparen ; return 0 ; }
```

2. 각 명칭에 대해서 설명하면 다음과 같다.
   (1) 올바른 명칭이다. 하지만 C++ 에서는 예약어로 정의되어 있기 때문에 C++ 과의 호환성을 고려한다면 사용을 피하는 것이 좋다.
   (2) 숫자로 시작하므로 올바르지 않은 명칭이다. 이는 [2files] 의 전처리기 숫자로 토큰화되고 2files 가 정상적인 형태의 상수가 될 수 없으므로 잘못된 프로그램이 된다.
   (3) 올바른 명칭이다. 하지만 예약어 switch 와 혼동될 우려가 있으므로 사용을 피하는 것이 좋다.
   (4) 올바른 명칭이다. 하지만, 8진 정수상수 0157 과 혼동될 우려가 있으므로 사용을 피하는 것이 좋다.
   (5) 올바르지 않은 명칭이다. 명칭을 구성하는 문자로는 + 를 사용할 수 없으므로 하나의 명칭이 아닌 [String][+][nullchr] 로 토큰화된다.
   (6) $ 로 인해 올바르지 않은 명칭이다. $ 의 존재 자체는 표준의 관점에서 해당 프로그램을 잘못된 프로그램으로 만들지만, 임플리멘테이션이 특별한 목적을 위해 확장으로 $ 문자를 명칭에 사용할 수 있도록 지원해주는 것은 가능하다.
   (7) 올바른 명칭이다. 하지만, 명칭의 의미를 이해하기 쉽지 않으므로 가독성을 위해 의미가 분명히 드러나도록 수정하는 것이 좋다.
   (8) 올바른 명칭이다.

3. 표준에서 공백은 토큰을 강제 분리하는 역할을 하기 때문에

   ```
   x+++ ++y;
   ```

   는 [x][++][+][++][y] 로 토큰화된다. x, y 과 증감 연산자를 적용할 수 있는 데이터형으로 선언되고 정상적인 값을 갖는다면 이 수식은 아무 문제 없이 실행될 수 있다.

4. 이중자 및 삼중자를 정상적으로 지원하고 표준이 요구하는 모든 문자를 (정확히는 문자의 모양 glyph 을) 정상적으로 지원하는 임플리멘테이션이라면 [예제 4-2]는 다음과 같은 결과를 출력하도록 의도된 것이다.

```
In music, # is used to represent sharp.
In music, %: is used to represent sharp? No!
~
```

```
<::>
```

이중자 혹은 삼중자가 문자열 내에서도 치환되는지 여부, 또 전처리기 연산자인 # 를 적용했을 때의 결과 등에 유의하자.

5. 다음의 다소 이상하게 생긴 프로그램은 C99 에 새로 추가된 C++ 스타일의 주석으로 인해 결과에 차이가 생기는 경우를 보여준다.

```
#include <stdio.h>

int main(void)
{
    int i = 15 //* strange comment */ 5;
    ;
    printf("%d\n", i);

