Chapter 2. 어휘, 원소, 연산자, C 시스템

연습문제

2-1. main은 키워드 인가? 설명하여라.

Sol) main은 함수이름으로서 식별자이다.

2-2. 다섯 개의 키워드를 쓰고, 그들의 사용법을 설명하여라.

Sol)

int : 정수형임을 알린다. 32bit integer 형의 경우 int x 는 $-2\times10^{9} \le x \le +x\times10^{9}$ 까지의 수를 나타낼 수 있다.

float : 실수형임을 알린다. 32bit real 형의 경우 float x 는 $-9\times10^{-38} \le x \le +9\times10^{38}$ 의 수를 나타낼 수 있다.

return : 어떤 프로그램에서 서브 함수로 들어 갔다가 나오면서 복귀하는 값을 지정해 주는 키워드이다.

signed : 어떤 상수가 부호를 가짐을 알린다.

unsigned : 어떤 상수가 부호를 가지지 않는 수임을 알린다.

2-3. 토큰의 여섯가지 유형에 대한 예제를 두개씩 들어라.

Sol) C 프로그램의 문자의 단어 하나하나가 언어의 기본이라고 생각될 수 있는 토큰으로 분류되어 해석 된다.

C에는 6가지의 토큰이 있다. 키워드, 식별자, 상수, 문자열 상수, 연산자, 구두점.
키워드에는 C언어에서 고유한 의미를 가지는 예약된 토큰으로 다시 정의되거나 사용 될 수 없다. Auto, do, goto, if, int, else, for, case, char 등등이 키워드로 분류 된다.
다음은 식별자로 함수 이름이나 변수들의 이름이다. 예를 들어 main이나 자기가 정한 상수들..
상수는 0,17같은 정수 상수와 'a' 'b' 와 같은 문자상수가 있다.
문자열 상수는 한 쌍의 큰 따옴표에 묶인 일련의 문자들로 "a string of text"같은 문자들을 말한다. 연산자와 구두점은 + - / * 와 같은 연산자와 ( ) { , ; 와 같은 구두점들이 있다.

2-4. 다음 중 사용할 수 없는 식별자를 고르고 그 이유를 설명하여라.

3id __yes o_no_o_no 00_go star*it 1_i_am one_i_aren"t me_to –2 xYshouldI int

Sol) 식별자는 다음과 같은 규칙을 가지고 있다.

indentifier ::= {letter|underscore}$_1${letter|underscore|digit}$_{0+}$

underscore ::= _

letter ::= a|b|c|d| .... |z|A|B|C|....|Z

digit ::= 0|1|2|3|4|5|6|7|8|9

따라서 이 규칙을 위배하는 식별자는 3id_yes, 00_go, 1_i_am, one_i_aren"t me_to-2 이다.

또한 키워드 역시 식별자로 쓸수 없으므로 int도 사용할 수 없다.

2-5. 프로그램을 누가 작성했고, 왜 작성했는지에 대한 정보를 줄 수 있는 표준화된 주석을 만들어 보아라.

Sol) 생략

2-6. + 같은 기호를 선택한 후 프로그램에서 사용되는 여러가지 방법들을 설명하여라.

Sol) + - * / %

이와 같은 연산자는 각각 덧셈, 뺄셈, 곱셈, 나눗셈, 나머지 등의 일반적인 수학 연산을 뜻한다. 하지만 *는 포인터를 나타낼 때 %역시 다른 용도로 쓰이기도 한다.

2-7. ANSI C는 중첩 주석을 허용하지 않지만, 어떤 컴파일러는 이를 허용하기도 한다. 다음 행을 컴파일했을 때 어떤 일이 일어나는지를 실험해 보아라.

/* This is an attempt /* to nest */ a comment. */

```
[romance@161s ch2_code]$ cc -Wall 2_7.c
2_7.c:1:25: warning: "/*" within comment
2_7.c:1: parse error before `comment'
[romance@161s ch2_code]$
```

2-8. ANSI C 위원회는 C언어에 C++ 주석을 추가하는 것을 고려하고 있고, 이미 어떤 컴파일러에서는 이것을 허용하고 있다. 다음 행이 들어가는 프로그램을 컴파일 해 보아라.

