Topic 3
Control Structures

Selections: if, switch

Iterations: for, while, do

Other controls: continue, break, return

if 문

- 조건에 따라서 2개 이상의 위치로 제어를 분기

```
int x ;
cin >> x ;
if ( x >= 0 )
    cout << "Zero or Positive number" << endl ;
else
    cout << "Negative number" << endl ;
```

if 문

- 복수 문장의 경우 { } 으로 블록을 만든다.

```
int x ;
cin >> x ;
if ( x > 0 ) {
    int y ;
    cin >> y ;
    cout << x + y << endl ;
}
else {
    cout << x << endl ;
}
```

if 문

- 여러 가지 경우에 가능한 경우에는 else if 사용

```
int score ;
cin >> score ;
char grade ;
if ( score >= 90 ) grade = 'A' ;
else if ( score >= 80 ) grade = 'B' ;
else if ( score >= 70 ) grade = 'C' ;
else if ( score >= 60) grade = 'D' ;
else grade = 'F' ;
cout << grade << endl ;
```

if 문

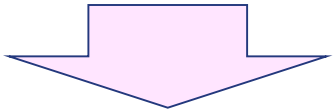
- if 문을 중첩시킴으로써 복잡한 상황을 표현

```
int x, y, z ;
cin >> x >> y >> z ;
if ( x > 0 && y > 0 && z > 0 ) {
   if ( x == y && x == z ) { // 중첩된 if 문
      cout << "정삼각형임" << endl ;
   }
   else {
      cout << "정삼각형이 아님" << endl ;
   }
}
else
   cout << "음수의 값은 허용되지 않음" << endl ;
```

Good Design

- if 문 사용시 else의 정의

```
void evaluate(int score) {
    if ( score >= 60 )
        cout << "Pass" << endl ;
}
```

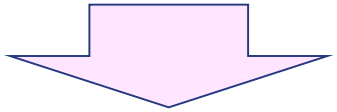


```
void evaluate(int score) {
    if ( score >= 60 )
        cout << "Pass" << endl ;
    else
        cout << "Fail" << endl ;
}
```

Good Design

- if 문 사용시 then-else의 쌍은 일치가 필요

```
int x = -10 ;
if ( x >= 0 )
    if ( x >= 50 )
        cout << "1" << endl ;
else // if ( x >= 0 )가 아니라 if ( x >= 50 )과 일치되어야 함
    cout << "2" << endl ;
```



```
int x = -10 ;
if ( x >= 0 ) {
    if ( x >= 50 ) {
        cout << "1" << endl ;
    } else {
        cout << "2" << endl ;
    }
}
```

Good Design

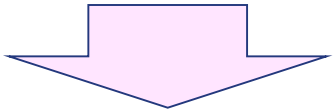
- 조건의 값이 항상 true/false가 되어서는 안 됨

```
if ( x >= 0 ) {
    if ( x < -10 ) {
        cout << "1" << endl ;
    } else {
        cout << "2" << endl ;
    }
}
```

Good Design

- 조건에서 && 및 || 의 피연산자는 괄호를 사용

```
if ( x + y == 0 || x – y == 0 && ok ) {
    …
}
```



```
if ( ( x + y == 0 ) || ( ( x – y == 0 ) && ok ) ) {
    …
}
```

Good Design

- 실수 값에 대해서 동등 비교를 권장하지 않음

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

int main() {
    double v1 = 0.2;
    double v2 = 1 / sqrt(5.0) / sqrt(5.0);

    if ( v1 == v2 )
        cout << "v1 == v2₩n";
    else
        cout << "v1 != v2₩n";
}
```

Good Design

- 가독성과 유지보수성을 고려하여 조건식은 함수로 구현

```
enum Month {JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC} ;
float getDiscountPrice(float price, Month m) {
      if ( m >= MAR && m <= MAY ) return price * 0.7 ;
      else if ( m >= SEP && m <= NOV ) return price * 0.8 ;
      else return price ;
}
```

```
enum Month {JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC} ;
bool isSpring(Month m) { return m >= MAR && m <= MAY ; } // 호주라면?
bool isFall(Month m) { return m >= SEP && m <= NOV ; } // 호주라면?
float getDiscountPrice(float price, Month m) {
      if ( isSpring(m) ) return price * 0.7 ;
      else if ( isFall(m) ) return price * 0.8 ;
      else return price ;
}
```

Good Design

- 가독성과 유지보수성을 고려하여 조건식은 함수로 구현

