6. Functions and Storage Class

6.1 Functions

* Top -down method of programming
    : breaking into small, manageable piece
* 함수를 사용하는 이유
    - repeated operation
    - modularization: maintenance
    - readability
    - a black box defined by input and output
* function definition

```
type function_name ( parameter list)
{
        decalaration
        statements
}
```

ex → see p.198

* by default type int is assumed if not specified

* return statement
    - value or expression is passed to caller
    - control is passed back to caller

ex → see p.201

* function prototypes

  - functions should be declared before they are used.

* function declaration style

```
ex)
int add(int a , in b)
{
        :
}

add (a,b)
int a, int b ;
{
        :
}
```

* funciton prototype style

```
int add(int a, int b) ;          → ANSI standard
int add(ini, int) ;              → ANSI allowed
```

* an alternate style for function declaration order

  → page 206

* function call -by -value

  - call by value
  - call by reference

* developing a large programs

  - use of make program

6.2 Storage Class

- 변수의 통용범위 (scope)와 존속기간을 결정

(1) auto (자동변수)
- default for variables defined within function bodies
- localized variable within a block

(2) extern (외부변수)
- default for varible defined outside function bodies
- global variable across the blocks

(3) register (레지스터변수)
- variable should be stored in high speed register
- size of variable is limited by the CPU of computer
- number of register variable is limited by the CPU of computer

(4) static (정적변수)
- value -retention use

ex → page 220.

(5) external static variable
- similar to external varible
- external variable is only used in a file scope

* default initialization

external, static variable → initialized by 0

automatic, regixter variable → initialized by garbage

* function recursion

see p.223 & 224 for example