Topic 3
Control Structures

Selections: if, switch

Iterations: for, while, do

Other controls: continue, break, return

Expression

- An expression is a sequence of operators and their operands, that specifies a computation
  - **expression evaluation may produce a result** and may generate side-effects
  - value categories (lvalue, rvalue, glvalue, prvalue, xvalue)
  - order of evaluation of arguments and subexpressions

```
2 + 2                          // 4
std::cout << 4                 //print the character '4'

// lvalue: 변수, 함수명, 함수 호출, 대입, *a, a->m, 문자 리터럴 등
// prvalue: 리터럴, non-reference를 반환하는 함수 호출, 연산식, 람다, this 등
// rvalue: prvalue or xvalue, rvalue는 & 연산자로 주소를 가질 수 없음, 할당 x
// xvalue: rvalue 참조를 반환하는 함수 호출 (std::move), a ? b : c
// glvalue: lvalue or xvalue

// return a() + b() + c();        //함수 호출 순서는?
```

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

Statement

- Statements are fragments of the C++ program that are executed in sequence
- Simple C++ statement is each of the individual instructions of a program
  - always end with a semicolon (;)
  - executed in the same order in which they appear in a program
- C++ provides **flow control statements(selection,iteration,jump)**
- Compound statement is a group of statements, but all grouped together in a block, enclosed in curly braces ({})

```
std::cout << "hello, world";                    //simple statement
// if (condition) statement                     //condition is the expression
// while (expression) statement
// for (initialization; condition; increase) statement;
if (x == 100) {                                 //compound statement
   cout << "x is ";
   cout << x;
}
```

https://www.cplusplus.com/doc/tutorial/control/

Types of Statement

- C++ includes the following types of statements

types	examples
expression statements	expression ;
compound statements	{ statement... }
selection statements	**if** statement, **switch** statement
iteration statements	**while** loop, **for** loop, range for loop
jump statements	**break**, **continue**, and return statement
declaration statements	block-declaration
try blocks	try/catch
atomic and synchronized blocks	synchronized, atomic_commit

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

표현식과 문장 in C++

- 프로그램의 흐름은 일반적으로 문장을 통해 제어함

표현식 (표현식은 재귀가 가능)	문장 (끝이 세미콜론(;)으로 끝남)
• 변수명, 상수, 리터럴: a, b, 3, 3.14f 등 • 연산자로 결합된 하나 이상의 표현식 (변수명, 상수, 리터럴): a+b, a+3, a+b+3 등 • 할당문: a=3, a=b, a=b=3+3 등 • 표현식을 인자로 넣은 함수 호출: f(a), f(a+b), f(a=3), f(f(a), b) 등 • 괄호로 둘러싸인 표현식: (a+b), (a+f(b)) 등	• a = 3; • a = b + 3; • a = f(a) * 3; • x + y; • ; • 함수, 클래스 정의 (definition) • { }-복합문 (compound statement) • 변수와 상수 선언(declaration) • 함수 및 클래스 정의 • **제어문 (if, for, while, switch 등)**

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

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 ;
```

operator	description
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Operators

Common Operators

operators	examples
assignment	a = b, a += b 등
increment, decrement	++a, --a, a++, a--
arithmetic	a + b, a / b, a << b, a & b 등
logical	!a, a && b, a || b
comparison	a == b, a != b, a > b 등
member access	a[b], *a, &a, a->b, a.b, a->*b, a.*b
other	**함수호출: f(), 콤마: a, b, 조건: a ? b : c**

Special Operators

- 형변환: static_cast, dynamic_cast, const_cast, reinterpret_cast, C-style cast 등
- 메모리 할당: new, delete
- 그 외: sizeof, typeid, alignof, noexcept

Program Flow Operators

- 프로그램을 제어하는 세 가지 연산자
  - 함수호출 (function call): C++에서는 연산자로 취급
  - 조건 연산자 (conditional operator): ***x ? a : b***, 조건 x 를 만족하면 표현식 값은 a값을 가지고, 아니면 b의 값을 가짐
  - 콤마 연산자 (comma operator): ***E1, E2***, 콤마의 왼쪽에 있는 E1를 먼저 계산한 뒤 오른쪽에 있는 E2를 계산하며, 표현식 값은 E2의 값임 (연산자 우선순위 중 가장 낮음)

```
add ( f(x), g(x) );
if ( f(x) );

int x = 3;
int a = x < 3 ? "yes" , "no";

int n = 1;
int m = (++n, std::cout << "n = " << n << '₩n', ++n, 2*n);  //n=2
std::cout << "m = " << (++m, m) << '₩n';                    //m=7
```

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

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 ;
```

if 문

- Short-circuit evaluation
  - A && B : if A is false, B is not evaluated
  - A || B : if A is true, B is not evaluated

```
int main() {
    int x = 0, y = 0;
    if ( x == 0 || ++y >= 0 )              // ++y >= 0 not evaluated
        x++;
    cout << x << ", " << y << endl;

    if ( x == 0 && ++y >= 0 )              // ++y >= 0 not evaluated
        x++;
    cout << x << ", " << y << endl;
}
```

- Eager evaluation: evaluate both A and B
  - A & B
  - A | B

Alternative operator representation

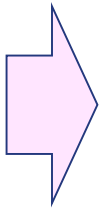
- C++ (and C) source code may be written in any non-ASCII 7-bit character set that includes the ISO 646:1983 invariant character set.
- Several C++ operators and punctuators require characters that are outside of the ISO 646 codeset: **{**, **}**, **[**, **]**, **#**, **₩**, **^**, **|**, **~**.
- To be able to use character encodings, C++ defines the alternatives composed of ISO 646 compatible characters.

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

Primary	Alternative
&&	and
&=	and_eq
&	bitand
|	bitor
~	compl
!	not
!=	not_eq
||	or
|=	or_eq
^	xor
^=	xor_eq
{	<%
}	%>
[	<:
]	:>
#	%:
##	%:%:

Alternative operator representation

```
int main() {
    int x, y;
    cin >> x >> y;
    if ( x >= 0 && y >= 0 )
        cout << "Both" << endl ;
    else if ( x >= 0 && ! y >= 0 )
        cout << "X" << endl ;
}
```

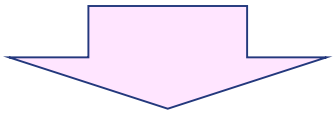


```
int main() {
    int x, y;
    cin >> x >> y;
    if ( x >= 0 and y >= 0 )
        cout << "Both" << endl ;
    else if ( x >= 0 and not y >= 0 )
        cout << "X" << endl ;
}
```

Good Design

- if 문 사용시 else의 정의

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

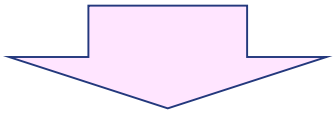


```
void evaluate(int score) {
    if ( score >= 60 )
        cout << "Pass" << endl ;
    else
        cout << "Fail" << endl ;
}
```

Good Design

- if 문 사용시 then-else의 쌍은 일치가 필요

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



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

Good Design

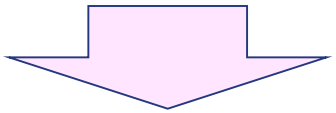
- 조건의 값이 항상 true/false가 되어서는 안 됨

```
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 ; }
bool isFall(Month m) { return m >= SEP && m <= NOV ; }
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 << "Zero 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 0.1F : … break ;
  case 0.2F : … 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 == 0.1F) … else if (f == 0.2F) … 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 ;
    cin >> x >> y;
    const 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 ;
    }
}
```