연습문제

6-1. 다음 코드를 실행시키면 4개의 값이 출력된다. 이것들 중 같은 값들은 몇 개 인가? 답에 대해 설명하여라.

```
char *format = "%p %d %d %d₩n";
int i = 3;
int *p = &i;
printf(format,p,*p+ 7,3* **&p + 1,5*(p-(p-2)));
```

Sol) 첫번째 p는 p가 위치한 주소를 표현하고 두번째 *p + 7은 *p는 I의 값을 가르키고 따라서 *p + 7은 10이 된다. 세번째에서 **&p는 *p 이므로 3 3 + 1 이므로 10이다. 마지막은 p가 제거되서 p에 상관없이 10이다.

```
[romance@161s ch6_code]$ ./a.out
0xbffffaa0 10 10 10
[romance@161s ch6_code]$
```

6-2. 컴파일러 중 어떤 것은 연습문제 1번의 수식 p – (p-2)에 대하여 정수 오버플로에 관한 경고를 낸다. 연습문제 1번에서 작성한 프로그램을 수정하여 p와 p-2 값을 출력하도록 하여라. 정수 오버플로가 발생하겠는가?

Sol) 에러나 경고가 나지 않는다.

```
[romance@161s ch6_code]$ cc -Wall 6_2.c
[romance@161s ch6_code]$ ./a.out
0xbffffaa0 10 10 10
0xbffffaa0 0xbffffa98
[romance@161s ch6_code]$
```

6-3. 다음 프로그램을 보자.

```
#include <stdio.h>
#include <stddef.h>
int main(void)
{
int a,b,*p = &a,*q = &b;
ptrdiff_t diff = p - q;
printf("diff   = %d₩n",diff);
return 0;
}
```

ANSI C에서는, 두 포인터의 차는 부호 있는 정수적형이어야 한다. 대부분의 UNIX 시스템에서 그 형은 int형이고, 대부분의 MS-DOS 시스템에서 그 형은 long이다. 모든 ANSI C 시스템에서, 그 형은 표준 헤더 파일 stddef.h 에 다음과 같은 형 선언으로 주어진다.

```
 typedef type ptrdiff_t;
```

현재 사용하고 있는 시스템의 stddef.h에서 이 typedef를 찾아보면, diff의 형을 알게 될 것이다. Diff가 int형이면 printf() 문에서 %d가 적절할 것이고, long형이면 %ld가 적절할 것이다. 이 프로그램을 수행한 후 그 효과를 이해하여라.

```
[romance@161s ch6_code]$ ./a.out
diff   = 1
[romance@161s ch6_code]$
```

그 후 프로그램에 다음과 같은 두 행을 추가하여라.

```
diff = p - (int *)0;
printf("diff = %d₩n",diff);
```

기대했던 출력값이 나왔는가? 만일 int *를 ptrdiff_t *로 대치한다면, 프로그램은 다르게 동작하겠는가?

```
[romance@161s ch6_code]$ ./a.out
diff   = 1
diff = -268435799
[romance@161s ch6_code]$
```

6-4. i와 j는 int형이고, p와 q는 int형 포인터이라면, 다음의 배정 수식 중 잘못 된 것은 어느 것인가?

p = &i p =&*&*i i = (int) p q = &p
q = &j i= (&&i) i = *&*&j I = *p + + + *q

Sol) p = &*&*i 와 q = &p 와 i = (&& i)가 잘 못되었다.

6-5. 변수가 선언되면, 그 변수들은 연속적인 메모리에 위치하는가? 다음과 같은 선언을 갖는 프로그램을 작성하여라.
char a,b,c,*p,*q,*r;
그리고 컴파일러에 의해 이 변수들이 배정된 메모리의 위치를 출력하여라. 메모리 위치는 순서적인가? 만일 순서적이라면, 증가하는 순서인가? 아니면 감소하는 순서인가? 각 포인터의 주소는 4로 나누어 지는가? 만일 그렇다면, 각 포인터 값은 기계워드에 저장됨을 의미하느것일 것이다.
Sol)

감소하는 순서이고 포인터의 주소는 4로 나누어 진다.

6-6. 다음 프로그램은 %p 형식을 사용하여 어떤 주소들을 출력한다.

```
#include <stdio.h>

int main(void)
{
int a =1,b=2,c =3;
printf("%s%p₩n%s%p₩n%s%p₩n","&a = ",&a,"&b = ",&b," &c = ",&c);
return 0;
}
```

```
[romance@161s ch6_code]$ ./a.out
&a = 0xbffffaa4
&b = 0xbffffaa0
 &c = 0xbffffa9c
[romance@161s ch6_code]$
```

변수 a,b,c가 초기화 되지 않았다면, 프로그램은 동일한 결과를 출력하겠는가?
%p를 %d로 수정하면 어떻게 되겠는가? 컴파일러는 오류 메시지를 출력하겠는가?

```
A,b,c가 초기화 되어 있지 않을 때 ::
[romance@161s ch6_code]$ ./a.out
&a = 0xbffffaa4
&b = 0xbffffaa0
 &c = 0xbffffa9c
[romance@161s ch6_code]$
```

```
%p를 %d로 수정한 경우 ::

[romance@161s ch6_code]$ cc 6_6_a.c
[romance@161s ch6_code]$ ./a.out
&a = -1073743196
&b = -1073743200
 &c = -1073743204
[romance@161s ch6_code]$
```

6-7. 만일 주소를 16진수 대신 10진수로 출력하여 보기 원한다면, 주소를 unsigned long으로 캐스트하고 %ld 형식을 사용 하는 것이 안전한 방법이다. 연습문제 6번의 printf()문을 다음과 같이 수정하여 실행하여 보아라.

