3장 데이터 처리

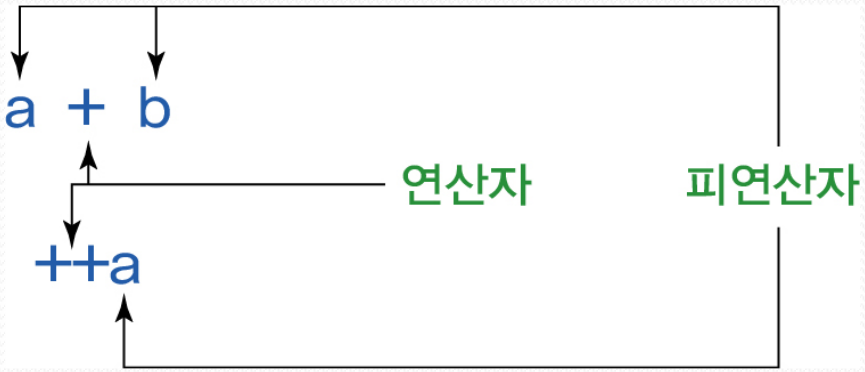
1. 연산자
2. 선택 제어문
3. 반복 제어문

학습목표

- 기본적인 연산자를 사용할 수 있다.
- if, if ~ else, switch 등의 선택문을 사용할 수 있다.
- for, while, do ~ while 등의 반복문을 사용할 수 있다.
- break, continue 등을 사용할 수 있다.

연산자의 종류

- 연산자(operator)
  - C++ 프로그램 내에서 연산 동작을 수행하는 기호
- 피연산자(operand)
  - 연산에 사용되는 데이터



- 피연산자의 개수
  - 단항 연산자(unary operator)
  - 이항 연산자(binary operator)
  - 3항 연산자(ternary operator)

연산자의 종류

구 분		연산자
산술 연산자	이항 연산자	+, -, *, /, %
	단항 연산자	++, --, -
대입 연산자		=, +=, -=, *=, /=, %=, <<=, >>=, &=, |=, ^=
관계 연산자		>, <, ==, !=, >=, <=
논리 연산자		&&, ||, !
비트처리 연산자	비트 논리 연산자	&, |, ^, ~
	shift 연산자	<<, >>
형 변환 연산자		(type)
기타 연산자	조건 연산자	? :
	콤마 연산자	,
	데이터 크기 연산자	sizeof
	주소 연산자	&
	간접 연산자	*
	멤버 참조 연산자	., ->

산술 연산자

- 이항 산술 연산자

연산자	의미	사용 예	설명
+	더하기	b + c	b와 c를 더한다.
-	빼기	b - c	b에서 c를 뺀다.
*	곱하기	b * c	b와 c를 곱한다.
/	몫 (정수) 나누기(실수)	b / c	(정수) b를 c로 나누었을 때 몫을 구한다. (실수) b를 c로 나눈다.
%	나머지(정수)	b % c	b를 c로 나누었을 때 나머지를 구한다.

- 단항 산술 연산자

연산자	의미	사용 예	설명
++	1 증가(increment)	b = a++ b = ++a	후행 연산; b = a, a = a+1 선행 연산; a = a+1, b = a+1
--	1 감소(decrement)	b = a-- b= --a	후행 연산; b = a, a = a-1 선행 연산; a = a-1, b = a-1
-	부호 반전	b = -a	b는 a의 부호를 반전한 값이다.

대입 연산자

- 대입 연산자(=)의 의미
  - (수학) 같다.
  - (C++) 대입 연산자.
    - 오른쪽에 있는 연산의 결과를 왼쪽에 있는 변수의 자료형에 맞추어 대입시킨다.

연산자	사용 예	설명
=	a = b	b를 a에 대입
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b

연산자	사용 예	설명
<<=	a <<= b	a = a << b
>>=	a >>= b	a = a >> b
&=	a &= b	a = a & b
|=	a |= b	a = a | b
^=	a ^= b	a = a ^ b

대입 연산자

- ex3_1.cpp (대입 연산자)

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

void main()
{
	int a, b, c, d, e;

	a = b = c = d = e = 10;
	cout << "a = b = c = d = e = " << a << endl;
	a += 6;
	cout << "a += 6 --> a = " << a << endl;
	b -= 6;
	cout << "b -= 6 --> b = " << b << endl;
	c *= 6;
	cout << "c *= 6 --> c = " << c << endl;
	d /= 6;
	cout << "d /= 6 --> d = " << d << endl;
	e %= 6;
	cout << "e %= 6 --> e = " << e << endl;
}
```