// Will this c++ style comment work in C?

```
[romance@161s ch2_code]$ cc 2_8.c
[romance@161s ch2_code]$                    (에러 없음)
```

2-9. 파운드와 온스를 킬로그램과 그램으로 변환하는 상호 작동하는 프로그램을 작성하여라. main() 함수 위에 정의 되는 기호 상수를 이용하여라.

C code

```
#include <stdio.h>

const double ozforkg =   0.02835;
const double lbforkg =   0.45359;

int main (void)
{
   double a;
   printf(" Input weight of Oz   ");
   scanf("%lf",&a);
   printf("₩n%lf [Oz] is %lf [Kg], %lf [g]",a,a*ozforkg,a*ozforkg*1000);
   printf("₩n Input weight of lb   ");
   scanf("%lf",&a);
   printf("₩n%lf [lb] is %lf [Kg], %lf [g] ₩n₩n",a,a*lbforkg,a*lbforkg*1000);
   return 0;
}
```

Result (결과)

```
[romance@161s ch2_code]$ ./a.out
 Input weight of Oz   10                    (Oz는 온스)

10.000000 [Oz] is 0.283500 [Kg], 283.500000 [g]
 Input weight of lb   20                    (lb는 파운드)

20.000000 [lb] is 9.071800 [Kg], 9071.800000 [g]

[romance@161s ch2_code]$
```

2-10. 이 연습문제는 연산자 주위의 공백의 위치가 얼마나 중요한 지를 보는 것이다. 수식 a+++b는 +기호의 묶여지는 방법에 따라 a++ + b 또는 a + ++b로 해석될 수 있다. 첫번째와 같이 묶여지는 것이 정확한 것이다. 그 이유는 컴파일러가 가장 긴 문자열을 먼저 토큰으로 묶기 때문이다. 따라서 첫번째 토큰은 +가 아닌 ++가 된다. 실제로 수행되는지 프로그램을 작성하여 확인해 보아라.

C code

```
#include <stdio.h>
int main(void)
{
 int a = 1;
 int b = 2;

 if ( a+++b == (a++)+b) {printf ("OK!₩n"); }
 else printf ("NO!₩n");
 return 0;
}
```

Result (결과)

```
[romance@161s ch2_code]$ ./a.out
OK!
[romance@161s ch2_code]
```

2-11. 2.12절의 pow_of_2 프로그램에서 ++i 수식을 i++로 바꾸면 어떻게 되겠는가?

C code

```
#include <stdio.h>
int main(void)
{
 int i=0,power = 1;
 while(++i<=10)
     printf("%-6d",power *=2);
 printf("₩n");
 i = 0;
 power = 1;
 while(i++<=10)
     printf("%-6d",power *=2);
 printf("₩n");


 return 0;
}
```

Result (결과)

```
[romance@161s ch2_code]$ ./a.out
2      4      8      16     32     64     128    256    512    1024
2      4      8      16     32     64     128    256    512    1024   2048
[romance@161s ch2_code]$
```

2-12. 다음 코드는 pow_of_2 프로그램에 사용 할 수 있는 다른 방법이다. 출력될 결과를 쓰고, 프로그램을 완성하여 수행한 후 답을 비교해 보아라.

```
int power = 2048;
while ((power/=2) >0)
    printf("%-6d",power);
```

```
[romance@161s ch2_code]$ ./a.out
1024   512    256    128    64     32     16     8      4      2      1
[romance@161s ch2_code]$
```

2-13. 연습문제 2-12번에서 작성한 프로그램은 while loop를 사용한다. While 루프 대신 for 루프를 사용하여 같은 일을 수행하는 프로그램을 작성하여라.

C code

```
#include <stdio.h>
int main(void)
{
 int i=0;
 int power = 2048;
for (i=0;(power/=2)>0;)
 printf("%-6d",power);

printf ("₩n");

 return 0;
}
```

2-14. 다음 코드를 보고, 출력될 값들을 적어보아라. 그 다음 프로그램을 작성하여 수행한 후 답을 비교해 보아라

.

```
 int a,b = 0, c= 0;
 a = + + b +  + + c;
 printf("%d %d %d₩n",a,b,c);
 a = b+ + + c+ + ;
```

```
printf("%d %d %d₩n",a,b,c);
a = + + b + c + + ;
printf("%d %d %d₩na,b,c);
a = b-- + --c;
printf("%d %d %d₩na,b,c);
```

```
[romance@161s ch2_code]$ ./a.out
2 1 1
2 2 2
5 3 3
5 2 2
[romance@161s ch2_code]$
```