Sol) 지금 사용하고 있는 머신이 리눅스 계열이라서 int와 포인터 값이 모두 4바이트로 간주 되기 때문에 6번의 경우에도 에러가 나지 않았고 7번에서 수정한 것 역시 의미 없어지게 되었다.

```
[romance@161s ch6_code]$ ./a.out
&a = -1073743196
&b = -1073743200
 &c = -1073743204
[romance@161s ch6_code]$
```

6-8. 다음 프로그램은 어떤 값을 출력하는가? 설명하여라.

```
#include <stdio.h>
typedef unsigned long ulong;

int main(void)
{
char *pc = NULL;
int *pi = NULL;
double *pd = NULL;
long double *pld = NULL;
printf("%5lu%5lu₩n%5lu%5lu₩n%5lu%5lu₩n",(ulong)(pc+ 1),(ulong)(pi+ 1)
,(ulong)(pd+ 1),(ulong)(pld+ 1),(ulong)(pc+ 3),(ulong)(pld+ 3));

return 0;
}
```

Sol)

초기 포인터는 pc는 0을 int형은 0를 double은 0을 long double은 0을 가르키고 있다. 초기화 NULL이기 때문이다. 따라서 포인터 연산을 통해서 pc+ 1은 현재 0의 주소 값에 char의 형인 1만큼을 더해서 1을 출력한다. 그리고 int형은 0에 포인터 변수의 데이터 형의 크기 만큼인 4를 더하게 되고, double은 8를 long double은 12를 더하게 된다. 또한 pc+ 3 은 0의 포인터에서 3*1을 더해서 3을 출력하게 되고, pld+ 3은 현재의 0의 포인터에 12*3을 해서 36의 주소를 가르키게 된다.
따라서 출력은 다음과 같게 된다.

```
[romance@161s ch6_code]$./a.out
    1     4
    8    12
    3    36
[romance@161s ch6_code]$
```

Additonal Text

(1) 포인터 연산이란

- 연산을 이용하여 포인터 값(주소)를 변경하는 것

- 가능한 포인터 연산 : =, +, -, ++, --, 비교연산(같은 형끼리)
  (예) p = &x;, p++;, p--;, p1-p2;, p+4;, p-2;, pa < pb;
- 포인터 연산은 정수만 사용 가능
  (예) p + 4.5
- 불가능한 포인터 연산 : *, /, %

(2) 포인터 연산의 실제 의미

실제 주소 = 포인터가 가리키는 주소 + 정수*포인터 변수 데이터형 크기

int *ip; (포인터 ip가 1000을 가리키고 있다고 가정)

ip + 2 => 1004(1000 + 2*4byte)
ip - 2 => 992(1000 - 2*4byte)

```
int a = 10;
int *p_a = &a;

printf("%d₩n", p_a);
printf("%d₩n", p_a + 1);
printf("%d₩n", p_a + 2);
```

6-9. 다음 배열 선언에는 몇 개의 오류가 있다. 오류들을 찾아보아라.

```
int a[N] = {0,2,2,3,4};
int b[N-5];
int c[3.0];
```

Sol) 첫번째 a[4]를 선언하고나서 5개의 원소를 넣음으로써 초과 된다.
두번째 b[-1]이 선언되므로 음수만큼의 원소는 선언될 수 없다.
세번째 c[3.0]에서 원소의 개수는 int형이어야만 하므로 에러가
날 것이다.

6-10. 다음 프로그램에서 change_it() 함수의 호출은 아무런 영향도 없을 것 같다. 이유를 설명하여라.

```
#include <stdio.h>
void change_it (int []);
int main(void)
{
int a[5],*p;
 p = a;
 printf("p has the value %p₩n",p);
 change_it(a);
 p = a;
 printf("p has the value %p₩n",p);
return 0;
}

void change_it (int a[])
{ int i = 777, *q = &i;
  a = q;
}
```

sol) a[]는 복사 되었기 때문에 call by value이기 때문에 main()으로 돌아 왔을 때는 a는 변하지 않은 상태로 되어 있다.

6-11. 다음 프로그램은 무엇이 잘못 되었는가? 프로그램을 올바르게 수정하고, 출력값의 의미를 설명하여라.

```
#include <stdio.h>

int main(void)
{
int a[] = {0,2,4,6,8},*p = a + 3;
printf("%s%d%s₩n%s%d%s₩n","a[?] = ",*p,"?","a[?+ 1] = ",*p+ 1,"?");

return 0;
}
```

Sol) 다음과 같이 수정하면 된다.

```
. . .
printf("%s%d%s₩n%s%d%s₩n","a[?] = ",*p,"?","a[?+ 1] = ",*(p+ 1),"?");
. . . .
```

따라서 a[0] 은 *a와 같은 의미이다. 따라서 a[1]은 *(a+ 1)
, 그리고 a[n]은 *(a+ n)이 되는 셈이다.