```
a = b = c = d = e = 10
a += 6 --> a = 16
b -= 6 --> b = 4
c *= 6 --> c = 60
d /= 6 --> d = 1
e %= 6 --> e = 4
계속하려면 아무 키나 누르십시오 . . .
```

관계 연산자

- 관계 연산자(relational operator)
  - 두 개의 피연산자를 비교하여 서로간의 크고, 작고, 같은 관계를 비교하여 결과를 참, 거짓으로 나타내는 연산자
  - 결과 값은 참이면 true(1), 거짓이면 false(0)으로 표현
  - 문자 사이의 관계는 ASCII 코드 값으로 비교한다.

연산자	의미	사용 예	설명
>	보다 크다	a = (b>c)	b가 c보다 크면 a = 1, 그렇지 않으면 a = 0 이 된다.
<	보다 작다	a = (b<c)	b가 c보다 작으면 a = 1, 그렇지 않으면 a = 0 이 된다.
==	같다	a = (b==c)	b와 c가 같으면 a = 1, 그렇지 않으면 a = 0 이 된다.
>=	크거나 같다	a = (b>=c)	b가 c보다 크거나 같으면 a = 1, 그렇지 않으면 a = 0.
<=	작거나 같다	a = (b<=c)	b가 c보다 작거나 같으면 a = 1, 그렇지 않으면 a = 0.
!=	같지 않다	a = (b!=c)	b와 c가 같지 않으면 a = 1, 그렇지 않으면 a = 0.

관계 연산자

- ex3_2.cpp (관계 연산자)

```
a > b --> 0
a < b --> 1
a == b --> 0
a != b --> 1
a <= b --> 1

e > f --> 0
e < f --> 0
e == f --> 1
e != f --> 0
계속하려면 아무 키나 누르십시오 . . .
```

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

void main()
{
        int a = 3, b = 5;
        char e = 'E', f = 'E' ;
        cout << "a > b --> " << (a > b) << endl;
        cout << "a < b --> " << (a < b) << endl;
        cout << "a == b --> " << (a ==b) << endl;
        cout << "a != b --> " << (a != b) << endl;
        cout << "a <= b --> " << (a <= b) << endl << endl;

        cout << "e > f --> " << (e > f) << endl;
        cout << "e < f --> " << (e < f) << endl;
        cout << "e == f --> " << (e == f) << endl;
        cout << "e != f --> " << (e != f) << endl;
}
```

논리 연산자

- 논리 연산자(logical operator)
  - AND, OR, NOT을 수행하여 연산 결과를 참, 거짓으로 나타내는 연산자
  - 피연산자가 0이면 거짓, 0이 아니면 참으로 취급
  - 결과 값은 참이면 true(1), 거짓이면 false(0)으로 표현된다.

연산자	의미	사용 예	설명
&&	AND	a = b && c	b와 c가 모두 참이면 a = true(1), 그렇지 않으면 a = false(0)가 된다.
||	OR	a = b || c	b와 c 어느 하나라도 참이면 a = true(1), 그렇지 않으면 a = false(0)가 된다.
!	NOT	a = !b	! 연산자는 단항 연산자이다. b가 참이면 a = true(1)가 되고, b가 거짓이면 a = false(0)가 된다.

논리 연산자

- ex3_3.cpp (논리 연산자)

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

void main()
{
	int a = 3, b = -5, c = 0;
	bool x;

	x = a && b;
	cout << "x = a && b --> x = " << x << endl;
	x = a && c;
	cout << "x = a && c --> x = " << x << endl;
	x = a || b;
	cout << "x = a || b --> x = " << x << endl;
	x = !c;
	cout << "x = !c --> x = " << x << endl;
}
```

