인자 전달 방식:
Call by Value & Call by Reference

- Argument passing
  - call by value: does not modify the actual argument
  - call by reference: change the value of the actual argument

```
void f(int x, int& y) { // 형식 매개변수 x, y
        x++ ;
        y++ ;
}
int main() {
        int i = 10 ;
        int j = 20 ;

        cout << i << '₩t' << j << endl ; // 10 20
        f(i, j) ; // 실 매개변수 i, j
        cout << i << '₩t' << j << endl ; // 10 21
}
```

Call by Value & Call by Reference: An Example

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

void callByValue(int, int ) ;
void callByReference(int&, int&) ;

int main() {
   int number1=10, number2=20;

   int v1 = number1, v2=number2 ;
   callByValue(v1, v2) ;
   cout  << v1 << '\t' << v2 << endl ;

   int r1 = number1, r2=number2 ;
   callByReference(r1, r2) ;
   cout << r1 << '\t' << r2 << endl ;
}
```

```
void callByValue(int n1, int n2) {
   int temp = n1 ;
   n1 = n2 ;
   n2 = temp ;
}

void callByReference(int& n1, int& n2) {
   int temp = n1 ;
   n1 = n2 ;
   n2 = temp ;
}
```

실 매개변수 값의 변경
Pointer and Reference

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

void swapByPointer(int* n1, int* n2) {
   int temp = *n1 ;
   *n1 = *n2 ;
   *n2 = temp ;
}
void swapByReference(int& n1, int& n2)
{
   int temp = n1 ;
   n1 = n2 ;
   n2 = temp ;
}
```

```
int main() {
   cout << "Enter two integers !" << endl ;

   int number1, number2 ;
   cin >> number1 >> number2 ;

   swapByPointer(&number1, &number2) ;
   swapByReference(number1, number2) ;

   cout << number1 << '₩t'
      << number2 << endl ;
}
```

실 매개변수 값의 변경
Pointer and reference

	포인터 방식	참조 방식
형식 매개변수 선언	func (int* num)	func (int& num)
실 매개변수 접근 방법	*num = 10	num = 10
실 매개변수 전달 방법	int x =10; func (&x) ;	int x =10; func (x) ;

매개변수 전달 방법: 요약

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

void multiplyWithRef(
   int& number, int times) {
   number *= times ;
}

void multiplyWithValue(
   int number, int times) {
   number *= times ;
}

void multiplyWithPointer(
   int* number, int times) {
   *number *= times ;
}
```

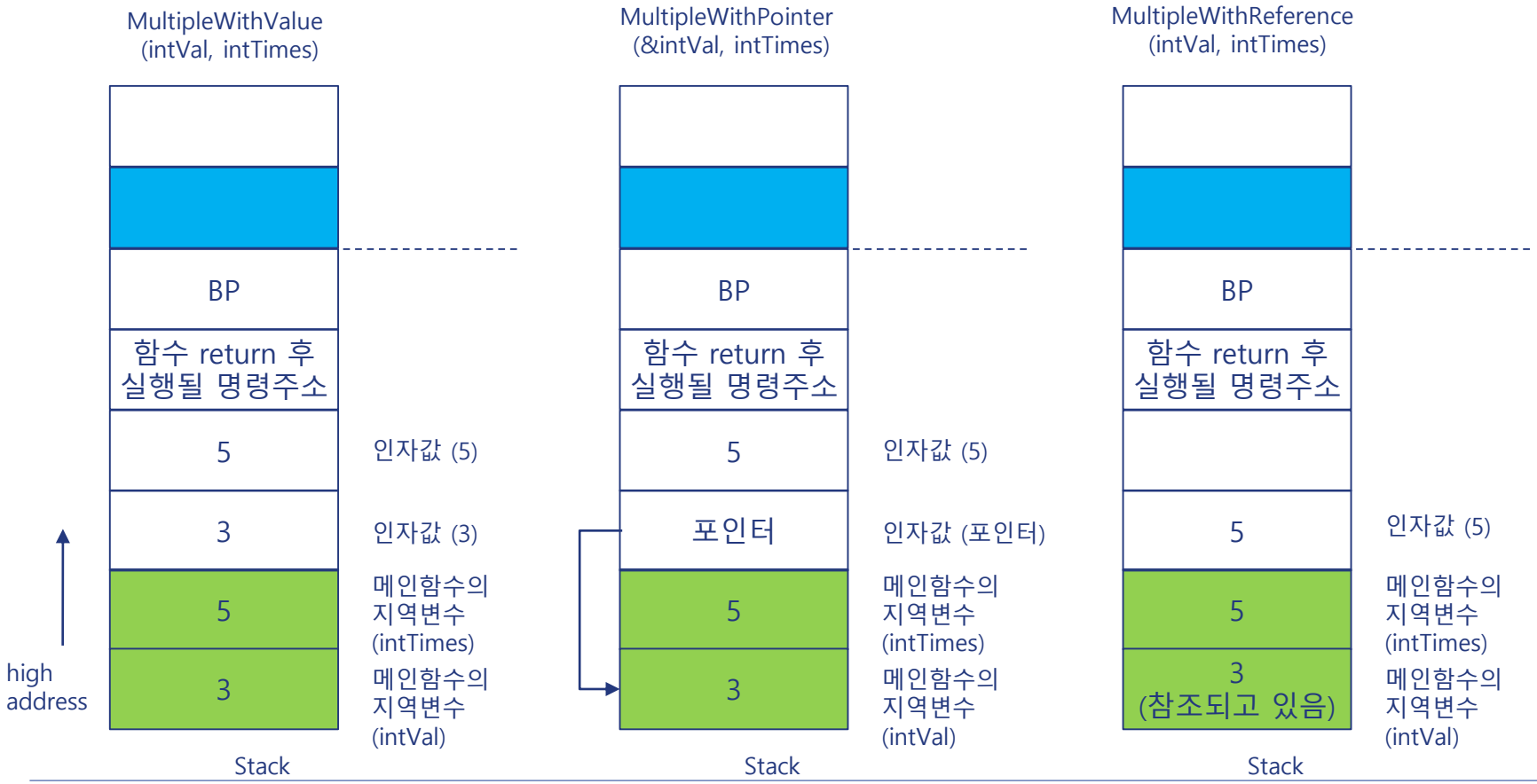
```
int main() {