6-12. n차 이하의 실수 다항식 p(x)는 다음과 같다.

$$p(x) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n$$

여기서 계수 $a_0, a_1, ..., a_n$ 은 실수 이다. 만일 a(n) != 0 이면, p(x)의 차수는 n이다. 다항식은 다음과 같은 배열로 컴퓨터에서 표현될 수 있다.

```
#define N 5
double p[N+ 1];
```

주어진 x에 대한 다항식 p의 값을 리턴하는 다음 함수를 작성해 보아라.

```
 Double eval(double p[] , double x , int n)
{ ....
```

두가지 버전의 함수를 작성하여라. 첫번재 버전은 직관적인 방법을 사용하여 값을 계산해야 한다. 두번째 버전은 Horner의 법칙을 이용하라.

Sol)

C code

1) 첫번째 버전

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

#define N 5
double p[N+1];
double eval(double *,double,int);

int main(void)
{
double x = 1.0;
p[0] = 1;
p[1] = 2;
p[2] = 3;
p[3] = 4;
p[4] = 5;
p[5] = 6;
printf("%g₩n",eval(p,x,N));
return 0;
}

double eval(double p[],double x,int n)
{
double sum;
int i;
 for (i = 0; i <= n; i++)
 { sum = sum +  p[i]*pow(x,n);
 }
 return sum;
}
```

1) 두번째 버전

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

#define N 5
double p[N+1];
double eval(double *,double,int);

int main(void)
{
double x = 1.0;
p[0] = 1;
p[1] = 2;
p[2] = 3;
p[3] = 4;
p[4] = 5;
p[5] = 6;
printf("%g₩n",eval(p,x,N));
return 0;
}

double eval(double p[],double x,int n)
{
 if (n == 0) return p[N-n];
 return p[N-n]+x*eval(p,x,n-1);
}
```

6-13. 최대 n차 두 다항식을 더하는 함수를 작성하여라.
void add (double f[], double g[],double h[],int n)
Sol) 같은 차수끼리 더하면 된다.

```
void add(double f[], double g[], double h[],int n)
{
 int i;
  for(i=0;i<n;i++)
  { f[i] = g[i]+h[i] ;
  }
}
```

6-14. 최대 n차의 다항식을 곱하는 알고리즘을 작성하여라. 그리고 중간 결과를 더하기 위해 연습문제 13번에서 작성한 add()함수를 사용하여라. 이 알고리즘은 매우 비효율적이다. 더 효율적인 루틴을 작성할 수 있는가?

Sol)

```
void multi(double f[],double g[],double h[],int n)
{ int i,j;
  for (i=0; i<=n; ++i)
  { for (j=0;j<=n; ++j)
        f[i+j] = f[i+j] + g[i] + h[i];
  }
  }
```

6-15. bubble() 함수를 수정하여 아무러 원소도 교환되지 않으면 종료 하도록 하여라.

Sol) bubble 함수에서 원소가 교환되는 부분에 카운터를 달아서
카운터가 초기화 된 그대로면 종료 하도록 하면 된다.

6-16. mergesort() 함수를 2의 거듭제곱 크기에 대해서만이 아니라 임의의 크기의 배열에 대해서도 사용할 수 있도록 수정하여라. 이때 임의의 양수는 2의 거듭제곱의 합으로 표현할 수 있음을 상기하여라. 예를 들면 다음과 같다.

27 = 16 + 8 + 2 + 1;

배열을 2의 거듭 제곱 크기의 부분 배열의 집합으로 생각해 보자. 각 부분 배열을 정렬하고, 최종적으로 정렬된 배열을 만들기 위해 merge()를 사용하여라.

Sol)

6-17. p가 포인터라면, *p++와 (*p)++은 다르다. 다음 코드의 출력 결과는 무엇이 겠는가? 설명하여라.

```
#include <stdio.h>
int main(void)
{
char a[] = "abc";
char *p;
int i;
p = a;
for(i=0;i<3;++i)
printf("%c₩n",*p++);
printf(" a= %s₩n",a);
p = a;
for(i=0;i<3;++i)
printf("%c₩n",(*p)++);
printf("a = %s₩n",a);
return 0;
}
```

Sol) *p++은 *(p++) 이고 (*p)++와 다르다. 따라서 다음과 같은 결과를 보인다.

Result (결과)

```
[romance@161s ch6_code]$ ./a.out
a
b
c
 a= abc
a
b
c
a = dbc
[romance@161s ch6_code]$
```

6-18. 6.11절에서 제시한 strcpy()를 위한 함수 정의를 잘 살펴보고, 문자열의 끝 부분을 문자열의 앞에 복사하는 것이 가능한지 생각해 보아라. 다음 코드의 출력 결과는 무엇이겠는가? 설명해 보아라.

```
#include <stdio.h>

int main(void)
{
char a[] = "abcdefghijklmnopqestuvwxyz";
char *p = a;
char *q = a +  strlen(a) - 3;
printf("a = %s₩n",a);
strcpy (p,q);
printf("a = %s₩n",a);

return 0;
}
```

Sol) 뒤에서 3번째 문자열부터 끝까지가 a에 덮어 쓰기가 된다.
따라서 나중 a 출력은 xyz가 된다.
만약 p와 q를 거꾸로 쓴다면 p에 비해 q가 작은 공간을 지니고 있기 때문에 segmentation fault 에러가 나게 된다.

