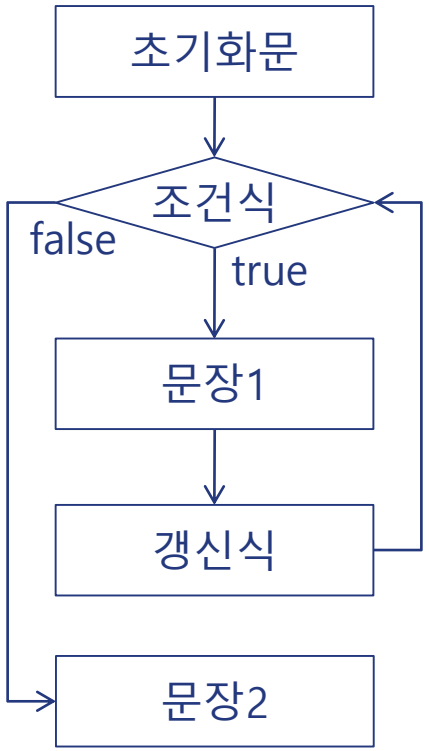
for 문

- 일정 횟수의 동작을 반복적으로 수행할 때 유용

```
int sum = 0 ;
for ( int i = 1 ; i <= 10 ; i ++ )
    sum += i ;
```

```
for ( 초기화문 ; 조건식 ; 갱신식 )
    문장1
문장2
```



for 문

- 배열을 접근할 때는 인덱스 변수는 0 부터 시작하고 배열의 크기보다 작은 조건을 사용

1차원 배열	2차원 배열

const int SIZE = 3 ;
 	
int intArray1[SIZE] = {0, 1, 2} ;
for (int **i = 0** ; **i < SIZE** ; i ++)
  intArray1[i] ++ ;
	const int ROW = 3 ;
	const int COLUMN = 4 ;
	
	int intArray2[ROW][COLUMN] ;
	for (int **i = 0** ; **i < ROW** ; i ++)
	  for (int **j = 0** ; **j < COLUMN** ; j ++)
	  intArray2[i][j] ++ ;

for 문

- vector 등과 같은 컬렉션의 각 원소를 접근

Iterator 사용 안 할 때	Iterator 사용할 때

vector<int> vInt(5) ;
for (int i = 0 ; i < vInt.size() ; i ++)
  cout << vInt[i] << endl ;
	vector<int> vInt(5) ;
	for (vector<int>::const_iterator
	  it = vInt.begin() ;
	  it != vInt.end() ; ++ it)
	  cout << *it << endl ;

Range-based for loop (Since C++11)

- used exclusively with ranges

```
for ( declaration : range ) statement;
```

```
vector<int> vInt{1, 2, 3}
for ( auto i = 0 ; i < vInt.size() ; i ++ )
    cout << vInt[i] << endl ;

for ( auto it = vInt.cbegin(); it != vInt.cend() ; ++ it )
    cout << *it << endl ;

for ( int v: vInt )
    cout << v << endl ;
```

Range-based for loop (Since C++11)

- used with array and string

```
int fibonacci[]{ 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 };
for (auto number : fibonacci)
    cout << number << endl;


string str {"Hello!"};
for (auto c : str)
    cout << "[" << c << "]";
```

Range-based for loop (Since C++11)

- With reference variable

```
int values[]{ 0, 1, 2, 3 };
for (auto& number : values)      // &가 없다면?
    number++;

for (const auto& number : values) // auto와의 차이점은?
    cout << number << endl;
```

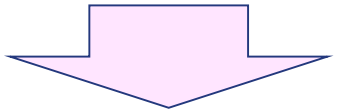
- What's the output when & is not used?

```
string strValues[]{"C++", "Is", "Wonderful"};
for (auto& str : strValues) {
    str += " OK";
    cout << str << endl;
}
for (const auto& str : strValues) // auto와의 차이점은?
    cout << str << endl;
```

Good Design

- for 문에서 콤마식의 사용은 자제

```
int sum, i ;
for (sum = 0, i = 1 ; i <= 10 && sum < 30 ;  i ++, cout << i << '₩t' )
  sum += i ;
// 2 3 4 5 6 7 8 9
```



```
int sum = 0 ;
for ( int i = 1 ; i <= 10 ; i ++ ) {
  if ( sum >= 30 ) break ;
  sum += i ;
  cout << i + 1 << '₩t' ;
}
```

Good Design : 선택할 수 있다면 for 문 보다는 range-based for 문을 선호하라! (ES.71)

- 가독성(Readability), 에러 예방(Error prevention), 효율성(Efficiency)
- 루프문 내에서 루프변수를 변경하는 경우가 없어야 한다
- 범위 기반 for 문에서 루프변수를 복사하지 마라

```
// bad
for (gsl::index i = 0; i < v.size(); ++i)
        cout << v[i] << '₩n';
// bad
for (auto p = v.begin(); p != v.end(); ++p)
        cout << *p << '₩n';

for (auto& x : v)     // OK
        cout << x << '₩n';

for (string s: vs) //bad, 루프변수 복사
        . . .

for (string& s : vs) //good

for (const string& s: vs)//수정되지 않으면
```