```
x = a && b --> x = 1
x = a && c --> x = 0
x = a || b --> x = 1
x = !c --> x = 1
계속하려면 아무 키나 누르십시오 . . .
```

비트 처리 연산자

- 비트 처리 논리 연산자
  - 두 데이터의 비트끼리의 논리 연산을 수행

연산자	의미	사용 예	설명
&	bitwise AND	x = a & b	a와 b의 비트 AND 연산한 결과를 x에 대입
|	bitwise OR	x = a | b	a와 b의 비트 OR 연산한 결과를 x에 대입.
^	bitwise XOR	x = a ^ b	a와 b의 비트 XOR 연산한 결과를 x에 대입.
~	bitwise NOT	x = ~a	a를 비트 별로 반전한 결과를 x에 대입.

비트 처리 연산자

- ex3_4.cpp (비트 처리 논리 연산자)

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

void main()
{
        unsigned short a = 0x1234, b = 0x5678, m = 0xff00, c ,d, e, f;

        c = a & m;
        d = b & ~m;
        e = c | d;
        f = m ^ ~m;

        printf("c = %04x₩n", c);
        printf("d = %04x₩n", d);
        printf("e = %04x₩n", e);
        printf("f = %04x₩n", f);
}
```

```
c = 1200
d = 0078
e = 1278
f = ffff
계속하려면 아무 키나 누르십시오 . . .
```

비트 처리 연산자

- shift 연산자
  - shift 연산자는 비트를 좌우로 이동 시키는데 사용되며 오른쪽 shift(>>)와 왼쪽 shift(<<)가 있다.
  - 이동 연산자의 왼쪽에는 처리할 데이터를 기술하고 오른쪽에는 몇 비트 shift 하는지를 기술한다.
  - 논리적 shift와 산술적 shift가 있다.

연산자	의미	사용 예	설명
>>	shift right	a >> 2	a의 값을 2비트 우측으로 이동
<<	shift left	a << 4	a의 값을 4비트 좌측으로 이동

비트 처리 연산자

- 논리적 shift
  - 데이터가 unsigned로 선언되어 있을 때
  - 각 비트를 좌측 혹은 우측으로 단순히 이동
  - shift가 이루어져서 비어 있는 비트에는 0으로 채워진다.

- 산술적 shift
  - 산술적 shift는 데이터가 부호 있는 정수일 때
  - 우측 이동(>>)
    - 우측으로 1비트 이동하면 원래 값을 2로 나눈 값이 된다.
    - 가장 오른쪽 비트가 1이면 없어져 버리고 오차가 발생한다.
    - 가장 왼쪽 비트는 부호비트로 채워 넣어진다. 즉 음수이면 1, 양수이면 0으로 채워진다.
  - 죄측 이동(<<)
    - 좌측으로 1비트 이동하면 원래 값의 2배로 된다.
    - 가장 왼쪽 비트가 바뀌면(부호가 바뀌는 경우) 오버플로우나 언더플로우가 발생한 경우로 결과 값은 2배 곱한 값이 아니고 틀린 값이 되므로 주의해야 한다.
    - 가장 오른쪽 비트는 0으로 채워 넣어진다.

비트 처리 연산자

- ex3_5.cpp (논리적 shift)

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

void main()
{
        unsigned short a = 0x8006, b = 0x0007; // 논리적 이동
        unsigned short c, d, e, f;

        c = a >> 1;
        d = b >> 1;
        e = a << 1;
        f = b << 1;

        printf("a = %08x, %u₩n", a, a);
        printf("b = %08x, %u₩n", b, b);
        printf("a >> 1 --> %08x, %u₩n", c, c);      // 우측으로 이동
        printf("b >> 1 --> %08x, %u₩n", d, d);      // 우측으로 이동

        printf("a << 1 --> %08x₩n", e);             // 좌측으로 이동
        printf("b << 1 --> %08x₩n", f);             // 좌측으로 이동
}
```

```
a = 00008006, 32774
b = 00000007, 7
a >> 1 --> 00004003, 16387
b >> 1 --> 00000003, 3
a << 1 --> 0000000c
b << 1 --> 0000000e
계속하려면 아무 키나 누르십시오 . . .
```

비트 처리 연산자

- ex3_6.cpp (산술적 shift)

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

void main()
{
        short int a = 0x8006, b = 0x0007; // 산술적 이동
        short int c, d, e, f;

        c = a >> 1;
        d = b >> 1;
        e = a << 1;
        f = b << 1;

        printf("a = %08x, %d₩n", a, a);
        printf("b = %08x, %d₩n", b, b);
        printf("a >> 1 --> %08x, %d₩n", c, c);       // 나누기 2
        printf("b >> 1 --> %08x, %d₩n", d, d);       // 나누기 2
        printf("a << 1 --> %08x, %d₩n", e, e);       // 곱하기 2
                        // 부호가 변경 --> 언더플로우 발생, 계산이 틀림
        printf("b << 1 --> %08x, %d₩n", f, f);       // 곱하기 2
}
```

```
a = ffff8006, -32762
b = 00000007, 7
a >> 1 --> ffffc003, -16381
b >> 1 --> 00000003, 3
a << 1 --> 0000000c, 12
b << 1 --> 0000000e, 14
계속하려면 아무 키나 누르십시오 . . .
```

