연습문제

8-1. 인자를 갖는 매크로를 사용하는 프로그램은 디버깅하기가 어렵다. 대부분의 C 컴파일러는 프로그램을 컴파일하지 않고 전처리기의 결과만 화면에 출력하는 옵션을 제공한다. 다음과 같은 코드 try_me.c 에 작성하여라.

```
- #include <stdio.h>
- #define PRN(x) printf("x₩n");
- int main(void)
- {
-    PRN(hello from main());
-    return 0;
- }
```

이 프로그램을 컴파일한 다음 실행시켜 보아라. 예상한대로 출력되지 않음을 알 수 있을 것이다. 전처리기가 이 코드를 어떻게 처리하는지 알기 위해 다음 명령을 사용해 보아라.

- cc –E try_me.c

식별자 PRN은 전처리기에 의해 생성되지 않음을 볼 수 있을 것이다. 그 이유를 설명하여라. 코드를 수정하여 원하는 것이 출력되도록 하여라.

Sol)

```
…..
# 2 "8_1.c" 2

int main(void)
{
   printf("x₩n");;
return 0;
}
```

-E 옵션은 전처리기만 호출하기 때문에 전처리기의 처리 과정만 보여주고 본문부분은 남겨지게 된다.

8-2. 다음과 같은 매크로 정의를 생각해 보자.

- #define forever(x) forever(forever(x))
- forever(more)

이 코드는 무한적인 재귀를 생성할 것 처럼 보이지만, ANSI C의 전처리기는 무한 재귀를 의도한 것이 아님을 알 수 있을 만큼 지능적이다. 이 매크로가 어떻게 확장되는 지 확인해 보아라.

Sol)

-E 옵션으로 컴파일 후

```
……..
# 2 "8_2.c" 2

int main(void)
{
forever(forever(more));

return 0;
}
```

8-3. 프로그램에서 x,y,z는 float 형 변수라고 가정하자. 이 변수들이 각각 1.1,2.2, 3.3을 가진다고 하고 다음 문장을 보자.

- PRN3(x,y,z);

이것이 다음과 같은 출력을 내도록 PRN3() 매크로를 정의하여라.

- x has value 1.1 and y has value 2.2 and z has value 3.3

Sol)

C code

```
#include <stdio.h>
#define x1 1.1
```

```
#define y1 2.2
#define z1 3.3
#define PRN3(x,y,z)                printf("x has value "x" , y has value "y"
, z has value "z" ₩n");
int main(void)
{
PRN3(x1,y1,z1)
return 0;
}
```

Result (결과)

```
…..
int main(void)
{
printf("x has value "1.1" , y has value "2.2" , z has value "3.3" ₩n");
return 0;
}
[romance@161s ch8_code]$
```

8-4. 다음 코드가 a_b_c.h 파일에 있다고 가정하자.

```
-#define TRUE 1
- #define A_B_C
- int main (void)
- {
-              printf("A Big Cheery ₩"Hello₩"!₩n");
- }
```

그리고 다음 코드가 a_b_c.c 파일에 있다고 가정하자.

```
- #if TRUE
-     #include <stdio.h>
-     #include <a_b_c.h>
-     A_B_C
- #endif
```

이 프로그램을 컴파일하면, 컴파일러는 오류를 발생한다. 그 이유를 설명하여라. 이 두 파일 중에 한 파일에 있는 두 행을 번경하여, 프로그램이 컴파일 되고 실행될 수 있도록 할 수 있는가? 그 이유를 설명하여라.

Sol)

- 8_4.h

```
#define A_B_C                ₩
int main(void)   ₩
{ printf("A Big Cheery ₩"Hello₩"!₩n");₩
  return 0; ₩
}₩
```

- 8_4.c

```
#if 1
  #include <stdio.h>
  #include "8_4.h"
  A_B_C
#endif
```

Result (결과)

```
[romance@161s ch8_code]$ cc 8_4.c
In file included from 8_4.c:3:
8_4.h:2:17: warning: backslash and newline separated by space
[romance@161s ch8_code]$ ./a.out
A Big Cheery "Hello"!
[romance@161s ch8_code]$
```

8-5. 매크로의 몸체에 있는 모든 매개변수가 괄호로 묶여 있다 하더라도 함수만큼 안전하지 못하다. 세인자 중 가장 큰 값을 출력하는 매크로를 정의 하여라.

