C++ Features and Data Types

프로그래밍 입문(2)

Topics

- C++ 프로그램의 기초적인 문법 및 형태
    - main()함수, 전처리 지시자, iostream, 문장(statement), 코드 블럭(code block), 주석문(comment) 등
- C++의 기본 자료형 (Primitive Data Types)
    - 정수(integer), 실수(floating point), 문자(character), 불리언(boolean), void
- **C++의 다양한 연산자 (Operators)**
    - **대입(assignment), 산술(arithmetic), 관계/비교(relation/comparison), 논리(logic), 비트(bitwise), 복합 대입(compound assignment) 연산자 등.**

Operators

Operators

- 프로그램의 가장 중요한 부분 중 하나는 데이터를 가지고 연산을 하는 것.

- 이런 연산을 표시하기 위해 C++에서는 굉장히 다양한 종류의 연산자를 제공하고 있습니다.

- 대입(assignment), 산술(arithmetic), 관계/비교(relation/comparison), 논리(logic), 비트(bitwise), 복합 대입(compound assignment) 연산자 등.

Arithmetic Operators

- 산술 연산자는 우리가 일반적으로 생각하는 사칙연산을 주로 표현합니다.

- 덧셈: a + b, 뺄셈: a - b, 곱셈: a * b, 나눗셈: a / b

- 나머지(modulo): a % b

- 그 외에도 단항 연산자 (+, -)와 증가/감소(increment/decrement)도 수학적 연산을 표시하는 연산자입니다.

Mixed Data Types

- 사칙연산에서 데이터형을 섞어서 계산하게 되는 경우 다음의 두 가지 부분을 주의해야 합니다.
  - 연산에 사용하는 데이터형
  - 연산의 결과가 저장되는 데이터형
- 실수를 이용하여 연산 → 정수 데이터형 변수에 값을 저장
  - 소수점 이하가 누락됨.
  - `int c = 2.4 / 2.0;` → `c = 1, c != 1.2`

Mixed Data Types

- 정수를 이용하여 연산 → 실수 데이터형 변수에 값을 저장
  - 일반적인 수학에서는 소수점 이하가 생길 수 있지만, C++에서는 모두 버림.
    - `double d = 3 / 2;` → `d = 1.0, d != 1.5`
- 더 큰 데이터형의 결과를 더 작은 데이터형 변수에 저장
  - 숫자가 작은 데이터형이 표현하는 범위를 넘어서면 다른 값으로 인식됨.
    - `int y = 40000; short x = y + y;` → `x = 14464, x != 80000`

Mixed Data Types

- 정밀도가 높은 데이터형의 연산 결과를 정밀도가 낮은 데이터형 변수에 저장
  - 원래의 정밀도가 보장되지 않음.

- `double i = 1.0000002;`
- `float j = i * 2.0;` → `j = 2.0000005`
- `double k = i * 2.0;` → `k = 2.0000004`

How to Avoid Confusions?

- 가능하면 연산에 사용하는 데이터형과 연산 결과를 저장하는 데이터형을 일치시킨다.
- double만을 사용한다?
  - 부동소수점 연산은 정수 연산에 비해 속도가 매우 느림.
  - 대량의 연산이 필요한 경우 특화된 GPU 등을 사용하기도 함.
  - 단순한 정수 계산을 위해 사용하면 낭비가 너무 심하게 됨.

Type Casting

- 데이터형 변환(type casting)은 한 데이터형을 다른 데이터형으로 변환하는 방법.

- 괄호 안에 변경하고 싶은 데이터형을 쓰는 식으로 변경합니다.
  - `int a = 5; double x = (double) a / 2;`

- 위의 경우, 변수 a의 데이터형이 영원히 변경되는 것이 아니고, 일시적으로 계산에 사용될 때만 double로 취급됩니다.

- C++에서는 `double (a)`와 같이 함수형으로도 형변환이 가능합니다.

Increment / Decrement

- 변수의 값을 ++와 --로 1만큼 증가하거나 감소함.
  - 정수, 실수 모두 다음과 같이 바꿔주는 것과 동일함.
  - `int a = 3; ++a; → a = a + 1;`
  - `double b = 3.14; ++b; → b = b + 1;`
- ++, --연산자가 변수 앞에 오면 변수를 사용하기 전 먼저 증가/감소를 시키고, 뒤에 오면 사용하고 나서 값을 증가/감소시킴.
  - `int a = 3; cout << ++a; → 4`
  - `int a = 3; cout << a++; → 3`

Increment / Decrement

- `int c = 1;`
- `int d = --c + c++ + c-- + ++c;`
- `cout << “d = “ << d << endl;`
- output)
  - `d = 2`
- `int d = --c + c++ + c-- + ++c;`
  0 0 1 1

Bitwise Operators

- 비트 단위에서 작동하는 연산자들로, 총 6가지가 있음.
  - NOT ~ : 모든 비트를 뒤집는 연산.
  - AND & : bitwise AND 연산.
  - OR | : 엔터키 위의 역슬래시(\)에서 쉬프트를 눌러입력하는 것.
  - XOR ^ : 이 연산자는 다른 언어에서 제곱을 나타내는 경우가 있으므로 주의.
  - Left Shift << : 각 비트를 왼쪽으로 한 칸씩 이동.
  - Right Shift >> : 각 비트를 오른쪽으로 한 칸씩 이동.