형 변환 연산자

- 형 변환
  - 묵시적 형 변환 (implicit conversion)
    - 변수에 다른 자료형의 값을 대입하거나 함수로부터 전달받는 경우
    - 서로 다른 자료형의 값을 가지는 수식을 연산하는 경우
    - 함수의 인자로 전달하는 경우
  - 명시적 형 변환 (explicit conversion)
    - 강제로 형 변환 하는 경우

형 변환 연산자

- 묵시적 형 변환 (implicit conversion) (1)
  - 연산 중에 발생하는 형 변환은 피연산자의 자료형에만 영향을 받는다.
    - 피연산자의 자료형이 서로 다르면 자료형의 크기가 큰 쪽으로 형 변환해서 연산이 실행된다.
    - 일반적으로 long double형 > double형 > float형 > unsigned long형 > long형 > unsigned int형 > int형의 순으로 형 변환이 된다.
    - (예) 5/2 : 피연산자가 모두 int 형이므로 결과는 int 형이 되어 소수점 이하가 잘려서 2.5가 아닌 2가 된다.
    - (예) 5.0/2 혹은 5/2.0 : 피연산자의 한 쪽이 float형이므로 다른 쪽의 int형은 float형으로 형 변환이 되어 5.0/2.0으로 계산을 하여 결과는 float 형으로 2.5가 된다.

형 변환 연산자

- 묵시적 형 변환 (implicit conversion) (2)
  - 어떤 자료형의 값을 다른 자료형의 변수에 대입할 수 있다.
    - 대입할 값은 대입되는 변수의 자료형으로 변환된다.
    - 범위가 작은 자료형의 값을 큰 자료형의 변수에 대입하는 것은 문제 없다.
    - 부동소수점수를 정수형에 대입할 때는 소수점 아래의 값은 버린다.
    - 부동소수점수의 값이 너무 커서 정수형에 들어갈 수 없을 때는 전혀 다른 값으로 바뀔 수도 있다.

형 변환 연산자

- ex3_7.cpp (묵시적 형 변환)

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

void main()
{
        float f1, f2, f3;
        int n1;

        f1 = 5 / 2;
        f2 = 5.0f / 2;
        f3 = 5 / 2.0f;
        n1 = 5.0f / 2.0f;

        cout << "5 / 2 = " << f1 << endl;
        cout << "5.0 / 2 = " << f2 << endl;
        cout << "5 / 2.0 = " << f2 << endl;
        cout << "5.0 / 2.0 = " << n1 << endl;
}
```

```
5 / 2 = 2
5.0 / 2 = 2.5
5 / 2.0 = 2.5
5.0 / 2.0 = 2
계속하려면 아무 키나 누르십시오 . . .
```

형 변환 연산자

- 명시적 형 변환 (explicit conversion)
  - C 스타일의 형 변환

```
(변환할_자료형)식;                    // C, C++에서 가능한 형 변환
변환할_자료형(식);                    // C++에서 가능한 형 변환
```

  - C++ 스타일의 형 변환
    - static_cast< >( )
    - const_cast< >( )
    - reinterpret_cast< >( )
    - dynamic_cast< >( )

조건 연산자

- 조건 연산자
  - 조건식이 참(true)으로 평가되면 식1을 수행하고 거짓(false)으로 평가되면 식2를 수행한다.

```
조건식 ? 식1 : 식2;
```

- ex3_8.cpp (조건 연산자)

```
#include <iostream>
using namespace std;
void main()
{
        int a, b, max;

        cout << "a와 b 값을 입력하시오: ";
        cin >> a >> b;

        a >= b ? max = a : max = b;
        cout << "큰 수는 " << max << "입니다." << endl;
}
```

```
a와 b 값을 입력하시오: 9 5
큰 수는 9입니다.
계속하려면 아무 키나 누르십시오 . . .
```

콤마 연산자

- 콤마 연산자
  - 나열할 때 사용한다.
  - 실행 순서는 좌측에서 우측으로 가면서 실행한다.

