7.7 Dynamic memory allocation

* 정보를 저장하기 위하여 변수를 이용하는 방법

1) 전역변수 :
- 프로그램 실행 내내 메모리에 상주
- 크기를 미리 알아야 함
- 가능한 크기는 메모리 모델에 따라 차이가 있으나 약 64 KB 이내
- Data Segment

2) 내부변수
- 함수의 실행 시에 만 메모리에 상주
- 함수의 종료와 함께 사라짐
- 크기를 미리 알아야 함
- 프로그램 수행 시 Stack에 저장됨
- Stack 의 크기에 따라 제한됨

segments
- header
- code segment
- data segment
- stack segment
- far/near heap

* malloc , calloc 및 free 이용하는 방법
- 필요에 따리 메모리 할당/해제
- 포인터와 배열을 적절히 이용
- 경우에 따라 변수의 크기 조절
- 메모리의 사용이 효율적임

* 관련함수
* header : alloc.h

(1) near heap 관련함수

malloc ()	: near heap 에 메모리 할당
calloc	: near heap 에 메모리 블록 할당
free	: 메모리 해제 (near heap)
coreleft	: near heap 에 남은 메모리 양

(2) far heap 관련함수

farmalloc () , _fmalloc	: far heap 에 메모리 할당
farcalloc, _fcalloc	: far heap 에 메모리 블록 할당
farfree< _ffree	: 메모리 해제 (near heap)
farcoreleft, _fcoreleft	: near heap 에 남은 메모리 양

* 사용법 : see page 216 example

* malloc , calloc 사용시 주의점
  - always check result
    ex)

```
a = malloc (n* sizeof(int)) ;
if(a==NULL)
{
        :
}
```

  - always free allocated memory

* Offsetting pointer
  - array & vectors for mathematical uses
  - use +1 item for normal indexing

7.9 Merge Sort

7.10 Strings

- C 언어에서는 string 이라는 데이터 형이 없으므로 문자 포인터 또는 문자 배열로 대치하여 이용.
- 문자 배열의 끝에 '\ 0' 를 두어서 끝을 판명

* string : special case of one dimensional array of type char
* NULL termination
    - string must ending with '\ 0'
    - array must have enough space to hold entire string and NULL
* initialization

```
char *s = "String" ;
char s[] = "String" ;
char s[100] = "String" ;
```

* rule
    - enough space
    - null termination

7.11 string 함수

- strcat : append a string
- strpy : copy string

ex)

```
strcpy(s,"Hello") ;
strcat(s," Mr. Kim ") ;
```

- strchr : find a character in a string

ex)

```
ith = strchr(s,'K') ;
```

- strcmp : string comparison
- strlen : string length
- 기타 : strcspn , strdup, strerr, stricmp, strlwr, strupr, strncat, strncmp, strnset 등

7.12 Mutidimensional Array

7.13 Array of Pointers

- array of pointers : useful for string handling

see example p.285

7.14 Arguments to main()

- argc and argv : communication with operating system

```
#include <stdtio.h>

int main (int argc, char *argv[])
{
        :
}
```

*argc : number of command -line parameters

*argv : array of pointers for command line parameters

7.15 Raged Arrays

7.16 Functions as arguments

7.17 Example : Bisection and Newton-Raphson Method

* Objective:

find x

which satisfies f(x) = 0

* methods

1) Open method: Newton-Raphson, Secant, and Muller's Method

2) Bracketed method: Bisection, Brent-Decker method …

(1) Bisection

* theory :

if f(a) and f(b) have opposite sign (f(a) * f(b)<0) , it has root in the interval [a,b]

* method :

divide intervals

* characteristic

most simple

most reliable

most time-consuming

see code example in p.297

-> recursive function call

to avoid recursive funciton call

```
if(f(a)*f(b) >0)
{
        printf("Invalid Interval) ;
        exit(1) ;
}

while ( b-a < eps)
{
        m = (a + b) / 2.0   ;
        if ( f(m) *f(a) >= 0 ) a = m ;
        else b = m ;
        ncount++   ;
}
```

(2) Newton-Raphson Method

* theory : based on Taylor series approximation formula

$f(x) = f(a) + f'(a)(x-a) + \ldots$

when $f(x) = 0$ then

$x - a = f(a) / f'(a)$

$x_{n+1} = x_n + f(x_n)/f'(x_n)$

* method : iterative calculation
* characteristic
  - simple calculation procedure
  - require function derivative $f'(x)$
  - faster than bisection, slower than other higher order method

* code exmaple

```
xold = 1.0
xnew = 100.0
while( dabs(xnew-xold) < epsilon && ncount < 100 )
{
        tmp = xnew ;
        xnew = xold + f(xold) / fp(xold) ;
        xold = tmp ;
        ncount ++ ;
}
```

* Improvement: require modification for fp(xold) = 0 cases

(HW) Kepler equation

$f(x) = x - e\sin(x) - m$, $m = 2.2$, $e = 0.5$

일 때 Bisection 과 N-R 방법을 이용하여 근을 구하고 유사한 정확도의 근을 얻기 위한 반복계산 횟수를 비교하라.