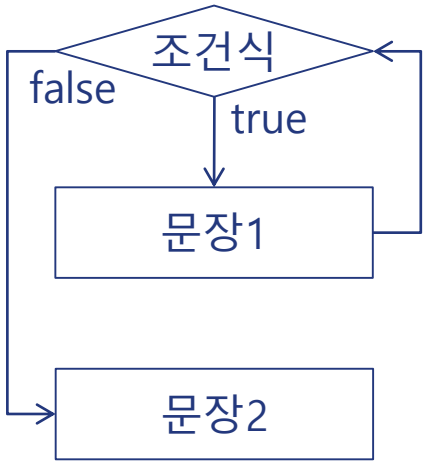
```
// touches two elements: can't be a range-for
for (gsl::index i = 1; i < v.size(); ++i)
        cout << v[i] + v[i - 1] << '₩n';

// possible side effect: can't be a range-for
for (gsl::index i = 0; i < v.size(); ++i)
        cout << f(v, &v[i]) << '₩n';

// body messes with loop variable:
// can't be a range-for
for (gsl::index i = 0; i < v.size(); ++i) {
        if (i % 2 == 0)
            continue;   // skip even elements
        else
            cout << v[i] << '₩n';
}
```

while 문

- 일반적인 형태로 반복을 표현
- 조건식이 참인 동안 반복

```
int sum = 0 ;
int i = 1 ;
while ( i <= 10 ) {
            sum += i ;
            i ++ ;
}
cout << sum ;
```

```
while ( 조건식 )
            문장1
문장2
```



while 문

for 문	while 문

for (초기화문 ; 조건식 ; 갱신식)
반복문
	초기화문;
	while (조건식) {
	반복문 ;
	갱신식 ;
	}

```
int sum = 0 ;
for ( int i = 1 ; i <= 10 && sum < 30 ; i ++ )
        sum += i ;
cout << sum ;
```

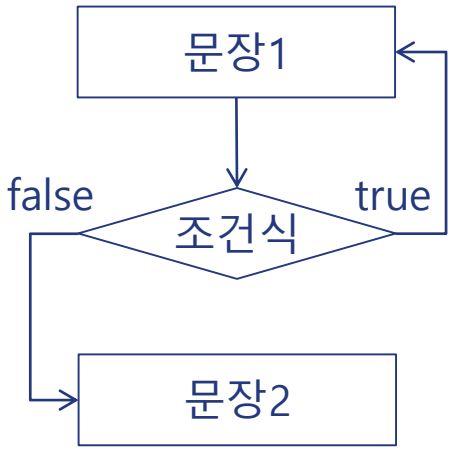
```
int sum = 0 ;
int i = 1 ;
while ( i <= 10 && sum < 30 ) {
        sum += i ;
        i ++ ;
}
cout << sum ;
```

do 문

- 조건식이 참인 동안 반복
- 최소 1회 이상은 수행

```
int sum = 0 ;
int i = 1 ;
do {
        sum += i ;
        i ++ ;
} while ( i <= 10 ) ;
cout << sum ;
```

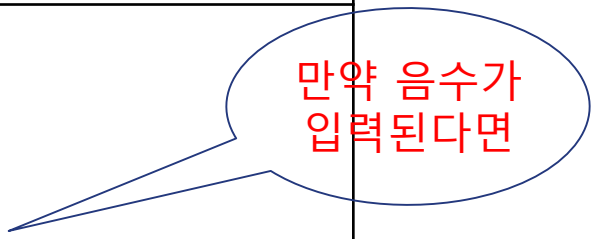
```
do
        문장1
while ( 조건식 ) ;
문장2
```



do 문

do 문	while 문

int sum = 0 ;
int limit ;
cin >> limit ;
do {
  int x ;
  cin >> x ;
  if (x <= 0) break ;
  sum += x ;
} while (sum <= limit) ;
cout << sum ;
	int sum = 0 ;
	int limit ;
	cin >> limit ;
	while (sum <= limit) {
	  int x ;
	  cin >> x ;
	  if (x <= 0) break ;
	  sum += x ;
	}
	cout << sum ;



continue

- 조건에 부합되지 않을 때는 이후 문장을 수행하지 않음

```
int sum = 0 ;
for ( int i = 1 ; i <= 10 ; i ++ ) {
    if ( i % 3 == 0 ) continue ;  // 3의 배수는 무시함
    sum += i ;
}
cout << sum ; // 37 = 55 - (3+6+9)
```

```
int sum = 0 ;
while ( sum <= 50 ) {
    int x ;
    cin >> x ;
    if ( x <= 0 ) continue ;  // 0 이하의 값이 입력되면 다음 값을 입력 받음
    sum += x ;
}
```