- MAX(x,y,z)

이때 원하지 않는 결과를 생성하도록 MAX()에서 사용할 수식을 만들어 보아라.

Sol)

C code

```
#include <stdio.h>
#define MAX_m(x,y) ((x)>(y)) ? (x) : (y)
#define MAX(x,y,z) (MAX_m(x,y) > (z)) ? (MAX_m(x,y)) : (z)
int main(void)
{
int x,y,z;
printf("Input x => ");
scanf("%d",&x);
printf("Input y => ");
scanf("%d",&y);
printf("Input z => ");
scanf("%d",&z);
printf("%d₩n",MAX(x,y,z));

return 0;
}
```

8-6. 각자의 시스템에서는 미리 정의된 모든 메크로를 이용할 수 있는가? 다음 코드를 수행해 보아라.

- printf("%s%s₩n%s%s₩n%s%s₩n%s%d₩n%s%s₩n","__DATE__=",__DATE__
- "__FILE__ = ", __FILE__,"__LINE__ =", __LINE__ , "__STDC__ =",__STDC__ ,
- "__TIME__ = ",__TIME__);

Sol)

```
[romance@161s ch8_code]$ ./a.out
__DATE__=Nov 14 2003
__FILE__ = 8_6.c
__LINE__ =6
__STDC__ =1
__TIME__ = 15:56:07
[romance@161s ch8_code]$
```

8-7. 소형 시스템에서 ANSI C 컴파일러를 사용할 수 있는가? 소형 시스템에서, Borland C나 Microsoft C와 같은 컴파일러는 다른 종류의 메모리 모델을 제공하고, 각 메모리 모델은 보통 ptrdiff_t에 대해 특정한 형 정의를 요구한다. Stddef.h를 찾아보고, 여러분의 시스템이 이런 경우에 해당하는지를 알아보아라.

Sol) 생략

8-8. ANSI C에서, 표준 헤더 파일에 있는 인자를 갖는 많은 매크로들은 함수로도 이용될 수 있어야 한다. 각자의 시스템은 이 함수들을 제공하고 있는가? 예를 들면 다음과 같은 코드를 처리할 수 있는 지를 알아보아라.

- #include <ctype.h>
- if ((isalpha)('a'))
- printf("Found the isalpha() function!₩n");

Sol)

```
[romance@161s ch8_code]$ ./a.out
Found the isalpha() function!
[romance@161s ch8_code]$
```

8-9. C는 문자처리를 위해 좋은 언어라는 평가를 받고 있다. 부분적으로, 이러한 평가는 문자 처리에 있어서 함수 보다 매크로가 많이 이용되기 때문이다. 프로그래머는 매크로를 사용하면 실행시간을 상당히 단축할 수 있다고 믿고 있다. 이것은 사실일까?

```
- 원 프로그램
#include <stdio.h>
#include <ctype.h>
#include <time.h>
int main(void)
{
int c;
printf("Clock ticks: %ld₩n",clock());

while ((c = getchar()) != EOF )
  if (islower(c)) putchar(toupper(c));
```

```
   else if (isupper(c)) putchar(tolower(c));

printf("₩nClock ticks: %ld₩n",clock());

return 0;
}

- 매크로를 이용한 프로그램
#include <stdio.h>
#include <ctype.h>
#include <time.h>
#define LO islower
#define TO toupper
#define UP isupper
#define TL tolower
int main(void)
{
int c;
printf("Clock ticks: %ld₩n",clock());
while ((c = getchar()) != EOF )
   if (LO(c)) putchar(TO(c));
   else if (UP(c)) putchar(TL(c));
printf("₩nClock ticks: %ld₩n",clock());
return 0;
}
```

두 프로그램 모두 같은 실행시간인 10000tick을 사용했음을 알 수 있다.
따라서 메크로가 실행시간을 단축시키지는 않는다.

8-10. ANSI C 에서, 표준해터 파일은 반복적으로 포함될 수 있고 또한 어떠한 순서로도 포함될 수 있다. 연습문제 9번에서 작성한 첫 번째 프로그램의 처음 부분을 다음과 같이 #include가 여러 번 중복되도록 수정해 보아라.