6-19. 6.7절에서 제시한 bubble() 함수와 연습문제 16번에서 작성한 mergesort()함수의 상대적 효율성을 검사하는 프로그램을 작성하여라. 배열을 초기화 하기 위해 rand() 함수를 사용하여라.

Sol)

6-20. 회문은 앞뒤 어느쪽으로 읽어도 똑 같은 문자열을 뜻한다. 예를 들면, 다음과 같다.
“ABCBA” “123343321”
문자열 인자를 받아들여 회문이면 int 1을 아니면 0을 리턴하는 함수를 작성하여라.

Sol)

C code

```
#include <stdio.h>

int main(void)
{

char a[] = "abcba";
printf("check = abcbc result == %d₩n",check(a));
strcpy (a,"12345");
printf("check = 12345 result == %d₩n",check(a));

return 0;
}
int check(char a[])
{
   int i, ch_cnt = 1;
   int n;
   n = strlen(a)/2 + 1;
      for (i=0;i<n;++i)
      { if (a[i] != a[strlen(a)-i-1]) {ch_cnt = 0; }
      }
    return ch_cnt;
}
```

6-21. 연습문제 20번에서 작성한 함수의 비교 과정에서 공백과 대문자는 무시하도록 수정하여라.

Sol) 공백문자를 처리하는 경우의 코드이다. 대문자를 무시하기 위해서는 알파벳일 경우 A를 a로 고치는 과정 ASCII code값을 빼주는 과정을 넣으면 된다.

C code

```
#include <stdio.h>

int main(void)
{

char a[] = "abcba";
printf("check = abcbc result == %d₩n",check(a));
strcpy (a,"a b cba");
printf("check = a b cba result == %d₩n",check(a));

return 0;
}
int check(char a[])
{
  int i,j,ch_cnt = 1;
  j = strlen(a)-1;
  i = 0;
    while (i < j)
    {   while (a[i] == ' ') {++i;}
        while (a[j] == ' ') {--j;}
        if (a[i] != a[j]) {ch_cnt = 0; }
        ++i; --j;
    }
   return ch_cnt;

}
```

6-22. 다음 코드가 출력하는 것을 적어보고, 설명하여라.
printf(“%c%c%c%c%c!₩n”,”ghi[1],*(“def”+ 1),*”abc”+ 11,”klm”[1],*”ghi”+ 8);

Sol) “ghi”[1] 은 h이고 *(“def”+ 1)은 e를 가르키고 *“abc”+ 11은 a의 아스키값을 11더한 값인 l이고 “klm”[1]은 ㅣ을 가르키고 “ghi” + 8은 g에 8을 더한 o가 된다. 따라서 hello!를 출력하게 된다.

```
[romance@161s ch6_code]$ ./a.out
hello!
[romance@161s ch6_code]$
```

6-23. 3장에서는 long int에 저장할 수 있는 가장 큰 수는 약 20억임을 보았다. 많은 응용 프로그램에서, 이 수의 크기는 충분치 않다. 예를 들어, 미연 방정부는 수조의 숫자를 다룬다. 이 연습문제에서는 원시적인 방법으로 어떻게 커다란 두개의 수를 다루는지 살펴본다.

Sol) 생략

6-24. sizeof 연산자는 형이나 수식을 저장하기위해 필요한 바이트 수를 구하는 데 사용된다. 이 연산자를 배열로 적용하면, 배열의 크기를 내지 않는다. 어떤 값이 출력되는가? 설명하여라.

```
#include <stdio.h>

void f(int a[]);
int main(void)
{
char s[] = "deep in the heart of texas";
char *p = "deep in the heart of texas";
int a[3];
double d[5];
```

```
printf("%s%d₩n %s%d₩n%s%d₩n%s%d₩n","sizeof(s) = ",sizeof(s),
"sizeof(p) = ",sizeof(p),"sizeof(a) = ",sizeof(a),"sizeof(d) = ",sizeof(d));
f(a);
return 0;
}
void f(int a[])
{ printf("In f() : sizeof(a) = %d₩n",sizeof(a));
}
```

Sol) sizeof(s)는 전체 배열의 길이를 나타내서 27을 나타내고 sizeof(p)는 int형 포인터의 길이를 나타내 4가 나온다. Sizeof(a)는 배열의 전체 길이인 4*3 = 12를 나타내고 double은 그것의 2배인 8*5 = 40을 sizeof(d)로 나타내게 된다.
반면에 서브 함수로 들어가는 것은 포인터 함수 이므로 f()안에서의 sizeof(a)는 4가 될 것이다.