2-15. 다음 문장에서 괄호를 부분적으로 또는 모두 제거 할 경우, 어떤 결과가 초래 될 것인지 설명하여라.

x= (y =2) + (z = 3);

Sol) 위의 문장에서는 x = y+ z; 이고 y = 2; z =3; 인 셈이다.
하지만 위의 문장에서 괄호를 하나씩 제거를 해보면 다른 결과를 얻을 것이다.
만약 x=y=2+ (z=3);
의 경우는 x = y = 2 + z; z = 3;를 계산하는 셈이고
x=y=2=z=3; 이라는 것은 x=y=z=2=3; 이므로 에러가 날 것이다.

2-16. 먼저 다음 표의 빈공간을 채워라. 그 다음 프로그램을 작성하여 답을 확인해 보아라. 그리고 표의 마지막 행의 수식이 왜 오류인지를 설명하여라. 이 오류는 컴파일할 때 발생하는 가 아니면 실행중에 발생하겠는가?

선언 초기화		
int a = 2, b = -3, c = 5, d = -7, e = 11;		
수식	동일한 수식	결과값
a/b/c	(a/b) /c	0
7+ c*--d/e	7+ ((c*(--d)) / e)	4
2*a%-b+ c+ 1	(2*(a%(-b)))+ c+ 1	10
a + = b+ =c + = 1 + 2	a+ =(b + =(c= + = (1+ 2)))	7

7 - + + + a%(3+ b)	7 – ((+ + (+ a))%(3+ b)))	Error

2-17. 다음 코드를 보자.

```
int a = 1, b=2, c=3;
a + = b + = c + = 7;
```

이 문장을 완전한 괄호로 표현된 동일한 문장으로 기술해 보아라. a,b,c 변수의 값은 어떻게 되겠는가? 생각하는 답을 먼저 적고, 프로그램을 작성하여 확인해 보아라.

Sol) int a =1,b=2,c=3;

c = c + 7;

b = b + c;

a = a + b;

따라서 c = 10 , b= 12 , a = 13 이 나올 것이다.

```
[romance@161s ch2_code]$ ./a.out
a=13 b=12 c=10
[romance@161s ch2_code]$
```

2-18. 컴파일러는 프로그램을 단지 문자의 스트림으로 보지만, 프로그램의 윤곽을 좋게 하는 것이 사람이 프로그램을 쉽게 읽을 수 있도록 하기 때문에 좋은 프로그래밍 기법이다. 다음 프로그램을 보자.

```
int main(void
) {float qx,
zz,
tt; printf(“gimme 3”
); scanf
( “%f%f  %f”,&qx,&zz
,&tt); printf(“averageis= %f”,
(qz+ tt+ zz)/3.0 ); return
0
;}
```

이 코드는 읽기 어려워도, 컴파일 되고 실행된다. 이 프로그램을 작성하고 수행하여

잘 수행되는 지를 확인해 보아라. 이제 공백과 주석을 사용하여 읽기 쉽고 문서화가 잘된 프로그램으로 다시 작성해 보아라.

Sol)

```
#include <stdio.h>

int main(void)
{
float x,y,z;
printf("gimme 3");
 scanf("%f %f   %f",&x,&y,&z);
 printf("Average is= %f",(x+ y+ z)/3.0 );
return 0;
}
```

2-19. 2.13절의 prn_rand 프로그램을 for 루프 대신 다음과 같은 while 루프를 사용하도록 다시 작성하여라.

```
While (i++<n) {
      .......
}
```

프로그램을 실행하고 그 결과를 이해한 후, i++ <n 을 ++i < n으로 바꾼 프로그램을 다시 작성하여라. 이제 프로그램은 다르게 작동할 것 이다. While루프의 몸체를 다시 작성하여 이전 결과와 같은 결과가 나오도록 하여라.

```
    a) while(i++<n) {....} 로 고친 코드

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  int i,n;
  printf("₩n%s₩n%s","Some randomly distributed integers will be printed.",
        "How many do you want to see?      ");
  scanf("%d",&n);
 i = 0;
```

```
  while (i++ < n)
  {
   if (i % 8 ==0)   putchar('₩n');
   printf(" %7d ",rand());
  }
  printf("₩n₩n");
  return 0;
}
```

b) ++i 와 i++ 의 결과 차이

(i++)의 경우

```
[romance@161s ch2_code]$ ./a.out

Some randomly distributed integers will be printed.
How many do you want to see?      10
 1804289383    846930886    1681692777    1714636915    1957747793
424238335   719885386
 1649760492   596516649   1189641421          (10개 출력)

[romance@161s ch2_code]$
```