- #include <stdio.h>
- #include <ctype.h>
-#include <time.h>

-#include <ctype.h>
-#include <time.h>
-#include <stdio.h>
-int main(void)
- {
-
- return 0;
- }

Sol)

```
[romance@161s ch8_code]$ cc -Wall 8_10.c
[romance@161s ch8_code]$                    (오류발생이 없음)
```

8-11. 8.7절에서 매크로 isascii()는 ctype.h에 있다고 하였다. 그러나 이 매크로는 ANSI C 문서에는 명시되어 있지 않다. 각자의 시스템에서, isascii() 매크로가 제공되는지 알아보아라.

Sol)

C code

```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
int c;
isascii(c);
return 0;
}
```

Result (결과)

```
[romance@161s ch8_code]$ cc 8_11.c
[romance@161s ch8_code]$ ./a.out
[romance@161s ch8_code]$                    (제공됨을 알수 있다.)
```

8-12. 이 연습문제는 ANSI C와 전통적인 C 사이의 미묘한 차이에 대해 다룬다. 전통적인 C에서, tolower()와 toupper()는 ctype.h에서 매크로로서 제공된다. ANSI C에서, 그와 대응되는 매크로가 역시 ctype.h에 있지만 각각 _tolower()와 _toupper()라는 이름으로 되어 있다. 전통적인 C에서는 c가 소문자라는 것을 이미 알고 있을 때에만 toupper(c) 와 같은 수식을 사용할 수 있다. 만일 c가 소문자가 아니라면, toupper(c)는 아무런 효과가 없다. 각자의 시스템에서 직접 실험해 보아라. 다음 수식은 어떤 출력을 내는지 알아보아라.

- tolower('a') _tolower('a') toupper('A') _toupper('A')

Sol)

C code

```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
printf("%c₩n",tolower('a'));
printf("%c₩n",_tolower('a'));
printf("%c₩n",toupper('A'));
printf("%c₩n",_toupper('A'));

return 0;
}
```

Result (결과)

```
[romance@161s ch8_code]$ ./a.out
a
a
A
A
[romance@161s ch8_code]$
```

8-13. 문자열 연산자 #은 매크로로 전달되는 인자를 큰 따옴표로 묶는다. 이때 인자가 이미 큰 따옴표로 묶여 있다면 어떻게 될까? 다음과 같은 코드를 포함하는 프로그램을 작성하여라.

- #define YANK(x) s=#x
- char *s;
- YANK("Go home, Yankee!");
- printf("%s₩n",s);

Sol)

```
-E 옵션 컴파일 후 코드
int main(void)
{
 char *s;
 s="₩"Go home, Yankee!₩"";
 printf("%s₩n",s);

return 0;
}

실행결과
[romance@161s ch8_code]$ ./a.out
"Go home, Yankee!"
[romance@161s ch8_code]$
```

8-14. 다음 코드의 출력을 쓰고, 설명하여라.

- #define GREETINGS(a,b,c) ₩
- printf(#a ", "#b", and "#c ":Hello ₩n")
- int main(void)
- { GREETINGS(Alice, Bob,Carole);
- return 0;
- }

컴파일하기 전에 전처리기가 생성한 내용을 살펴보아라. GREETINGS을 찾을 수 있는가?

Sol)

```
-E 옵션 컴파일
```

```
int main(void)
{
printf("Alice" ", ""Bob"", and ""Carole" ":Hello ₩n");
return 0;
}

출력결과
[romance@161s ch8_code]$ ./a.out
Alice, Bob, and Carole:Hello
[romance@161s ch8_code]$
```

8-15. 다음 지시자를 보자.

#undef TRY_ME

TRY_ME가 이전에 #define 매크로가 정의 되었다면, 이 지시자 때문에 그 매크로 정의는 무효가 된다. TRY_ME가 이전에 정의 되지 않았다면, 이 지시자는 아무런 영향도 미치지 않을 것이다. 각자의 시스템에서는 어떤 일이 발생하는 지를 검사하는 프로그램을 작성하여라. TRY_ME가 이전에 정의되어 있지 않다면, 시스템은 오류를 발생하는가?

Sol) 발생하지 않는다.

8-16. NDEBUG가 정의되어 있으면, 매크로 assert()는 무시된다. 각자의 시스템은 과연 기대한 것처럼 작동하는가? 다음 프로그램을 시험해 보아라.

```
-#define NDEBUG
- #include <assert.h>
- int main(void)
- {
-       int a=1,b=2;
-       assert(a>b);
-        return 0;
- }
```

이 프로그램의 첫 두줄을 서로 바꾸면 어떤일이 발생하는가?

