Function Overloading

- Using the same name for operations on different types
  - void print(int) ;
  - void print(const char*) ;

```
void print(int) ;
void print(const char*) ;

int main() {
  print (10) ;
  print (“Hello !”) ;
}
```

C vs C++

허용되지 않을 때: C 언어	허용 될 때: C++ 언어
void printInt(int) ;	void print(int) ;
void printString(const char*) ;	void print(const char*) ;
void printDouble(double) ;	void print(double) ;
void printChar(char) ;	void print(char) ;

Overload Resolution

- To determine the right function, compiler tries to resolve the invocation in the following ways.
  1. Exact match
  2. Promotion
  3. **Conversion**
     - Match using standard conversinons
     - Match using user-defined conversions
     - Match using the ellipsis …

```
void print(char) { }
void print(int) { } ;
void print(double) { } ;

void h(char c, int i, short s, float f) {
    print(c) ;// exact match: invoke print(char)
    print(i) ;// exact match: invoke print(int)
    print(s) ;// promotion: invoke print(int)
    print(f) ;// promotion: invoke print(double)
}
```

```
int add (int a, int b) { return a+b; } ;
int add (float a, float b){return a+b;};

//void func(int a)  { } ;
void func(int& a) { } ; // 참조
void func(const int& a)  { } ; //참조

void h (int i, short s, float f) {
    add(i, i) ;
    add(f, f) ;
    //add(i, f) ; //ambiguous

    const int ci = 1;
    //func(1); //ambiguous
    func(i);
    func(ci);
}
```

https://en.cppreference.com/w/cpp/language/overload_resolution

Implicit Conversion

- Order of the conversions
  - standard conversion sequence
  - user-defined conversion
- Standard conversion sequence
  - lvalue-to-rvalue conversion, array-to-pointer conversion, and function-to-pointer conversion
  - **numeric promotion or numeric conversion**
  - function pointer conversion
  - qualification conversion
- Numeric promotions (int → long, float → double)
  - data loss 를 피하기 위해 더 큰 자료형으로 변환됨
- Numeric conversions
  - 큰 자료형 값이 더 작은 자료형으로 변환됨
  - 서로 다른 자료형 간에 변환됨
  - 산술연산자 (+, - , *, /, …)
    - long double부터 시작해서 int 순서로 둘 중 더 큰 자료형으로 변환
    - 그래도 다르면, int로 변환

https://en.cppreference.com/w/cpp/language/implicit_conversion

Overloading and Return Type

- Return types are not considered in resolution

```
void print(int) ;
int print(int) ; // error
```

```
float sqrt(float) ;
double sqrt(double) ;

int main() {
   float flt ;
   double dbl ;

   float    f1  = sqrt(flt) ;     // invoke sqrt(float)
   double d1  = sqrt(flt) ;     // invoke sqrt(float)
   float    f2  = sqrt(dbl) ;    // invoke sqrt(double)
   double d2  = sqrt(dbl) ;    // invoke sqrt(double) ;
}
```

Example

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

struct Point {
   int x, y ;
} ;
struct Rectangle {
   Point leftTop ;
   Point rightBottom ;
} ;

bool isEqual(const Point& pt1, const Point& pt2) ;
bool isEqual(const Rectangle& rect1, const Rectangle& rect2) ;
```

```
int main() {
  Point p1, p2 ;
  cin >> p1.x >> p1.y ;
  cin >> p2.x >> p2.y ;

  Rectangle r1, r2 ;
  r1.leftTop = r2.leftTop = p1 ;
  r1.rightBottom = r2.rightBottom = p2 ;

  cout << isEqual(p1, p2) << endl ;
  cout << isEqual(r1, r2) << endl ;
}

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

bool isEqual(const Rectangle& rect1, const Rectangle& rect2) {
  // isEqual(const Point&, const Point&)를 호출하므로 재귀적 호출이 아님
  return isEqual(rect1.leftTop, rect2.leftTop)
    && isEqual(rect1.rightBottom, rect2.rightBottom) ;
}
```