```
while ( true ) {
    cout << "Enter width and height: " ;
    int width, height ;
    cin >> width >> height ;
    // 너비 또는 높이가 음수이면 사각형 입력 종료; 만약 0에서 100으로 제한된다면?
    if ( width <= 0 || height <= 0 ) break ;

    cout << width << ' ' << height << endl ;
}
```

```
bool isInvalidRectangle(const int w, const int h) { return w <= 0 || h <= 0 ; }

while ( true ) {
    ...
    if ( isInvalidRectangle(width, height) ) break ;
    ...
}
```

조건식(conditional expression)

- if-then-else는 조건식(conditional expression)으로 간단하게 표현

if-then-else 사용	조건식 사용

int max ;
if (a >= b)
max = a ;
else
max = b ;
	int max = (a >= b) ? a : b ;

조건식(conditional expression)

- 프로그램의 가독성이 높일 수 있는 경우에 사용

```
int x ;
cin >> x ;
if ( x >= 0 )
  cout << "Zeror or Positive number" << endl ;
else
  cout << "Negative number" << endl ;
```

```
int x ;
cin >> x ;

string msg = (x >= 0) ? "Zero or Positive number" : "Negative number" ;
cout << msg << endl ;
```

```
int x ;
cin >> x ;

cout << ( (x >= 0) ? "Zero or Positive number" : "Negative number" ) << endl ;
```

조건식(conditional expression)

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

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

    int n1, n2 ;
    cin >> n1 >> n2 ;

    int max1 ;
    if ( n1 > n2 ) max1 = n1 ;
    else max1 = n2 ;

    int max2 = ( n1 > n2 ) ? n1 : n2 ;

    assert (max1 == max2) ;
}
```

switch

- 동일한 기준에 따른 복수개의 분기 상황을 표현

```
char shapeCode ;
cin >> shapeCode ;
switch ( shapeCode ) {
    case 'R' : cout << "Rectangle" << endl ; break ;
    case 'C' : cout << "Circle" << endl ; break ;
    default : cout << "Unknown Code" << endl ; break ;
}
```

switch

- 각 case를 블록으로 정의하여 지역 변수를 정의

```
const float PI = 3.14F ;
char shapeCode ;
cin >> shapeCode ;
float area = 0 ; // switch 내부의 case ‘R’, case ‘C’, default에서 사용 가능
switch ( shapeCode ) {
    case 'R' : {
            int width, height ; // case ‘R’ 블록의 지역변수
            cin >> width >> height ;
            area = width * height ;
        }
        break ;
    case 'C' : {
            int radius ; // case ‘C’ 블록의 지역변수
            cin >> radius ;
            area = PI * radius * radius ;
        }
        break ;
    default: cout << "Unknown Code" << endl ; break ;
}
cout << "Area: " << area << endl ;
```

switch

- case를 복수 개 나열하여 동일한 처리 가능

```
const float PI = 3.14F ;
cin >> shapeCode ;
float area = 0 ;
switch ( shapeCode ) {
   case 'R' : // 다음의 'r'과 동일한 동작을 위해서 의도적으로 break를 생략함
   case 'r' : {
         int width, height ;
         cin >> width >> height ;
         area = width * height ;
      }
      break ;
   case 'C' : // 다음의 'c'과 동일한 동작을 위해서 의도적으로 break를 생략함
   case 'c' : {
         int radius ;
         cin >> radius ;
         area = PI * radius * radius ;
      }
      break ;
   default: cout << "Unknown Code" << endl ; break ;
}
cout << "Area: " << area << endl ;
```

switch

- switch 문은 나열형의 변수와 함께 사용

```
enum ShapeKind {RECTANGLE, CIRCLE, UNDEFINED} ;

int main() {
   cout << "Enter the type of a shape !" << endl ;
   char shapeCode ;
   cin >> shapeCode ;

   ShapeKind kind = UNDEFINED ;
   if ( shapeCode == 'c' || shapeCode == 'C' ) kind = CIRCLE ;
   else if ( shapeCode == 'r' || shapeCode == 'R' ) kind = RECTANGLE ;

   switch ( kind ) {
       case CIRCLE : ... break ;
       case RECTANGLE : ... break ;
       default:
           cout << "Invalid shape type !" << endl ;
           break ;
   }
}
```

switch

- 실수 값 또는 범위의 값을 바탕으로 분기 표현 불가

실수 값에 의한 분기	범위에 따른 분기

float f ;
switch (f) {
  case 0.1 : … break ;
  case 0.2 : … break ;
  default: … break ;
}
	int n ;
	switch (n) {
	  case n > 0 && n < 10: … break ;
	  case n > 10 : … break ;
	  default: … break;
	}

실수 값에 의한 분기	범위에 따른 분기
float f ; if (f == 0.1) … else if (f == 0.2) … else …	int n ; if (n > 0 && n < 10) … else if (n > 10) … else …