Sol)

기대한 것처럼 assert()는 무시된다.
하지만 위의 첫 두줄의 바꾸게 되면 다음과 같이 assert()가 동작하게 된다.

[romance@161s ch8_code]$ ./a.out
a.out: 8_16_a.c:6: main: Assertion `a>b' failed.
Aborted
[romance@161s ch8_code]$

8-17. 연습문제 16번의 프로그램에서 첫 두줄을 다음과 같은 세 줄로 대치하여라.
- #include <assert.h>
- #define NDEBUG
- #include <assert.h>
C컴파일러는 오류를 발생하는가?
Sol) 오류가 발생하지 않고 assert()는 무시된다.

8-18. 대형 C 프로그램을 4바이트 워드를 갖는 컴퓨터에서 2 바이트 워드를 갖는 컴퓨터로 옮긴다고 가정해 보자. 2 바이트 워드를 갖는 컴퓨터에서 int형의 값은 근사적으로 –32000과 +32000 사이의 값으로 제한한다. 옮길 프로그램의 어느 한 부분에서 이 값의 범위가 너무 좁다고 가정하자. 이 프로그램의 해더 파일에 다음과 같은 지시자문을 삽입하면, 이 문제가 해결되겠는가? 설명하여라.
- #define int long
Sol) 해결된다.

8-19. 해더 파일 stddef.h에서 size_t에 대한 typedef를 찾아 보아라. 또한 stdlib.h 에서도 이 typedef를 찾아보아라. 다음과 같은 행으로 시작되는 프로그램을 작성한다고 가정해 보자.
-#include <stddef.h>
- #include <stdlib.h>
중복된 형 정의는 허용하지 않기 때문에, size_t에 대한 typedef이 두 번씩 포함 되지 못하게 할 수 있어야 한다. 이것이 어떻게 수행되는지 설명하여라.

Sol) 생략

8-20. 만일 qsort()를 사용하고자 한다면, 이 함수의 원형이 무엇인지 알아야 한다. 어떤 ANSI C 시스템에서는 이 함수의 원형이 stdlib.h에 다음과 같이 기술되어있다.

- void qsort(void *a_ptr , size_t n_els size_t el_size,
- int compaer(const void *, const void *));

각자의 시스템이 제공하는 함수 원형은 어떠한가? 이 정의와 동일한가? 함수 원형에서 매개 변수 식별자는 컴파일러에 의해 무시된다. 따라서 다음과 같은 함수 원형도 같은 의미를 갖는다.

- void qsort(void ,size_t , size_t , int(*)(const void *, const void *));

Sol)

NAME

stdlib.h - standard library definitions

SYNOPSIS

#include <stdlib.h>

DESCRIPTION

[CX] ⌦ Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of IEEE Std 1003.1-2001, Section 2.2, The Compilation Environment) to enable the visibility of these symbols in this header. ⌫

.....

void qsort(void *, size_t, size_t, int (*)(const void *,const void *));

8-21. 8.6 절 qsort 프로그램은 두 개의 비교 함수를 사용하였다. 이때 qsort() 함수 원형의 마지막 매개변수와 일치하지 않는 비교 함수를 작성하면 어떻게 되겠는가? 그 두 비교 함수를 다음과 같이 다시 작성하여라.

```
- int lexico(char *p, char *q)
- {
-    return (*p - *q);
- }
- int compare_decimal_part(float *p, float *q)
- {
-    float x;
-    x = decimal_part(*p) – decimal_part(*q);
-    return ((x == 0.0) ? 0 : (x <0.0) ? –1 : + 1);
- }
```

Sol) 생략

8-22. 8.15 절의 quicksort 코드를 검사하는 프로그램을 작성하여라. 프로그램은 다음과 과 같은 코드로 시작해야 한다.

```
- #include <stdio.h>
- #include <stdlib.h>
- #include <time.h>
-
- #define N 10000
- void quicksort(int *,int *);
- int main(void)
- {
-    int a[N],i;
-    srand(time(NULL));
-    for (i = 0; i < N; + + i)
-      a[i] = rand() % 1000;
-    quicksort(a,a+ N-1);
```

정렬된 배열의 원소를 출력할 수 있도록 프로그램을 완성하여라. 만일 모든 원소를

보기가 지루하다면, 그 배열이 정렬되었음을 알 수 있게 그 배열의 첫번째 값, 중간값, 마지막 값만 출력해도 좋다.