(++i)의 경우

```
[romance@161s ch2_code]$ ./a.out

Some randomly distributed integers will be printed.
How many do you want to see?      10
 1804289383      846930886      1681692777      1714636915      1957747793
424238335   719885386
 1649760492   596516649                  (9개 출력)

[romance@161s ch2_code]$
```

c) ++i 의 코드를 고침.

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

```
int main(void)
{
   int i,n;
   printf("₩n%s₩n%s","Some randomly distributed integers will be printed.",
          "How many do you want to see?       ");
   scanf("%d",&n);
 i = 0;
 while (+ + i < n+ 1)
   {
    if (i % 8 ==0)   putchar('₩n');
    printf(" %7d ",rand());
   }
   printf("₩n₩n");
   return 0;
}
```

2-20. rand() 함수에 의해 생성되는 정수들은 모두 [0,n] 구간 내에 들어간다. 여기서 n은 시스템 종속적이다. ANSI C의 n값은 표준 헤더 파일 stdlib.h 에 정의된 기호 상수 RAND_MAX에 의해 주어진다. 사용자 시스템에서 사용된 RAND_MAX의 값을 출력하는 프로그램을 작성하여라.

```
[romance@161s ch2_code]$ ./a.out
RAND_MAX = 2147483647
```

2-21. 2.13절의 prn_rand 프로그램을 세번 실행시켜 각각 100개의 난수를 출력하여라. 각 실행때마다 동일한 숫자들이 출력되는 살펴보아라. 많은 응용 프로그램에서 이러한 결과는 바람직한 것이 아니다. 난수 발생기에 시드를 주는 srand()를 사용하도록 prn_rand 프로그램을 수정하여라. 프로그램의 첫부분을 다음과 같이 작성하여라.

```
[romance@161s ch2_code]$ ./a.out
Some randomly distributed integers will be printed.
How many do you want to see?      5

 1804289383   846930886   1681692777   1714636915   1957747793

[romance@161s ch2_code]$ ./a.out
Some randomly distributed integers will be printed.
How many do you want to see?      5

 1804289383   846930886   1681692777   1714636915   1957747793

[romance@161s ch2_code]$
```

두번의 실행으로 모두 같은 값을 출력하고 있다.
이와 같은 결과를 수정하기 위해 seed값을 사용하는
srand() 함수를 사용해서 다시 결과를 출력하면 다음과 같다.

```
[romance@161s ch2_code]$ ./a.out
Some randomly distributed integers will be printed.
How many do you want to see?      5

 2133853005   1588391111   1323813279   2073956772   1557381609

[romance@161s ch2_code]$ ./a.out
Some randomly distributed integers will be printed.
How many do you want to see?      5

 142047438   1022718592   1115902234   907855712   1125881725

[romance@161s ch2_code]$
```

이와 같이 다른 값을 출력한다.

2-22. 앞의 연습문제에서는 다음과 같은 코드를 제안하였다.

seed = time(NULL);
srand(seed);

대부분의 프로그래머는 다음과 같은 코드를 대신 사용한다.
Srand(time(NULL));
앞의 프로그램을 이와 같이 수정한 다음, 컴파일 하고, 실행시켜 보아라.
이전의 프로그램과 똑같이 동작하는가?

Sol) 같은 코드로서 똑같이 동작한다.

2-23. 앞의 두 연습문제에서는 난수 발생기에 시드를 주기 위해 time()이 리턴하는 값을 사용하였다. 이번 연습문제에서는 time()의 리턴값을 사용하여 rand()함수를 호출하는데 걸리는 시간을 측정한다. 다음 프로그램은 이를 위한 한가지 방법을 보여주고 있다.