    return 0;
}
```

C++ 스타일의 주석이 지원되지 않는 경우 (C90/C95) 에는 3 이, C++ 스타일의 주석이 지원되는 경우 (C99/C++/표준을 따르지 않는 임플리멘테이션의 확장) 에는 15 가 출력된다.

6. 필자가 사용하는 임플리멘테이션에서는 [예제 4-6]의 결과는 -16 과 240 으로 다르게 나왔다 (1바이트에 8비트를 할당하고 음수 표현에 2의 보수를 사용하는 대부분의 임플리멘테이션에서 동일한 결과를 보일 것이다). 하지만, char 형을 unsigned char 형과 동일하게 다루는 임플리멘테이션이나 signed char 형이 240 (0xF0) 을 표현할 수 있도록 1바이트에 9비트 이상의 비트 (값 비트와 부호 비트) 를 할당한 임플리멘테이션에서는 두 결과가 동일한 값으로 출력된다 (사용 중인 임플리멘테이션에 char 형을 unsigned char 형과 동일하게 다루는 옵션이 존재한다면 이를 이용해 프로그램의 실행 결과를 확인해 보자).

7. (생략)

8. "C:\application\freesoftware\nobo.exe" 라는 문자열에서 노란색으로 강조된 부분은 원래 의도와는 달리 확장 문자열로 인식되는 부분이다.

9. (특정 옵션 생략)

비록 특정 임플리멘테이션이 (표준을 따르지 않는) 확장으로 중첩된 주석을 허락해 준다고 해도 이를 현재 작성하는 프로그램에 사용하는 것은 이식성이라는 관점에서 볼 때 결코 바람직하지 않다. 다만, 그러한 옵션은 이미 오래 전에 중첩된 주석을 사용해 작성된 프로그램을 다룰 필요가 있을 때 실질적인 편의를 위해서만 사용하는 것이 좋다.

10. (출력 결과 생략)

gcc (2.95.3) 는 소스 차원에서 개행문자를 포함하는 문자 상수는 문자열 상수와는 달리 잘못된 프로그램으로 인식하여 번역을 거절한다 – "unterminated character constant" 라는 진단 메시지를 출력한다. 다문자 상수의 경우에는 "warning: multi-character character constant" 라는 경고 차원의 진단 메시지와 함께 해당 다문자 상수에 대해 이식성 없는 값을 정해준다.

11. 각 전처리기 숫자를 프로그램 소스에 삽입해 필자가 사용중인 임플리멘테이션에서 실행한 결과는 다음과 같다.

(1) 1Ex

[1][Ex] 를 의도하고, Ex 가 매크로 명칭으로 인식되어 +1 로 치환된 후 결과 2 를 의도했겠지만, 전체가 전처리기 숫자로 인식되어 "error: invalid floating constant" 라는 진단 메시지가 생성되고 프로그램의 번역은 거절된다.

(2) 0xEE+23

[0xEE][+][23] (결과는 261) 를 의도했겠지만, 전체가 전처리기 숫자로 인식되어 "error: extra text after expected end of number" 라는 진단 메시지가 생성되고 프로그램의 번역은 거절된다.

(3) 0x7E+macro

[0x7E][+][macro] 를 의도하고, macro 가 매크로 명칭으로 인식되어 0 으로 치환된 후 결과 126 을 의도했겠지만, 전체가 전처리기 숫자로 인식되어 진단 메시지와 함께 프로그램의 번역은 거절된다.

(4) 0x100E+value+macro

[0x100E][+][value][+][macro] 를 의도하고, value 와 macro 가 모두 0 으로 치환된 후 결과 4110 을 의도했겠지만, 전체가 전처리기 숫자로 인식되어 진단 메시지와 함께 프로그램의 번역은 거절된다.

참고로, gcc 는 다소 이상한 진단 메시지들과 함께 "missing white space after number `1E'" 같은 상당히 바람직한 (직접적인 해결책을 제시하는) 진단 메시지를 출력해 주었다.

12. (1)

```
[구분자 #] [명칭 include] [헤더명 <stdio.h>]

[명칭 int] [명칭 main] [구분자 (] [명칭 void] [구분자 )]
[구분자 {]
[명칭 int] [명칭 fool] [구분자 ;]

[명칭 printf] [구분자 (] [문자열 상수 "%d\n"] [구분자 ,] [명칭 foo]
    [구분자 )] [구분자 ;]

[명칭 return] [전처리기 숫자 0] [구분자 ;]
[구분자 }]
```

(2)

```
[#] [include] [<stdio.h>]

[명칭 int] [명칭 a] [구분자 [] [구분자 ]] [구분자 ;]

[int] [main] [(] [void] [)]
[{]
[printf] [(] ["%d\n"] [,] [명칭 a] [구분자 [] [전처리기 숫자 0] [구분자 ]]
    [구분자 )] [구분자 ;]

[return] [0] [;]
[}]
```

(3)

```
[1번 줄부터 5번 줄까지 생략]
[명칭 union] [구분자 {]
[명칭 unsigned] [명칭 long] [명칭 int] [명칭 li] [구분자 ;]
[명칭 unsigned] [명칭 char] [명칭 uc] [구분자 []] [명칭 sizeof] [구분자 (]
    [명칭 long] [명칭 int] [구분자 )] [구분자 ]] [구분자 ;]
[구분자 }] [명칭 u] [구분자 =] [구분자 {] [전처리기 숫자 1] [구분자 }] [구분자 ;]
```

```
[명칭 if] [구분자 (] [명칭 u] [구분자 .] [명칭 uc] [구분자 []] [전처리기 숫자 0]
    [구분자 ]] [구분자 ==] [전처리기 숫자 1] [구분자 )] [명칭 puts] [구분자 (]
    [문자열 상수 "little-endian"] [구분자 )] [구분자 ;]
[명칭 else] [if] [(] [u] [.] [uc] [[] [sizeof] [(] [long] [int] [)] [구분자 -]
    [전처리기 숫자 1] []] [==] [1] [)] [puts] [(] ["big-endian"] [)] [;]
[else] [puts] [(] ["unknown endian"] [)] [;]
[14번 줄부터 마지막까지 생략]
```

(4)