Result (결과)

```
[romance@161s ch6_code]$ ./a.out
sizeof(s) = 27
 sizeof(p) = 4
sizeof(a) = 12
sizeof(d) = 40
In f() : sizeof(a) = 4
[romance@161s ch6_code]$
```

6-25. UNIX를 사용하고 있고 diff 유틸리티에 익숙하다면, 다음을 실험해 보아라.
다음 문장을 하나씩 갖는 두개의 프로그램을 작성하여라.

```
printf("abc₩n");
printf("a%cb%cc₩n",'/0'.'/0');
```

재지정을 사용하여 각 프로그램의 출력을 각각 tmp1과 tmp 파일에 저정하여라.
그리고 cat을 사용하여 두 파일을 출력해 보고, 결과를 비교해 보아라. 아마 출력결과는 같을 것이다. 다음 명령을 수행해보아라.

```
diff tmp1 tmp2
```

```
[romance@161s ch6_code]$ cat tmp1
abc
[romance@161s ch6_code]$ cat tmp2
abc
[romance@161s ch6_code]$

[romance@161s ch6_code]$ diff tmp1 tmp2
Binary files tmp1 and tmp2 differ
[romance@161s ch6_code]$
```

6-26. 비록 나쁜 프로그램 습관이지만, 전통적인 C에서는 문자열 상수의 내용을 변경할 수 있게 하였다. ANSI C는 프로그래머가 문자열 상수를 변경할 수 없도록 규정하고 있다. 그러나, 이것의 적용은 컴파일러에 따라 다르다. 다음 코드를 보자.

```
char *p = “abc”;
*p = ‘X’;                    // illegal?
printf(“%s₩n”,p);           // Xbc gets printed?
```

Sol)

```
[romance@161s ch6_code]$ cc 6_26.c
[romance@161s ch6_code]$ cc -Wall 6_26.c
[romance@161s ch6_code]$

[romance@161s ch6_code]$ ./a.out
Segmentation fault
[romance@161s ch6_code]$
```

6-27. 다음 코드를 보자.

```
char *p = "abc" , *q = "abc";
if (p == q)
```

printf("The two string have the same address!₩n");
else
printf("As I Expected, the address are different.₩n");

p와 q는 문자열 상수 “abc”의 기본 메모리 주소로 초기화 되었다. P == q는 p와 q의 값이 같은지 검사하는 것이다. 즉, p와 q가 포인트하고 있는 곳의 내용을 검사하는 것은 아니다. 메모리에 이 두 문자열 상수가 따로 있겠는가? 아니면 하나로 있겠는가?

Sol) 같은 주소에 있음을 확인 했다.

```
[romance@161s ch6_code]$ cc 6_27.c
[romance@161s ch6_code]$ ./a.out
The two string have the same address!
[romance@161s ch6_code]$
```

6-28. ANSI C 위원회는 C언어에 새로운 예약어인 형 한정자 const를 추가하였다.
다음 예제를 보자.
const char *p;
형 한정자 const는 컴파일러에게 p가 포인트하는 문자는 변경될 수 없음을 알려준다. 다음 코드를 수행해 보아라.

```
char s[] = "abc";
const char *p = s;
*p = 'A';
printf("%s₩n",s);
```

Sol)

```
[romance@161s ch6_code]$ cc 6_28.c
6_28.c: In function `main':
6_28.c:7: warning: assignment of read-only location
[romance@161s ch6_code]$ ./a.out
Abc
[romance@161s ch6_code]$
```

6-29. 기계 번역 문제에 많은 노력이 집중되고 있다. 우너시적인 방법이 얼마나 성공적인가? 가장 일반적으로 사용 되는 영어 단어 100개를 찾아보아라. 그 100개의 단어를 적고, 한영 사전에서 그 단어들을 찾은 후 뜻을 적어보아라. 한글 문장을 번역하기 위해 다음과 같은 두 배열을 사용하는 프로그램을 작성해 보아라.

```
char *foreign[100], *english[100];
```

Sol) 생략

6-30. 간단한 암호화 기법은 일대일로 알파벳 문자를 바꾸는 것이다. 이 기법은 52개의 소문자와 대문자에 대한 번역표를 가지고 수행할 수 있다. 이 기법을 사용하여 문서를 암호화 하는 프로그램을 작성하여라.

Sol)
생략

6-31. 다음은 어떤 값을 출력하겠는가? 설명해 보아라.

```
#include <stdio.h>
void try_me(int [][3]);

int main(void)
{
int a[3][3] = {{2,5,7},{0,-1,-2},{7,9,3}};
 try_me(a);
return 0;
}
void try_me(int (*a)[3])
{ printf("%d %d %d . . . infinity₩n",a[1][0],-a[1][1],a[0][0],a[2][2]);
}
```

```
[romance@161s ch6_code]$ cc 6_31.c
./[romance@161s ch6_code]$ ./a.out
0 1 2 3 . . . infinity
[romance@161s ch6_code]$
```

이제 try_me() 함수 정의의 헤더에서 매개 변수 선언을 다음과 같이 수정해 보아라.

```
 int *a[3]
```

컴파일러가 오류 메시지를 출력하는가?

```
[romance@161s ch6_code]$ cc 6_31_a.c
6_31_a.c:11: conflicting types for `try_me'
6_31_a.c:2: previous declaration of `try_me'
[romance@161s ch6_code]$
```

6-32. 0-2π 구간에서 sin()과 cos() 함수를 화면에 그래프로 그리기 위해, 하나의 문자와 화면의크기에 맞는 2차원 배열을 선택하여라. 대부분의 화면에서 출력되는 한 문자의 영역이 정방형이 아니므로, 수평/수직 외곡이 일어난다.