C code

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define NCALLS 10000000
#define NCOLS 8
#define NLINES 3

int main (void)
{
  int i,val;
  long begin,diff,end;
  begin = time (NULL);
  srand(time(NULL));
  printf("TIMING TEST : %d calls to rand()₩n",NCALLS);
  for (i = 1; i <= NCALLS; + + i)
  { val = rand();
```

```
      if ( i<=NCOLS * NLINES) { printf ("%7d",val);
      if (i % NCOLS == 0) putchar('₩n');
                                        }
   else if ( i == NCOLS * NLINES +  1)
    printf ("%7s₩n₩n","......");
   }
end = time(NULL);
diff = end - begin;
printf ("%s%1d₩n%s%ld₩n%s%ld₩n%s%.10f₩n₩n","        end time : ", end,
 "     begin time : ",begin, " elapsed time : ",diff,"time for each call : "
, (double) diff / NCALLS);
 return 0;
}
```

Result (결과)

```
[romance@161s ch2_code]$ ./a.out
TIMING TEST : 10000000 calls to rand()
16788145088178676510941596901933102831595975256465237146657927708106577121 6
16772692133714668101823351534607709729449210650126235453321189959194540970 21
73140262152315555317316414891635818506129391268812896736881136425895730474942
 ......

       end time : 1066994221
    begin time : 1066994220
 elapsed time : 1
time for each call : 0.0000001000

[romance@161s ch2_code]$
```

Result 2::

코드 중에 NCALLS 10000 NCOLS 7 NLINES 7 로 바꿨을 경우

```
[romance@161s ch2_code]$ ./a.out
TIMING TEST : 10000 calls to rand()
14123979481962764170542068480812029642721231517211911157611212063 38
16376538601910944701111127458112133988941102380749183116900154745 1656
```

```
197884387811937046541277270314174294886119743910621257122101886080436
148983327149780637717509151289671084580502373649568020472552801
172665846114287894044622315899157276121056431109882916381803602403
679390979959919566777325093169561191723380619188859967513829600 85
182576136820717165757829280931657121598111793758220601984081831416484
 ......

        end time : 1066994348
      begin time : 1066994348
   elapsed time : 0
  time for each call : 0.0000000000

  [romance@161s ch2_code]$
```

2-24. rand() 함수는 [0,RAND_MAX] 구간의 값을 리턴한다. Double 형의 변수 median 을 선언하고 초기값을 RAND_MAX / 2.0으로 하여라. 연속해서 rand()를 호출할 때 어떨때에는 median보다 큰 값을 어떨 경우에는 작은 값을 리턴할 것이다. 평균적으로 median보다 클 경우와 작을 경우의 확률은 같은 것이다.

C code

```
#include <stdio.h>
#include <stdlib.h>
#define MEDIAN   RAND_MAX/2.0
int main(void)
{
 int i,above_cnt,below_cnt;
 above_cnt = 0;
 below_cnt = 0;
  for ( i=0;i<1000000;i++ )
  {
    if (rand() >= MEDIAN ) {++above_cnt; }
    else { ++below_cnt; }
     if (i % 10000 == 0) { printf ("%7d %7d %7d₩n",above_cnt,below_cnt,above_cnt -
below_cnt);}
```

```
  }
  return 0; }
```

2-25. 이 연습문제는 연습문제 20번에 이어지는 것이다. Rand()를 호출하면, [0,RAND_MAX] 구간의 값을 생성하고, RAND_MAX는 일반적으로 32767 값을 가진다. 이값은 이렇게 큰 값이 아니기 때문에, 많은 과학 문제에 적합하진 않다. UNIX 컴퓨터 상의 대부분의 C 시스템은 rand48 계열의 함수 발생기를 제공한다. Rand48은 숫자를 생성하는데 48bit 계산을 이용함으로써 붙여진 이름이다. 예를 들어 drand48() 함수는 [0,1] 범위 내에 무작위로 분산된 double 값을 생성하기 위해 사용 될 수 있다. 그리고 lrand48()은 $[0,2^{31}-1]$ 구간에서 무작위로 분산된 정수값을 생성하기 위해 사용될 수 있다. 연습문제 20번의 프로그램에서 rand()를 land48()로 대치하고 srand()를 srand48()로 대치하여 프로그램을 다시 작성하여라 그러면 평균적으로 더 큰 숫자가 생성되는 것을 볼 수 있다.