   int intVal ;
   int intTimes ;
   cin >> intVal >> intTimes ;              //3, 5

   // 참조를 이용한 매개 변수 전달
   multiplyWithRef(intVal, intTimes) ;
   cout << intVal << endl ;

   // 일반 변수를 이용한 매개변수 전달
   multiplyWithValue(intVal, intTimes) ;
   cout << intVal << endl ;

   // 포인터를 이용한 매개변수 전달
   multiplyWithPointer(&intVal, intTimes) ;
   cout << intVal << endl ;
}
```

Call Stack: Pointer and Reference

```
void multiplyWithValue(
    int number, int times) {
    number *= times ;
}
```

```
void multiplyWithPointer(
    int* number, int times) {
    *number *= times ;
}
```

```
void multiplyWithRef(
    int& number, int times) {
    number *= times ;
}
```



인자 전달 방식: const
상수 매개변수

- 매개변수를 const로 선언함으로써 해당 매개변수의 값이 피호출함수에서 변경되는 것을 불허함
- 기본적으로 포인터와 참조는 const로 전달하라 (C++ Core Guidelines)

```
void f(int x, const int y) { // y는 상수 매개변수임
    cout << x << '₩t' << y << endl ;
    x++ ;
    y++ ;   // ERROR: y는 상수이므로 변경될 수 없음
}
```

https://www.cppkorea.org/CppCoreGuidelines/Const/#Rconst-ref

인자 전달 방식: const
상수 매개변수

const 미사용시	const 사용시

void print(char* str) {
  cout << str << endl ;
  str = "Hi";
}
int main() {
  char* msg = "Hello" ;
  print(msg) ; // Hello
  print(msg) ; // Hi
}
	void print(**const** char* str) {
	  cout << str << endl ;
	  strcpy(str, "Hi") ; // **ERROR**
	}
	int main() {
	  char* msg = "Hello" ;
	  print(msg) ; // Hello
	  print(msg) ; // Hello
	}

인자 전달 방식: const
상수 매개변수

To prevent the object from changing in the called function, declare it as "**const**"

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

struct Point { int x, y ; } ;
void readPoints(Point* const pts, const int size) ;
bool isEqual(const Point pt1, const Point pt2) ;
bool findPoint(const Point* const pts, const int size, const Point pt) ;
int main() {
  int no ;
  cin >> no ;
  Point* const pts = new Point[no] ;

  readPoints(pts, no) ;

  Point pt ;
  cin >> pt.x >> pt.y ;

  const string msg = findPoint(pts, no, pt) ? "Found." : "Not Found." ;
  cout << msg << endl ;
}
```