Good Design

- switch 문은 반드시 마지막에 default가 정의

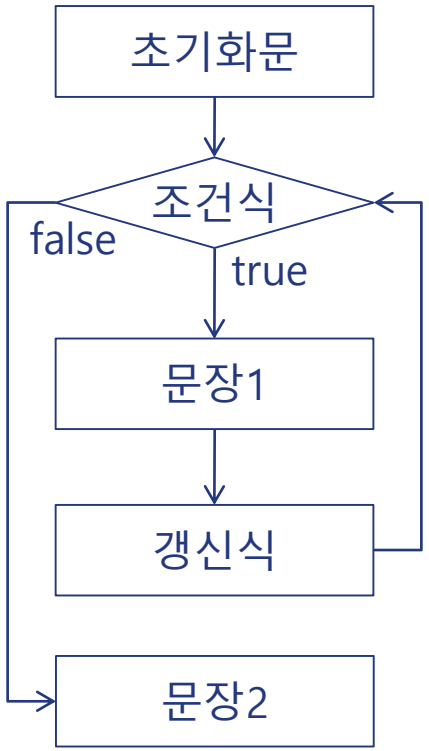
```
enum Operator { PLUS, MINUS, MULTIPLY, DIVIDE } ;

int main() {
    int x, y ;
    Operator op = getOperator() ;
    int result ;
    switch ( op ) {
      case PLUS : result = x + y ; break ;
      case MINUS : result = x – y ; break ;
      default: cout << “Operator Not Supported₩n” break ;
    }
}
```

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

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

```
int sum = 0 ;
for ( int i = 1 ; i <= 10 && sum < 30 ; i ++ )
  sum += i ;
cout << sum ; // 36
```

```
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
```

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

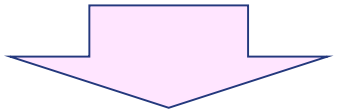
Iterator 사용 안 할 때	Iterator 사용할 때

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



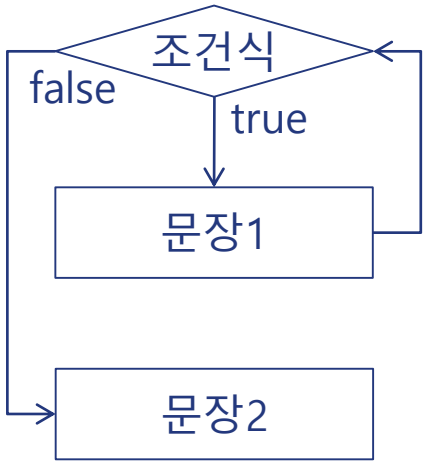
```
int sum = 0 ;
for ( int i = 1 ; i <= 10 ; i ++ ) {
  if ( sum >= 30 ) break ;
  sum += i ;
  cout << i + 1 << '₩t' ;
}
```

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 ;
```

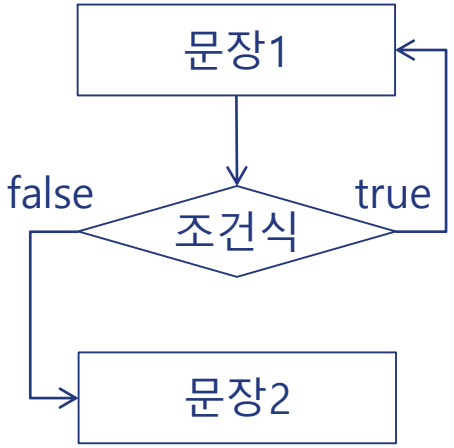
```
int sum = 0 ;
int i = 1 ;
while ( i <= 10 && sum < 30 ) {
        sum += i ;
        i ++ ;
}
cout << sum ;
```

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 ;
```

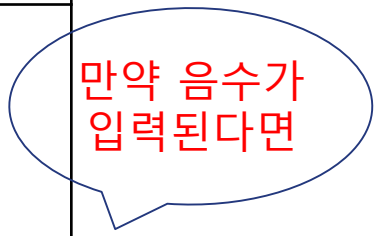
```
do
        문장1
while ( 조건식 ) ;
문장2
```



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 ;  // 음수가 입력되면 while 반복이 종료됨
    sum += x ;
}
```

return

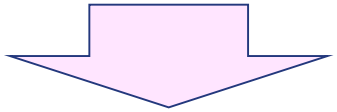
- 함수의 수행을 종료

```
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 ;
}
```

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 ;
    intArray.push_back(10) ;
    intArray.push_back(20) ;
    intArray.push_back(-10) ;
    intArray.push_back(-20) ;
    intArray.push_back(30) ;
    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] ;
}
```