```
[romance@161s ch2_code]$ ./a.out
RAND_MAX = 2147483647

Some randomly distributed integers will be printed.
How many do you want to see?      10

       0   2116118   89401895   379337186   782977366   196130996   198207689
1046291021
 1131187612   975888346

[romance@161s ch2_code]$
```

2-26. ++a + a++와 a+=++a 같은 수식의 값은 시스템에 종속적이다. 왜냐하면 증가 연산자 ++의 부작용이 각각 다른 시점에서 일어날 수 있기 때문이다. 이것은 C의 장점이지만 동시에 약점이기도 하다. 한쪽 면으로 보면, 컴파일러는 기계 수준에서 가장 적합한 것을 할 수 있다. 그리고 다른 면으로 보면, 수식이 시스템 종속적이기 때문에 서로 다른 컴퓨터에서 다른 값을 갖게 될 것이다. 경험이 많은 C 프로그래머들은 이와 같은 수식이 잠재적인 위험성을 내포하고 있음을 알기 때문에 이러한 표현은 사용하지 않는다. A를 0으로 초기화 한 후 ++a + a++이 어떤 값을 갖는지 각자 컴퓨터에서 실험해 보아라.

[romance@161s ch2_code]$ ./a.out

2
[romance@161s ch2_code]$

2-27. UNIX 시스템에서 라이브러리는 “archive”를 의미하는 .a로 끝나고, Windows 98 NT 시스템에서는 .lib로 끝난다. 다음과 같은 명령어를 입력하면, 라이브러리에 포함된 모든 파일의 이름을 볼 수 있을 것이다.
ar t /usr/lib/libc.a

```
.....( 생략)
dl-conflict.o
dl-open.o
dl-close.o
dl-support.o
dl-iteratephdr.o
dl-iteratephdr-static.o
dl-addr.o
enbl-secure.o
dl-profstub.o
dl-origin.o
dl-libc.o
dl-sym.o
dl-procinfo.o
[romance@161s ch2_code]$
```

2-28. ANSI C와 전통적인 C에서, 문자열 상수의 마지막에 ₩가 오면, 문자열이 다음행으로 이어진다는 것을 나타낸다.
“by using a backslash at the end of the line ₩
a string can be extended from one line to the next”

이것을 포함하는 프로그램을 작성하여라. 보통화면의 한행에는 80개의 문자를 쓸수 있다. 만일 80자 이상의 문자열을 출력하면 어떤 일이 발생하겠는가?

C code

```
#include <stdio.h>
int main(void)
{
   printf("by using a backslash at the end of the line ₩
a string can be extended from one line to the next"
);
return 0;
}
```

80자 이상 문자열을 출력할 수 있다.

2-29. ANSI C에서 각 행의 끝의 ₩는 이 행을 다음 행으로 계속 연결시켜 주는 효과를 낸다. 이러한 ₩의 효과는 문자열 상수와 매크로 정의에서 사용할 수 있다. 그러나, 모든 ANSI C 컴파일러가 일반적인 방법을 제공하는 것은 아니다. 결국, 매크로 정의를 제외하면, 이 방법은 거의 사용되지 않는다. 사용자의 C 컴파일러는 일반적인 방법으로 이것을 제공하는지 다음 프로그램을 실행하여 보아라.

```
#inc₩
lude<stdio.h>
int mai₩
n(void)
{ printf("Will this Word?₩n");
ret₩
urn 0;
}
```

Sol)

```
[romance@161s ch2_code]$ cc 2_29.c
2_29.c:1:10: invalid preprocessing directive #inc          (에러 발생)
[romance@161s ch2_code]$
```

2-30. 컴파일러를 호출하면, 시스템은 먼저 전처리계를 호출한다. 이 연습문제에서는 어떤 일이 일어나는지 알아보기 위해 일부러 전처리 오류가 발생하도록 하였다.
다음 프로그램을 실행하여 보아라.

```
#incl <stdixx.h>
int main(void)
{ printf("Try me.₩n");
return 0;
}
```

a) 위 코드를 실행시킨후 결과

```
[romance@161s ch2_code]$ cc -Wall 2_30.c
2_30.c:1:2: invalid preprocessing directive #incl
[romance@161s ch2_code]$
```

b) #incl를 #include로 수정시킨 후 결과

```
[romance@161s ch2_code]$ cc 2_30_a.c
2_30_a.c:1:20: stdixx.h: No such file or directory
[romance@161s ch2_code]$
```

Chapter 2 End Point.