인자의 유형

유형	예
기본 타입	void print(**const float value**) ; bool isPositive(**const int n**) ;
나열형	enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ; void upgrade(**Grade&**) ;
배열	int findChar(**const char *const str**, const char ch) ;
구조체	struct Student { string name ; float gpa ; } ; float getGPA(**const Student&**) ;

인자: 나열형

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

enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ;

void print(const Grade grade) {
   string gradeLabels[] = { "Fresh", "Sophomore", "Junior", "Senior" } ;
   cout << gradeLabels[grade-1] << endl ;
}
void upgrade(Grade& now) {
   if ( now != SENIOR ) now = static_cast<Grade>(now+1) ;
}
int main() {
   Grade grade = FRESH ;
   print(grade) ;                              // “Fresh”
   upgrade(grade) ; print(grade) ;             // “Sophomore”
   upgrade(grade) ; print(grade) ;             // “Junior”
   upgrade(grade) ; print(grade) ;             // “Senior”
   upgrade(grade) ; print(grade) ;             // “Senior”
}
```

인자: 나열형의 배열

```
const int COUNT = 4 ;
enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ;
void upgrade(Grade& now) {
  if ( now != SENIOR ) now = static_cast<Grade>(now+1) ;
}
void print(const Grade grade) {
  const string gradeLabels[] =  { "Fresh", "Sophomore", "Junior", "Senior" } ;
  cout << gradeLabels[grade-1] << endl ;
}
void upgradeAll(Grade grades[], const int size) {
  for ( int i = 0 ; i < size ; i ++ ) upgrade(grades[i]) ;
}
void printAll(const Grade grades[], const int size) {
  for ( int i = 0 ; i < size ; i ++ ) print(grades[i]) ;
}
```

```
int main() {
  Grade grades[COUNT] = {FRESH, SOPHOMORE, JUNIOR, SENIOR} ;
  upgradeAll(grades, COUNT) ;
  printAll(grades, COUNT) ;
}
```

인자: 문자의 배열

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

int findChar(const char *const str, const char ch) ;

int main() {
   char str[] = "A string to be searched." ;
   cout << "Enter a character to find in " << str << endl ;
   char ch ;
   cin >> ch ;
   const int index = findChar(str, ch) ;
   if ( index == -1 )
      cout << ch << " isn\'t in " << str << endl ;
   else
      cout << ch << " is found at "<< index << endl ;
}

int findChar(const char *const str, const char ch) {
   for ( int i = 0 ; i < strlen(str) ; i ++ )
      if ( str[i] == ch ) return i ;
   return -1 ;
}
```

인자: 구조체

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

struct Point {
  int x, y ;
} ;

bool isEqual(const Point pt1, const Point pt2) ;

int main() {
   Point point1, point2 ;
   cin >> point1.x >> point1.y ;
   cin >> point2.x >> point2.y ;

   const string msg = isEqual(point1, point2) ?
      "Two points are equal." : "Two points are not equal." ;
   cout << msg << endl ;
}

bool isEqual(const Point pt1, const Point pt2) {
   return pt1.x == pt2.x && pt1.y == pt2.y ;
}
```

인자: 구조체의 배열

```
# include <string>
# include <iostream>
using namespace std ;
struct Point { int x, y ; } ;
void readPoints(Point* const pts, const int size) ;
bool isEqual(const Point pt1, const Point pt2) ;
bool findPoint(const Point* const pts, const int size, const Point pt) ;

int main() {
  int no ;
  cin >> no ;
  Point* const pts = new Point[no] ;

  readPoints(pts, no) ;

  Point pt ;
  cin >> pt.x >> pt.y ;

  const string msg
     = findPoint(pts, no, pt) ? "Found." : "Not Found." ;
  cout << msg << endl ;
  delete [] pts;
}
```

```
void readPoints(Point* const pts, const int size) {
  for ( int i = 0 ; i < size ; i ++ )
     pts[i].x = pts[i].y = i ;
}
bool isEqual(const Point pt1, const Point pt2) {
  return pt1.x == pt2.x && pt1.y == pt2.y ;
}
bool findPoint(const Point* const pts, const int size,
const Point pt) {
  for ( int i = 0 ; i < size ; i ++ )
     if ( isEqual(pts[i], pt) ) return true ;
  return false ;
}
```

Passing Large Object by Reference

- Call by reference is useful when a large object is passed