Sol) 생략

6-33. 다음 프로그램을 분석하여 분석한 내용을 적어보아라. 이를 위해서는 기억장소 사상 함수에 대한 이해가 필요하다. 마지막 printf()문은 더욱 전문적이기 때문에, 컴퓨터를 전공하는 학생만 설명하라.

```
#include <stdio.h>

int main(void)
{
int a[3][5],i,j,*p = *a;
for (i=0;j<5;++i)
 for (j=0;j<5;j++)
    a[i][j] = i *5 +  j;
for (i=0; i<3; ++i)
 for( j = 0;j<5;++j)
    printf("%s%12d",(j == 0) ? "₩n" :"", a[i][j]);
 printf("₩n");
for ( i = 0; i <15;++i)
```

```
 printf("%s%12d",(i % 5 == 0) ? "₩n" :"",*(p+ i));
printf("₩n₩n%12d%12d₩n%12d%12d₩n%12d%12d₩n%12d%12d₩n₩n",
*a,**(a+ 1),
*(a[0]+ 1),*(*a+ 1),
*(a[1]+ 2),*(*(a+ 1)+ 2),
*(a[2]+ 3),*(*(a+ 2)+ 3));
printf("%-11s*s%12d₩n%-11s%s%12d₩n%-11s%s%12d₩n₩n",
"(int) a","=",(int)a,
"(int) *a","=",(int) *a,
"(int) **a","=",(int) **a);

return 0;
}
```

Sol) 이 프로그램에서는 a[3][5]에 0부터 14까지의 수를 차례대로 집어 넣게 된다. 그 후에 *(p+I)는 포인터 연산이므로 I가 0부터 14까지 증가함에 따라 a[0][0]부터 a[2][4]까지의 수를 차례대로 출력하게 된다. 그 다음 *a는 a[0][]를 **(a+1)은 a[0][1]을 *(a[0]+1)은 a[0][1]을 출력하게 된다. 나머지도 포인터 연산을 잘 계산하면 그러한 값을 출력할 수 있음을 알 수 있게 된다.
다음은 출력 결과이다.

Result (결과)

```
[romance@161s ch6_code]$ ./a.out

          0          1          2          3          4
          5          6          7          8          9
         10         11         12         13         14

          0          1          2          3          4
          5          6          7          8          9
         10         11         12         13         14

 -1073743264             5
          1          1
          7          7
```

```
            13              13

(int) a      = -1073743264
(int) *a     = -1073743264
(int) **a    =              0

[romance@161s ch6_code]$
```

6-34. 6.14절의 my_echo 프로그램을 수정하여 –c 옵션을 사용할 경우 프로그램의 인자를 대문자로 출력하게 하여라. 단 이 옵션을 포함하는 인자는 출력하면 안된다.

Sol)

C code

```
#include <stdio.h>

int main(int argc, char *argv[])
{
int i,j;
int c;
printf("argc = %d₩n",argc);
if (argv[1][0] == '-' && argv[1][1] == 'c' ) {
for (i=2; i< argc;++i)
{ printf("argv[%d] = ",i);
    j = 0;
       while (j < strlen(argv[i]) )
             {   c = argv[i][j];
                 if (c >= 'a' &&   c <= 'z')
                     putchar(c - 'a' + 'A');
               else putchar(c);
               ++j;
             }
         printf("₩n");
}
}
```

```
else
{
for (i=0; i< argc; + + i)
 printf("argv[%d] = %s₩n",i,argv[i]);
}
return 0;
}
```

6-35. 다음 표를 완성하여라.

선언 및 초기화		
- char *p[2][3] = {“abc”,”defg”,”hi”,”jklmno”,”pqrstivw”,”xyz”};		
수식	등가 수식	값
***p	p[0][0][0]	a
**p[1]	p[1][0][0]	j
**(p[1]+ 2)	p[1][2][0]	x
((p+ 1)+ 1)[7]	/* error */	
(*(*(p+ 1)+ 1))[7]	p[1][1][7]	w
*(p[1][2]+ 2)	p[1][2][2]	z

Result (결과)

```
[romance@161s ch6_code]$ ./a.out
a
j
x
w
z
[romance@161s ch6_code]$
```

6-36. 확률적 사건에 대하여 난수 발생기를 연속적으로 사용하는 모의 실험을 몬테카를로 시뮬레이션이라고 한다. 이렇게 부르는 이유는 몬테카를로가 세계에서 가장 유명한 카지노 게임이 벌어지는 곳 중 하나이기 때문이다. 이 연습문제에서는 n명의 사람

이 있는 방에서 최소한 두 사람의 생일이 같은 해 같은 날짜일 확률을 구하여 한다.

Sol) 생략

6-37. int형 배열이 아닌 char형 포인터의 배열에 대해 동작하도록 앞에서 제시한 merge() 함수와 연습문제 16번에서 작성한 mergesort() 함수를 수정하여라.