Sol)

C code

- 8_22_q.h

```
#define swap(x,y) { int t; t = x;   y = t; }
#define order(x,y) if (x>y) swap(x,y)
#define o2(x,y) order (x,y)
#define o3(x,y,z) o2(x,y); o2(x,z); o2(y,z)

typedef enum {yes,no} yes_no;
static yes_no fine_pivot(int *left, int *right, int *pivot_ptr);
static int *partition (int *left, int *right, int pivot);
```

- 8_22_q.c

```
#include "8_22_q.h"
#include <stdio.h>
#include <stdlib.h>
#define N 10000
void quicksort(int *, int *);

int main(void)
{ int a[N],i;
  srand(time(NULL));
  for(i=0; i < N; ++i)
      a[i] = rand() % 1000;
  quicksort(a,a+N-1);
  printf("1st %d, center %d, last %d",a[0],a[N/2],a[N-1]);
  return 0;
}



void quicksort(int *left, int *right)
```

```
{
 int *p,pivot;
 if (find_pivot(left,right,&pivot) == yes) {
    p = partition(left,right,pivot);
    quicksort(left,p-1);
    quicksort(p,right);
    }
}
static yes_no find_pivot(int *left,int *right, int *pivot_ptr)
{
   int a,b,c,*p;
   a = *left;
   b = *(left +  (right - left)/2);
   c = *right;
   o3(a,b,c);
   if (a < b) {
      *pivot_ptr = b;
      return yes;
}
if (b< c) {
      *pivot_ptr = c;
      return yes;
}
for (p = left +  1; p <=right; ++p)
   if (*p != *left) {
       *pivot_ptr = (*p < *left) ? *left : *p;
      return yes;
}
    return no;
}

static int *partition(int *left,int *right,int pivot)
{
   while(left <= right) {
    while(*left < pivot)
          ++left;
```

```
    while(*right >= pivot)
        --right;
    if (left < right) {
    swap(*left,*right);
    ++left;
            --right;
    }
    } return left;
}

tatic int *partition(int *left,int *right,int pivot)
{
   while(left <= right) {
    while(*left < pivot)
        ++left;
    while(*right >= pivot)
        --right;
    if (left < right) {
    swap(*left,*right);
    ++left;
            --right;
    }
    } return left;
}
```

Result (결과)

```
[romance@161s ch8_code]$ ./a.out
1st 760, center 995, last 999
[romance@161s ch8_code]$ ./a.out
1st 148, center 999, last 999
[romance@161s ch8_code]$ ./a.out
1st 857, center 998, last 999
[romance@161s ch8_code]$ ./a.out
1st 364, center 942, last 999
[romance@161s ch8_code]$
```

8-23. 연습문제 22번에서 각 원소가 구간 [0,999] 사이에서 무작위로 분포된 임의의 정수를 갖는 크기가 10,000인 배열을 정렬해 보았다. 그리고 배열의 실행시간을 측정하면서 이 프로그램을 다시 실행시켜 보아라. 그리고 배열의 원소가 구간[0,9] 사이에 무작위로 분포된 값을 갖도록 프로그램을 수정한 후 다시 수행해 보아라. 왜 시간 차이가 나는지 설명해 보아라.

Sol)

pivot의 선택에 따라 배열의 분포가 0부터 999 보다 0부터 9 까지의 분포가 당연히 중복되는 원소가 많을 확률이 높다. 따라서 이렇게 중복되는 원소가 많을 경우에는 pivot에 따라 원소가 한쪽으로 몰릴 가능성이 많아지고 이는 결국 quicksort를 재귀적으로 더 많이 부를 가능성이 많아 진다는 것이다.

8-24. 퀵 정렬 알고리즘은 정수 배열뿐만 아니라 모든 종류의 배열을 정렬하는 데 사용될 수 있다. Quicksort 코드를 수정하여 문자열 배열을 정렬하도록 하여라. 수정한 코드를 검사하는 프로그램을 작성하여라.