```
식1, 식2, 식3 ... ;
```

- ex3_9.cpp (콤마 연산자)

```
#include <iostream>
using namespace std;
void main()
{
        int c;                  // 콤마 연산자 사용
        float t, f;

        cout << "섭씨 온도를 입력하시오: ";
        cin >> c;

        f = (t = c * 9.0f/5.0f, t += 32);          // 콤마 연산자 사용
        cout << "화씨 온도는 " << f << "도입니다." <<endl;
}
```

```
섭씨 온도를 입력하시오: 20
화씨 온도는 68도 입니다.
계속하려면 아무 키나 누르십시오 . . .
```

주소 연산자, 간접 연산자

- 주소(address)
  - 컴퓨터의 메모리에는 바이트(byte) 단위로 주소(address)가 지정되어 있다.
  - 변수를 선언하면 일정한 메모리 공간(memory space)을 차지하게 되고, 변수가 저장된 기억 공간의 시작주소가 정해진다.
- 포인터
  - 주소를 저장하기 위해서는 포인터(pointer)를 사용해야 한다. 즉, 포인터는 메모리의 주소를 저장하는 변수이다.
  - 따라서 포인터에 변수의 시작 주소를 넣어 포인터로서 변수의 값 혹은 컴퓨터 메모리를 조작할 수 있다.
  - 포인터를 선언하기 위해서는 변수 이름 앞에 반드시 포인터 연산자 * 를 붙여야 한다.
  - 포인터 선언

```
자료형 *포인터_변수;
```

주소 연산자, 간접 연산자

- 주소 연산자
  - 변수 이름 앞에 &를 붙이면 변수의 주소가 된다.
  - 이 때 &는 주소 연산자(address operator)라고 한다.
  - 포인터의 자료형과 포인터가 가리키는 변수의 자료형이 일치해야 한다.

- 간접 연산자
  - 포인터가 가리키는 기억 공간의 값을 알려면 포인터 이름 앞에 *를 붙여주면 된다.
  - 이 때 *는 간접 연산자(dereference operator)라고 한다.

주소 연산자, 간접 연산자

- ex3_10.cpp (주소 연산자와 간접 연산자)

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

void main()
{
        int inum = 100;
        int* iptr;
        double fnum = 12.5;
        double *fptr;

        iptr = &inum;
        printf("inum의 주소= %p₩n", &inum);
        printf("iptr이 가리키는 기억장소의 값= %d₩n", *iptr);

        fptr = &fnum;
        cout << "fnum의 주소= " << &fnum << endl;
        cout << "fptr이 가리키는 기억장소의 값= " << *fptr << endl;
}
```

```
inum의 주소 = 0012FF60
iptr이 가리키는 기억장소의 값 = 100
fnum의 주소 = 0012FF44
fptr이 가리키는 기억장소의 값 = 12.5
계속하려면 아무 키나 누르십시오 . . .
```

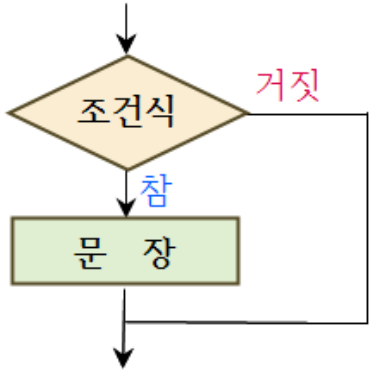
제어 구조

- 순차적 실행
  - main() 함수의 첫 부분에서 시작하여 한 번에 한 문장씩 순서대로 실행한다.
- 선택적 실행
  - 어떤 조건을 평가한 후 조건에 맞으면 실행하고, 맞지 않으면 실행하지 않는다.
  - if 문
  - if ~ else 문
  - switch 문
- 반복적 실행
  - 조건이 만족하는 동안 정해진 부분을 계속 반복 실행한다.
  - for 문
  - wile 문
  - do ~ while 문

선택 제어문

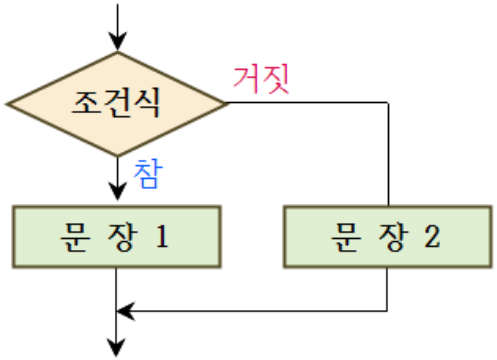
- if 문
  - 조건식을 평가하여 결과가 참(true)이면 문장을 실행하고, 거짓(false)이면 문장을 실행하지 않는다.

```
if (조건식 )
    문장;
```