Sol) 생략

6-38. 다음 코드는 문자열에 있는 문자를 역순으로 만들기 위하여 사용될 수 있다.

```
char *reverse(char *s)
{
  char *p,*q,tmp;
  int n;
  n = strlen(s);
  q = (n >0) ? s+ n -1 :s;
  for (p = s; p <q; + + p,--q) {
   tmp = *p;
   *p = *q;
   *q = tmp;
  } return s;
}
```

다음 문장을 포함하는 프로그램을 작성하여 이 함수를 검사해 보아라.

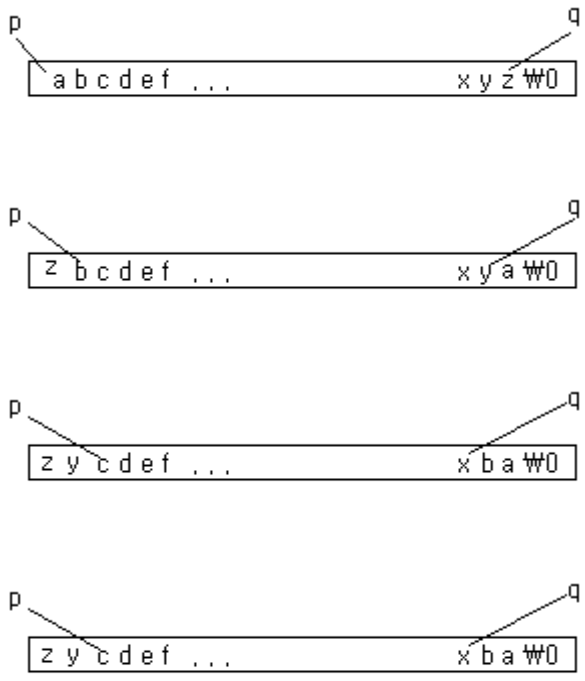
```
char str[] = "abcdefghijklmnopqrstuvwxyz";
printf("%s₩n",reverse(str));
```

```
[romance@161s ch6_code]$ ./a.out
zyxwvutsrqponmlkjihgfedcba
[romance@161s ch6_code]$
```

for 루프의 각 반복 후의 메모리 상태를 유사한 그림으로 그려보아라.

Sol)

Result (결과)



....
이런식으로 문자열이 서로 뒤 바뀌게 된다.

6-39. 일반적으로 argc 와 argv 는 main() 함수의 인자로써 사용된다. Argc는 명령어 라인에서의 인자 수이기 때문에, 배열 argv의 크기를 argc라고 생각할 수도 있지만 그렇지 않다. 배열 argv는 argc +1 의 크기를 갖고, 배열의 마지막 원소는 널 포인터이다. 다음은 echo 명령어의 다른 버전이다.

```
#include <stdio.h>

int main(int argc, char **argv)
{
while(*argv != NULL)
   printf("%s ",*argv++);
 putchar('₩n');
return 0;
}
```

이 프로그램을 수행시켜 보고, 이것을 이해하여라. 그리고 이 프로그램이 어떻게 동작하는지 상세하게 기술하여라.

Sol)

```
int main(int argc, char **argv)
{
while(*argv != NULL)          // ' '문자열로 나누어지는 문자열을 *argv로 받아서
                                 이것이 NULL이 아니면, 즉 문자열이 있으면
                                  계속 while loop 를 돈다. //
  printf("%s ",*argv++);     // 먼저 argv++ 로 포인터 연산을 한후,
                                  *argv를 받아서 문자열을 출력한다.
                                  따라서 첫번째 문자열인 ./a.out은 출력되지 않는
                                 다.
 putchar('₩n');
return 0;
}
```

6-40. C에서 하나의 함수 원형은 여러 번 나올 수 있다. 더욱이 동등한 함수 원형은 서로 충돌되지 않는다. 6.18절에 제시한 find_roots 프로그램에는 다음 함수 원형이 있었다.

- dbl bisection (dbl f(dbl x),dbl a, dbl b);

이것을 다음과 같은 함수 원형 목록으로 대치해 보아라.

- dbl bisection (pfdd,dbl,dbl);
- dbl bisection (pfdd f, dbl a,dbl b);
- dbl bisection (dbl (*)(dbl), dbl a, dbl b);
- dbl bisection (dbl (*f)(dbl),dbl a, dbl b);
- dbl bisection (dbl f(dbl), dbl a, dbl b);
- dbl bisection (dbl f(dbl x), dbl a, dbl b);

컴파일 할 때 문제가 발생하는 가?

Sol) 문제가 없다.

6-41. 6.18절 “함수 포인터의 배열”에서 find_roots 프로그램을 주어진 방정식의 근을 계산할 때마다 bisection() 을 52번 호출하였다. 왜 정확히 52번 호출하는지 손으로 계산해 보아라.

Sol) m = (10 + -10) /2.0
f(m) = -3
bisection (f,10,-3)

m = (10 + -3)/2.0
f(m) = 497.7
bisection (f,3.5,-3)

....
계속 반복하여 52번 호출하게 된다.