```
struct StudentInfo {
    string name ;
    int age ;
    int year ;
    float gpa ;
} ;

void printStudentInfo(StudentInfo& st) {
    cout << st.name << '\'t ;
    cout << st.age << '\t' ;
    cout << st.year << '\t' ;
    cout << st.gpa << endl ;
}
```

Passing Large Object by Const Reference

- To prevent the object from changing in the called function, declare it as "const"

```
struct StudentInfo {
    string name ;
    int age ;
    int year ;
    float gpa ;
} ;

void printStudentInfo(const StudentInfo& st) {
    cin >> st.name ; // error
    cout << st.name << '\'t ;
    cout << st.age << '\t' ;
    cout << st.year << '\t' ;
    cout << st.gpa << endl ;
}
```

구조체의 전달: 요약

```
struct Point {
  int x, y ;
} ;
void readPoint(Point& pt) {
      cin >> pt.x >> pt.y ;
}
void printPoint(const Point& pt) {
      cout << pt.x << '₩t' << pt.y << endl ;
}
bool isEqualPoint(const Point& pt1, const Point& pt2) {
      return pt1.x == pt2.x && pt1.y == pt2.y ;
}
int main() {
      Point point1, point2 ;
      readPoint(point1) ;
      readPoint(point2) ;

      printPoint(point1) ;
      printPoint(point2) ;

      const string msg = isEqualPoint(point1, point2) ? "Equal." : "Not equal." ;
      cout << msg << endl ;
}
```

구조체의 전달 요약

	형식 매개 변수 명시	예
in 매개변수 실매개변수의 변경 불가시	상수 참조 사용	printPoint(**const Point&**) ; isEqualPoint(**const Point&, const Point&**) ;
out 매개변수 실매개변수의 변경 허용시	참조 사용	readPoint(**Point&**) ;

인자 전달 방식: 요약

- **in 매개변수**: 호출자에서 전달된 매개변수의 값이 피호출자에서 변경되는 것을 허용하지 않는 경우
- **out 매개 변수**: 피호출함수에서의 변경이 호출함수의 실 매개변수에 반영되는 경우

유형	in 매개변수	out 매개변수
기본 또는 나열형	const T	T &
배열	상수에 대한 상수 포인터 const T* const	상수 포인터 T* const
구조체	const T &	T &

기본 매개변수 값

- 피호출 함수의 선언에서 생략된 매개변수에 대한 기본(default) 값을 미리 정의한 경우에만 실 매개변수 값의 생략이 가능

```
int multiplyBy(const int, const int = 1) ; // 2번째 매개변수의 기본 값은 1임
int main () {
  cout << multiply(10, 2) << endl ;  // 20
  cout << multiply(10) << endl ;     // 10
}
int multiplyBy(const int v1, const int v2) { return v1 * v2 ; }
```

기본 매개변수 값

- 뒤쪽의 매개변수부터 기본값을 차례로 지정

```
void f1(int a, int b, int c=5, int d=10) ;        // OK
void f2(int a, int b=5, int c, int d=10) ;        // ERROR
void f3(int a=10, int b, int c, int d) ;          // ERROR
```

- 기본 매개변수의 값은 오직 1회만 지정되어야 하며, 동일한 값이라 하더라도 여러 회 지정되는 것은 허용되지 않음

```
void scaleBy(Circle&, const float = 1.0F) ;       // OK
void scaleBy(Circle&, const float = 1.0F) ;       // ERROR
```

기본 매개변수 값

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

const float PI = 3.14F ;
struct Circle {
   int centerX, centerY ;
   float radius ;
} ;
void scaleBy(Circle& circle, const float ratio = 1.0F) ;
float getArea(const Circle& circle) ;

int main() {
   cout << "Enter the information of a circle: x y radius" << endl ;
   Circle circle ;
   cin >> circle.centerX >> circle.centerY >> circle.radius ;

   cout << "The area is " << getArea(circle) << endl ;
   scaleBy(circle, 2) ;
   cout << "The area is " << getArea(circle) << endl ;
   scaleBy(circle) ;
   cout << "The area is " << getArea(circle) << endl ;
}
void scaleBy(Circle& circle, const float ratio) { circle.radius *= ratio ; }
float getArea(const Circle& circle) { return PI * circle.radius * circle.radius ; }
```