- if ~ else 문
  - 조건식을 평가하여 참이면 if 다음의 문장(문장1)을 실행하고, 거짓이면 else 다음의 문장(문장2)을 실행한다.

```
if (조건식)
  문장1;
else
  문장2;
```



선택 제어문

- ex3_11.cpp (if ~ else 문)

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

void main()
{
        int score;

        cout << "점수를 입력하세요(0 ~ 100): " ;
        cin >> score;

        if (score >= 60)
                cout << "합격입니다." << endl;
        else
                cout << "불합격입니다." << endl;
}
```

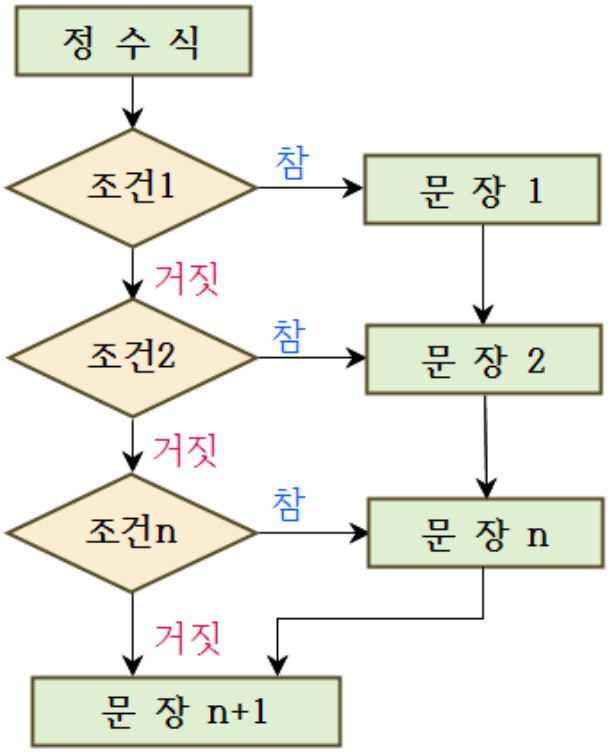
```
점수를 입력하세요(0 ~ 100): 75
합격입니다.
계속하려면 아무 키나 누르십시오 . . .
```

```
점수를 입력하세요(0 ~ 100): 55
불합격입니다.
계속하려면 아무 키나 누르십시오 . . .
```

선택 제어문

- switch 문
  - 여러 가지 조건에 따라 프로그램의 흐름을 분기(branch)시킬 수 있다.

```
switch (정수식)
{
   case 조건1 :
        문장1;
   case 조건2 :
        문장2;
   ....
   case 조건n :
        문장n;
   default :
        문장n+1
}
```



선택 제어문

- ex3_12.cpp (1) (switch 문)

```
#include <iostream>
using namespace std;
void main()
{
        int score;
        char grade;

        cout << "점수를 입력하세요(0 ~ 100): " ;
        cin >> score;
```

```
점수를 입력하세요(0 ~ 100): 85
B 학점입니다.
계속하려면 아무 키나 누르십시오 . . .
```

선택 제어문

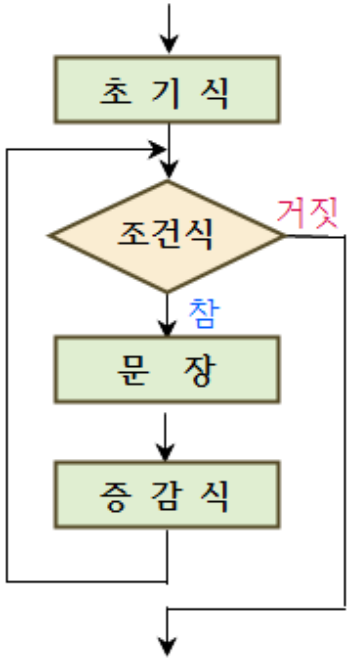
- ex3_12.cpp (2) (switch 문)

```
    switch (score/10) {
    case 10:
    case 9 :
            grade = 'A';
            break ;
    case 8 :
            grade = 'B';
            break ;
    case 7 :
            grade = 'C';
            break ;
    case 6 :
            grade = 'D';
            break ;
    default :
            grade = 'F';
    }
    cout << grade << " 학점입니다." << endl;
}
```

반복 제어문

- for 문
  - 정해진 횟수만큼 어떤 문장들을 반복하여 실행할 수 있는 반복 구조이다.
  - 반복 횟수를 미리 알고 있는 경우 유용하다.