Sol)

C code

8-23의 코드에서 int 부분을 char로 변경하고,
대문자에 대해서 배열을 받으려면 다음과 같이 rand()%26+65 를 이용해서
코드를 받고 quicksort를 이용해 정렬할 수 있다.

```
…..
int main(void)
{ char a[N],i;
   srand(time(NULL));
   for(i=0; i < N; ++i)
       a[i] = rand() % 26 +  65;
   quicksort(a,a+N-1);
   printf("1st %c, center %c, last %c₩n",a[0],a[N/2],a[N-1]);
   return 0;
}
…..
```

Result (결과)

```
[romance@161s ch8_code]$ ./a.out
1st E, center Z, last Z
[romance@161s ch8_code]$ ./a.out
1st F, center Z, last Z
[romance@161s ch8_code]$
```

8-25. 만일 배열이 적은 원소를 갖고 있다면, 예를 들어 7개 이하, 버블 정렬이나 치환 정렬이 퀵정렬보다 빨라야 한다. 다음 quicksort() 버전은 이러한 사실을 고려한 것이다.

```
- int quicksort(int *left,int *right)
- {
-       int *p, *q,pivot;
-       static int cnt = 0;
-    if (right – left    < 7) {
-             for (p = left; p <right; + + p)
-                for(q = p+ 1; q <=right; + + q)
-                       if(*p > *q)
-                             swap(*p,*q);
-           }
-       else if (find_pivot(left,right,&pivot) == yes) {
-              p = partition(left,right,pivot);
-              quicksort(left, p-1);
-              quicksort(p,right);
- }
- return + + cnt;
- }
```

변수 cnt의 사용을 주목하여라. 함수를 호출한 곳으로 리턴되는 값은 함수가 호출된 횟수이다. 이 새로운 quicksort()가 과연 빠르게 실행되는 지를 실험해 보아라. 실행시간과 함수가 호출된 횟수사이에는 상관관계가 있는가?

Sol)

```
int main(int argc,char *argv[])
{
    int a[1000],N,i,cnt = 0;
    N = atoi(argv[1]);
  printf("N = %d₩n",N);
  srand(time(NULL));
  for(i=0; i < N; ++i)
      a[i] = rand() % 1000;
  cnt = quicksort(a,a+N-1);
  printf("1st %d, center %d, last %d, cnt %d ₩n",a[0],a[N/2],a[N-1],cnt);
  return 0;
}
```

위의 코드를 삽입한 후, 다음과 같은 N을 받아서 cnt값과 N값과의 상관관계를 보게 되면,

```
[romance@161s ch8_code]$ ./a.out 1
N = 1
1st 61, center 61, last 61, cnt 1
[romance@161s ch8_code]$ ./a.out 7
N = 7
1st 685, center 685, last 699, cnt 1
[romance@161s ch8_code]$ ./a.out 10
N = 10
1st 529, center 601, last 824, cnt 5
[romance@161s ch8_code]$ ./a.out 20
N = 20
1st 911, center 911, last 970, cnt 21
[romance@161s ch8_code]$ ./a.out 30
N = 30
1st 296, center 863, last 985, cnt 13
[romance@161s ch8_code]$ ./a.out 50
N = 50
1st 704, center 994, last 995, cnt 29
[romance@161s ch8_code]$ ./a.out 100
N = 100
1st 534, center 992, last 995, cnt 53
[romance@161s ch8_code]$ ./a.out 200
```

```
N = 200
1st 896, center 990, last 999, cnt 125
[romance@161s ch8_code]$ ./a.out 400
N = 400
1st 523, center 996, last 996, cnt 213
[romance@161s ch8_code]$
```

1-7까지는 N이 버블정렬에 의해 정렬되고 있기 때문에 cnt가 1이고,
7이상에서는 N의 값의 절반정도로 따라가고 있음을 알 수 있다.
(N=30,cnt = 13) , (50,29), (100,53),(200,124),(400,213) .....

8-26. ~ 8-33. 까지는 quicksort()에 대한 추가 문제이므로, 생략하도록 하겠습니다.

8-34. 연습문제 33번에서, 큰 배열을 사용하여 qsort()와 quicksort()의 수행시간 비율을 구하였다. 이번에는 [0,100000] 구간에 무작위로 분포된 원소를 가지는 배열을 정렬하는데 걸리는 시간과 [0,1]구간에서 무작위로 분포된 원소를 가지는 배열을 정렬하는데 걸리는 시간의 비율을 구하여라.

Sol) 생략.

Chapter 8 End Point.