break

- switch 문에서 switch문을 벗어날 때
- 반복문에서 반복문을 벗어날 때

```
int sum = 0 ;
for ( int i = 1 ; i <= 10 ; i ++ ) {
    if ( i % 5 == 0 ) break ;  // i가 5일 때 for 반복문이 종료됨
    sum += i ;
}
cout << sum ; // 10 ( =1 + 2 + 3 + 4)
```

```
int sum = 0 ;
while ( sum <= 50 ) {
    int x ;
    cin >> x ;
    if ( x <= 0 ) break ;  // 0 이하의 값이 입력되면 while 반복이 종료됨
    sum += x ;
}
```

return

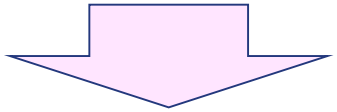
- 함수의 수행을 종료

```
int main() {
    int sum = 0 ;
    for ( int i = 1 ; i <= 10 ; i ++ ) {
        if ( i % 5 == 0 ) return 0 ;  // main() 함수의 수행이 종료됨
        sum += i ;
    }
    cout << sum ; // Not reached!
}
```

Good Design

- return 문은 함수의 마지막 문장으로 1회만 사용

```
int compare(int x, int y ) {
    if ( x > y ) return -1 ;
    if ( x == y ) return 0 ;
    return 1 ;
}
```



```
int compare(int x, int y ) {
    int result ;
    if ( x > y ) result = -1 ;
    else if ( x == y ) result = 0 ;
    else result = 1 ;
    return result ;
}
```

반복문: for, while, do

```
int main() {
  int no ;
  cin >> no ;
  vector<int> numbers(no) ;
  for ( int i = 0 ; i < no ; i ++ )
    cin >> numbers[i] ;
}
```

```
int main() {
  int no ;
  cin >> no ;
  vector<int> numbers(no) ;
  int i = 0 ;
  do {
     cin >> numbers[i] ;
     i ++ ;
  } while ( i < no )
}
```

```
int main() {
  int no ;
  cin >> no ;
  vector<int> numbers(no) ;
  int i = 0 ;
  while ( i < no ) {
    cin >> numbers[i] ;
    i ++ ;
  }
}
```

분기: continue, break, return

```
#include <vector>
#include <iostream>
using namespace std ;
int sum1(const vector<int>& intArray) ;
int sum2(const vector<int>& intArray) ;
int sum3(const vector<int>& intArray) ;
int sum4(const vector<int>& intArray) ;

int main() {
   vector<int> intArray{10, -10, 20, -20, 30}; // Since C++11
   cout << sum1(intArray) << endl ;
   cout << sum2(intArray) << endl ;
   cout << sum3(intArray) << endl ;
   cout << sum4(intArray) << endl ;
}
int sum1(const vector<int>& intArray) {
   int sum = 0 ;
   for ( int i = 0 ; i < intArray.size() ; i ++ )
      sum += intArray[i] ;
   return sum + intArray[0] ;
}
```

```
int sum2(const vector<int>& intArray) {
   int sum = 0 ;
   for ( int i = 0 ; i < intArray.size() ; i ++ ) {
      if ( intArray[i] < 0 ) continue ;
      sum += intArray[i] ;
   }
    return sum + intArray[0] ;
}
int sum3(const vector<int>& intArray) {
   int sum = 0 ;
   for ( int i = 0 ; i < intArray.size() ; i ++ ) {
      if ( intArray[i] < 0 ) break ;
      sum += intArray[i] ;
   }
    return sum + intArray[0] ;
}
int sum4(const vector<int>& intArray) {
   int sum = 0 ;
   for ( int i = 0 ; i < intArray.size() ; i ++ ) {
      if ( intArray[i] < 0 ) return sum ;
      sum += intArray[i] ;
   }
    return sum + intArray[0] ;
}
```

Good Design: 제어문을 최소화 하자!

- 코드를 복잡하게 만드는 원인으로 if 조건문, for 반복문 등의 제어문이 자주 지목됨
- C++의 STL 알고리즘 함수를 효과적으로 사용해야 함

```
if (x > 10)
    x = 10;
```

```
x = std::min(x, 10);
```

```
if (x < 0)
    x = 0;
```

```
x = std::max(x-1, 0);
```

```
for(auto it = vec.begin(); it != vec.end(); ++it)
        it->doSomething();
```

```
std::for_each(begin(vec), end(vec), [] (T* t) { t->doSomething(); } );
```

```
auto find_it = end (vec);
for(auto it = vec.begin(); it != vec.end(); ++it)
        if ( (*it).name == "student")
                find_it = it; break;
if (find_it != end(vec)
    find_it->doSomething();
```

```
auto find_it = find(begin(vec), end(vec),
      [ ] (T* t) { return t->name() =="Jane"; });

if (find_it != end(vec)
        find_it->doSomething();
```

Q&A