```
for (초기식; 조건식; 증감식)
        문장;
```



반복 제어문

- ex3_13.cpp (for 문)

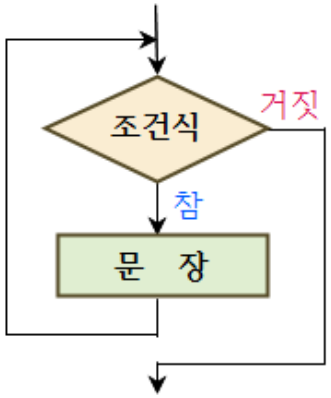
```
#include <iostream>
using namespace std;

void main()
{
        int i, sum;

        for (i=1, sum=0; i<=100; i++)
                sum += i;
        cout << "1부터 100까지 정수의 합은 " << sum << endl;
}
```

```
1부터 100까지 정수의 합은 5050
계속하려면 아무 키나 누르십시오 . . .
```

반복 제어문

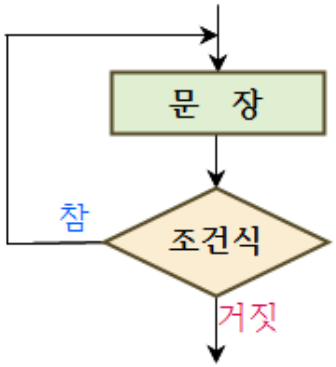
- while 문
  - while 문은 조건을 만족하는 동안은 문장들을 반복하여 실행한다.
  - 반복 횟수를 미리 알 수 없고 조건에 따라 반복하는 경우 사용한다.

```
while (조건식)
        문장;
```



- do ~ while 문
  - 문장이 적어도 한 번은 실행되어야 하는 경우에 사용하면 편리하다.

```
do {
        문장;
} while(조건식);
```



반복 제어문

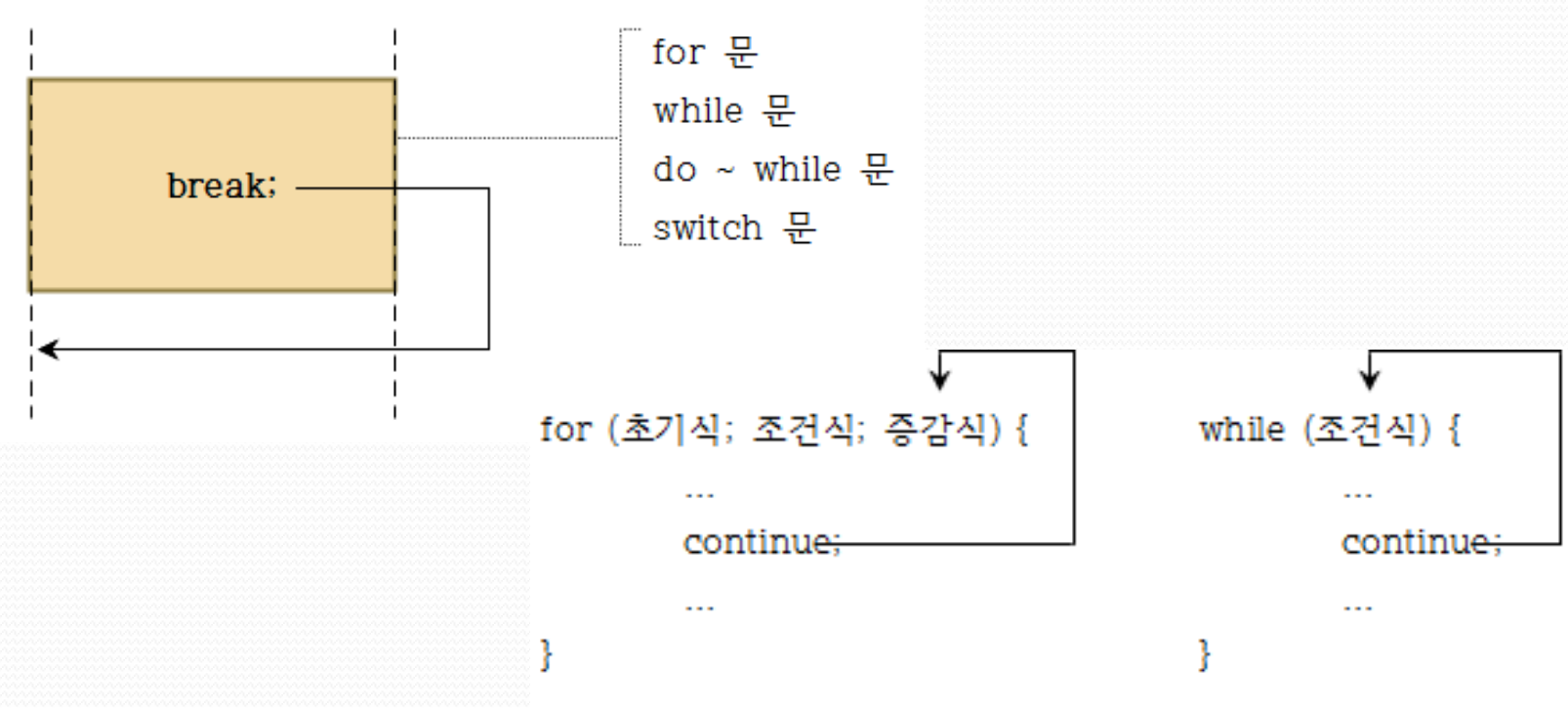
- ex3_14.cpp (while 문)

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

void main()
{
	int i, sum;

	i = 0, sum = 0;
	while (i<100) {
		i++;
		sum += i;
	}
	cout << "1부터 100까지 정수의 합은 " << sum << endl;
}
```

```
1부터 100까지 정수의 합은 5050
계속하려면 아무 키나 누르십시오 . . .
```