```
[1번 줄부터 4번 줄까지 생략]
[명칭 double] [명칭 x] [구분자 ;]

[명칭 x] [구분자 =] [전처리기 숫자 1.0] [구분자 /] [전처리기 숫자 3.0] [구분자 ;]
[if] [(] [x] [==] [1.0] [/] [3.0] [)] [puts] [(] ["equal"] [)] [;]
[else] [puts] [(] ["different"] [)] [;]
[10번 줄부터 12번 줄까지 생략]
```

이 답안에서,

- (전처리기 지시자가 있는 줄을 제외하고) 줄바꿈을 한 이유는 독자 여러분의 편의를 위해 원래 예제와 가능한 유사한 형태를 유지하기 위함이며 (토큰의 열이기 때문에 줄바꿈이 필수적으로 필요하지는 않음),
- 들여쓰기 된 줄은 원래의 예제에서 윗줄과 이어지는 한 줄임을 의미하고,
- 이미 해당 토큰의 이름 (“명칭”, “구분자” 등) 을 충분히 보인 토큰은 편의를 위해 [] 안에 토큰 이름을 생략하였다.

(5)

```
[#] [include] [<stdio.h>]

[int] [main] [(] [void] [)]
[{]
[명칭 int] [명칭 forwhile] [구분자 =] [전처리기 숫자 12] [구분자 ;]
[명칭 char] [구분자 *] [명칭 str1] [구분자 =]
```

```
    [문자열 상수 "A character constant has the form 'c'."] [구분자 ;]
[명칭 char] [구분자 *] [명칭 str2] [구분자 =]
    [문자열 상수 "A comment has the form /* ... "] [구분자 *] [구분자 /] [;]
[어디에도 속하지 않는 부류 "] [구분자 /] [구분자 *]
[명칭 another] [명칭 comment] [구분자 ?] [구분자 *] [구분자 /]
    [어디에도 속하지 않는 부류 "]

[int] [number] [=] [전처리기 숫자 0x12FE+forwhile+if.not] [;]
[}]
```

강조된 부분은 특별히 주의해야 하는 부분을 보여준다. 우선, str1 의 초기화에 사용되는 문자열 상수 "A character constant has the form 'c'." 에 대해서는 164 쪽에서 충분히 설명하였으므로 큰 문제 없이 이해할 수 있으리라 믿는다.

str2 의 초기화에 사용된 문자열 상수가 끝난 이후 남은 (마치 주석을 닫는 것처럼 생긴) */ 는 각각 두 개의 구분자로 토큰화된다 – 물론, 이는 문법 위반에 해당하여 해당 프로그램을 잘못된 것으로 만든다.

그 다음 줄은 /* A string literal has the form "... */ 까지가 주석으로 인식되어 공백으로 치환된다. 따라서, 남아있는 부분은 다음과 같은 형태를 갖게 되고,

```
"/*
another comment? */"
```

이는 사실상 소스 차원에서 개행문자를 문자열 상수에 직접 포함시킨 형태가 되어 잘못된 프로그램이 된다. 이 잘못된 프로그램이 토큰으로 인식되는 다소 까다로운 과정을 위의 답안이 보여준다 (어렵다면 굳이 이해할 필요는 없다). 즉, 언어의 정의를 따르면 문자열 상수는 소스에서 직접 개행문자를 포함할 수 없도록 정의되어있기 때문에 각 줄에 있는 " 는 독특한 전처리기 토큰인 “어디에도 속하지 않는 부류” 로 인식된다 (정상적인 구분자 토큰에는 " 가 없음에 유의하자). 표준은 문자 " 가 “어디에도 속하지 않는 부류” 로 인식되는 경우의 행동을 정의되지 않는다고 명시하고 있다. 즉, 결국은 위와 같이 소스 차원에서 개행문자를 문자열 상수에 포함시키는 행동이 잘못된 것임을 이야기하는 것이지만 특이한 전처리기 토큰인 “어디에도 속하지 않는 부류” 를 사용해 그 과정을 기술하고 있는 셈이다.

마지막으로 0x12FE+forwhile+if.not 은 [전처리기 숫자 0x12FE][구분자 +][명칭 forwhile][구분자 +][명칭 if][구분자 .][명칭 not] 이 아닌 하나의 전처리기 숫자로 인식됨에 (즉, 잘못된 것임에) 유의하자.