6-42. 다음 프로그램을 컴파일하고 실행하여 프로그램을 이해하여라.

```
#include <stdio.h>
#include <string.h>
void tell_me(int f(const char *,const char *));

int main(void)
{
tell_me(strcmp);
tell_me(main);
return 0;
}

void tell_me (int f(const char *,const char *))
{ if (f == strcmp)
    printf("Address of strcmp() : %p ₩n",f);
  else
    printf("Function address : %p ₩n",f);
}
```

```
[romance@161s ch6_code]$ cc 6_42.c
```

```
6_42.c: In function `main':
6_42.c:8: warning: passing arg 1 of `tell_me' from incompatible pointer type
[romance@161s ch6_code]$ ./a.out
Address of strcmp() : 0x8048340
Function address : 0x8048490
[romance@161s ch6_code]$
```

tell_me() 함수에 대한 두번째 호출에서, 전달되는 포인터가 잘못된 형을 가지고 있기 때문에 컴파일러는 경고 메시지를 발생시킬 것이다. 그러한가? 경고메세지가 나타나지 않도록 void * 포인터를 사용하는 프로그램으로 수정하여라.

6-43. 다음 프로그램은 오류를 포함하고 있다.

```
#include <stdio.h>
#include <string.h>
int main(void)
{
char *p1 = "abc",*p2 = "pacific sea";
printf("%s    %s    %s ₩n",p1,p2,strcat(p1,p2));
return 0;
}
```

그 결과는 컴파일러에 따라 달라진다. 어떤 컴파일러를 사용하면, 이 프로그램은 실행시간 오류를 발생시킨다. 그리고, 다른 컴파일러를 사용하면, 다음과 같은 결과가 화면에 출력된다.

abcpacific sea abcific sea abc pacific sea

이러한 결과는 의미가 있으며, 컴파일러에 대한 무엇인가를 알려 준다. 프로그래밍 오류는 무엇인가?

Sol) 프로그램의 의도는 먼저 p1 문자열과 p2 문자열을 출력하고 나서 p1과 p2를 합친다음 그 문자열을 출력하고자 하지만 위의 코드에서는 strcat(p1,p2)함수가 먼저 실행되기 때문에 p1 에 이전의 p1문자열과 p2 문자열이 합쳐진 'abcpacific sea가 들어가 있게 된다.
또한 p1의 문자열이 p2와 합쳐져서도 충분한 공간을 확보하고 있어야 하지만 그렇지 못하기 때문에 segmetation 에러가 나게 된다.

6-44. 컴파일러는 함수 이름 그 자체를 포인터로 취급한다. 이것은 C의 일반적인 규칙이다. 다음 코드는 옳은가? 컴파일 하기 전에 답하여 보아라.

```
#include <stdio.h>
void f(void);
void g(void);
void h(void);
int main(void)
{
   (*f)();

return 0;
}

void f(void)
{ printf("Hello from f().₩n");
 (((*g)))();
}
void g(void)
{ printf("Hello from g().₩n");
   (*(*(*h)))();
}
void h(void)
{ printf("Hello from h().₩n");
}
```

Sol) 코드는 올바르다. 다음 출력결과이다.

```
[romance@161s ch6_code]$ ./a.out
Hello from f().
Hello from g().
Hello from h().
[romance@161s ch6_code]$
```

6-45. bisection()이 실행되기 위해 선행 조건은 f(a)와 f(b)가 서로 반대 부호를 갖는 것이다. 이 조건을 검사하기 위하여 bisection()의 시작 부분에서 단정을 사용하여 보아라.

Sol) assert(f(a) – f(b) < 0); 를 추가하면 된다.

6-46. 고대 이집트 인들은 상형문자를 사용하였다. 이 시스템에서는 모음이 없고 자음만이 있다. 일반적으로 모음을 사용하지 않는 영어를 이해할 수 있겠는가? 시험적으로 주어진 문자가 모음이면 1을 리턴하고 그렇지 않으면 0을 리턴하는 is_vowel()함수를 작성하여라. 이 함수를 사용하여 표준 입력 파일에서 읽고 모음을 삭제한 후 표준 출력파일에 쓰는 프로그램을 작성하여라.

Sol)

C code

```
#include <stdio.h>

int is_vowel(char);

int main(void)
{
char c;
while((c = getchar()) != EOF)
{ if (is_vowel(c)) {}
   else { putchar(c); }
}

return 0;
}

int is_vowel(char c)
{
 if ( c == 'a' || c == 'A' || c == 'e' || c == 'E' || c == 'i' || c=='I' || c == 'o'
|| c == 'O' || c == 'u' || c == 'U' )
 return 1;
```

```
 else return 0;
}
```

Result (결과)

```
[romance@161s ch6_code]$ cc 6_46.c
[romance@161s ch6_code]$ ./a.out < 6_46.c > outfile
                    // 위의 파일을 소스로 하여 모음을 제거 하였다.
[romance@161s ch6_code]$ vi outfile
#ncld <std.h>

nt s_vwl(chr);

nt mn(vd)
{
chr c;
whl((c = gtchr()) != F)
{ f (s_vwl(c)) {}
   ls { ptchr(c); }
}

rtrn 0;
}

nt s_vwl(chr c)
{
 f ( c == '' || c == '' || c == '' || c == '' || c == '' || c== '' || c == ''
|| c == '' || c == '' || c == '' )
 rtrn 1;
 ls rtrn 0;
}
```

Chapter 6 End Point.