재귀적 함수

```
# include <iostream>
using namespace std ;

long factorial(const int number) ;

int main() {
  int number ;
  cin >> number ;

  const long result = factorial(number) ;
  cout << "Factorial of " << number << " : " << result << endl ;
}

// 재귀적함수; stack overflow의 위험이 있으므로 반복문이 좋을수가 있음
long factorial(const int number) {
  if ( number == 1 ) return 1 ;

  return number * factorial(number-1) ;
}
```

재귀적 함수

```
int binarySearch(const vector<int>& numbers,
    int min, int max, int target) {
  int mid = (min + max) / 2 ;
  if ( numbers[mid] == target ) return mid ;
  if ( min >= max ) return -1 ;
  if ( target < numbers[mid] )
    binarySearch(numbers, min, mid-1, target) ;
  else
    binarySearch(numbers, mid+1, max, target) ;
}

int main() {
  int data[] = {1, 3, 5, 8, 20, 30} ;
  vector<int> numbers(data, data+sizeof(data)/sizeof(int)) ;
  cout << binarySearch(numbers, 0, numbers.size()-1, 50) ;
}
```

재귀적 함수의 위험성

- 재귀적 함수는 스택 공간의 부족(stack overflow)을 유발
- 대신에 반복문을 사용

```
long factorial(const int number) {
  long result = 1 ;
  for ( int i = 2 ; i <= number; i ++)
    result *= i ;
  return result ;
}
```

```
int binarySearch(const vector<int>& numbers, const int min, const int max, const int target)
{
  while ( max > min ) {
    const int mid = (min + max) / 2 ;
    if ( numbers[mid] == target ) return mid ;
    if ( min >= max ) return -1 ;
    if ( target < numbers[mid] ) max = mid -1 ;
    else min = mid+1 ;
  }
  return -1 ;
}
```

함수의 선행 조건

```
# define NDEBUG
# include <cassert>

long factorial(const int number) {
  assert (number >= 0 ) ;
  // 1만 비교하기 때문에 factorial( -10)을 호출하면 무한반복이 될 수 있다

  if ( number == 1 ) return 1 ;

  return number * factorial(number-1) ;
}

int findChar(const char *const str, const char ch) {
  // 선행조건; 기본적으로 포인터는 반드시 null이 아님이 확인되어야 함
  assert ( str != 0 ) ;

  for ( int i = 0 ; i < strlen(str) ; i ++ )
    if ( str[i] == ch ) return i ;

  return -1 ;
}
```

함수의 후행 조건

```
# define NDEBUG
# include <cassert>

void sortNumbers(vector<int>& numbers) {
   int size = numbers.size() ;
   for ( int i = 0 ; i < size - 1 ; i ++ )
      for ( int j = i + 1 ; j < size ; j ++ )
         if ( numbers[i] < numbers[j] ) swap(numbers[i], numbers[j]) ;

   // 후행조건 (이것도 2개 이상의 원소를 가정한 것임)
   for ( int i = 0 ; i <= numbers.size() - 2 ; i ++ )
      assert ( numbers[i] >= numbers[i+1] ) ;
}
```