반복 제어문

- break 문
  - 반복 루프(for, while, do ~ while)를 벗어나기 위하여 사용된다.
  - switch 문을 벗어나기 위해서도 사용된다.
- continue 문
  - 현재 실행하고 있는 반복(for, while, do ~ while) 과정의 나머지를 건너뛰고 다음 반복 과정을 시작하게 한다.



반복 제어문

- ex3_15.cpp (1)

```
#include <iostream>
using namespace std;
const double PI = 3.14159;
void main()
{
        int num;
        double a, b, area;
        do {
                cout << "=======================================" << endl ;
                cout << " 1. 삼각형의 넓이구하기 " << endl ;
                cout << " 2. 사각형의 넓이구하기 " << endl ;
                cout << " 3. 원의 넓이구하기 " << endl ;
                cout << " 9. 끝내기 " << endl ;
                cout << "=======================================" << endl ;
                cout << "원하는 도형을 선택하세요: " ;
                cin >> num;
```

반복 제어문

- ex3_15.cpp (2)

```
switch (num) {
case 1:
case 2:
        cout << "밑면의 길이를 입력하세요: " ;
        cin >> a;
        cout << "높이를 입력하세요: " ;
        cin >> b ;
        break ;
case 3:
        cout << "반지름을 입력하세요: " ;
        cin >> a;
        break ;
case 9:
        cout << "감사합니다." << endl ;
        return ;
default:
        cout << "입력을 잘못하였습니다! " << endl << endl ;
        continue;
}
```

반복 제어문

- ex3_15.cpp (3)

```
            switch (num) {
            case 1:
                area = a * b / 2;
                break ;
            case 2:
                area = a * b;
                break ;
            case 3:
                area = PI * a * a;
            }
            cout << "area = " << area << endl << endl;
        } while (num != 9);
    }
```

반복 제어문

- ex3_15.cpp (4)

```
==================================
     1. 삼각형의 넓이 구하기
     2. 사각형의 넓이 구하기
     3. 원의 넓이 구하기
     9. 끝내기
==================================
원하는 도형을 선택하세요: 3
반지름을 입력하세요: 10
area = 314.159

==================================
     1. 삼각형의 넓이 구하기
     2. 사각형의 넓이 구하기
     3. 원의 넓이 구하기
     9. 끝내기
==================================
원하는 도형을 선택하세요: 5
입력을 잘못하였습니다!

==================================
     1. 삼각형의 넓이 구하기
     2. 사각형의 넓이 구하기
     3. 원의 넓이 구하기
     9. 끝내기
==================================
원하는 도형을 선택하세요: 9
감사합니다.
계속하려면 아무 키나 누르십시오 . . .
```

Question & Answer

Any Question?

Please.