a = 48

128	64	32	16	8	4	2	1
0	0	1	1	0	0	0	0

b = 37

128	64	32	16	8	4	2	1
0	0	1	0	0	1	0	1

Bitwise NOT : ~a = 207 = 255 - 48

128	64	32	16	8	4	2	1
1	1	0	0	1	1	1	1

Bitwise AND : a & b

128	64	32	16	8	4	2	1
0	0	1	1	0	0	0	0

&

128	64	32	16	8	4	2	1
0	0	1	0	0	1	0	1

=

128	64	32	16	8	4	2	1
0	0	1	0	0	0	0	0

Bitwise OR : a | b

128	64	32	16	8	4	2	1
0	0	1	1	0	0	0	0

|

128	64	32	16	8	4	2	1
0	0	1	0	0	1	0	1

=

128	64	32	16	8	4	2	1
0	0	1	1	0	1	0	1

a = 48

128	64	32	16	8	4	2	1
0	0	1	1	0	0	0	0

b = 37

128	64	32	16	8	4	2	1
0	0	1	0	0	1	0	1

Bitwise XOR : a ^ b

128	64	32	16	8	4	2	1
0	0	1	1	0	0	0	0

^

128	64	32	16	8	4	2	1
0	0	1	0	0	1	0	1

=

128	64	32	16	8	4	2	1
0	0	0	1	0	1	0	1

Bitwise Left Shift : a << 1 = 96

128	64	32	16	8	4	2	1
0	0	1	1	0	0	0	0

<< 1

128	64	32	16	8	4	2	1
0	1	1	0	0	0	0	0

Bitwise Right Shift : a >> 2 = 12

128	64	32	16	8	4	2	1
0	0	1	1	0	0	0	0

>> 2

128	64	32	16	8	4	2	1
0	0	0	0	1	1	0	0

Bitwise Operators

- signed와 unsigned형에서 동작이 다를 수 있음.
- 정수형을 비트로 표현하는 방법의 문제 (2의 보수).
- 실수형 값에 대해서는 유효하지 않은 연산자
  - e.g.) double x = 1.0; double y = 2.0;
  - double z = x & y; → invalid!
- 비트단위에서의 연산은 빠르게 처리가 가능함.

Relation / Comparison Operators

- 두 표현식을 비교하기 위한 연산자로 수학에서 부등식에 사용되는 기호를 떠올리면 됩니다.

- ==, !=, >, <, <=, >= 등이 있고, 연산자 좌우로 표현식이 오는 이항 연산자입니다.

- 같다(==)와 다르다(!=)의 부호에 주의.

- 마찬가지로 크거나 같다, 작거나 같다도 부등호 뒤에 등호가 온다는 점에 주의.

- 관계/비교 연산자는 표현식의 참/거짓에 따라 boolean 값을 나타냄.

Floating Point Number Comparison

- 부동소수점 수의 저장 방식에 따라 정밀도 문제로 오차가 발생할 수 있음.

- `double x = 0.3; double y = 0.1 + 0.1 + 0.1;`

- `x == y` ? → 참이라는 보장이 없음.

- `x = 0.29999999999999998`

- `y = 0.30000000000000004`

Floating Point Number Comparison

- 부동소수점 방식에서 정밀도 범위 밖의 값의 차이로 비교가 어려움.

- 대신 `EPSILON = 0.000001; x - y < EPSILON`과 같이 두 수의 차이가 오차보다 적은지 비교하는 방식을 사용.

- `float`형의 경우 정밀도가 6자리 → 0.000001보다 작은 차이가 있는 경우는 같은 수로 볼 수 있음.

  - e.g.) 0.000001**00001**, 0.000001**00201** → 6자리까지만 표시된다면 두 수는 사실상 같은 수.

  - 만약 두 수를 다른 수로 구분하고 싶다면 더 정밀도가 높은 `double`형을 사용해야함.

Compound Assignment

- 대입과 다른 연산의 결합 형태.
- +=, -=, *=, /=, %= 처럼 다른 연산자 + 등호(=)의 형태로 연산의 결과를 대입하는 것을 나타냄.

```
a += 1; //a = a + 1;
b *= 2; //b = b * 2;
c /= d - 1; //c = c / (d - 1);
```

Operator Precedence

- 괄호가 있으면 괄호 먼저(안쪽부터 바깥쪽으로).
- 단항(unary)은 이항(binary)보다 먼저.
- 결합방향(grouping)은 대부분 왼쪽에서 오른쪽으로.
- 사칙연산을 비교/관계보다 먼저.
- http://www.cplusplus.com/doc/tutorial/operators/

Operator Precedence

- `a = c * (x / (b + (y - 4)))`

- ***v1*** `= y - 4`
- ***v2*** `= b +` ***v1***
- ***v3*** `= x /` ***v2***
- ***v4*** `= c *` ***v3***

- `b + x >= y % z / --c`

- ***v1*** `= b + x`
- ***v2*** `= y % z`
- ***v3*** `= --c`
- ***v4*** `=` ***v2*** `/` ***v3***
- ***v5*** `=` ***v1*** `>=` ***v4***

Operator Precedence

- `x = a / b * c;`
  - `(a/b)*c`와 `a/(b*c)`의 결과가 달라짐
  - 이 경우는 결합방향에 따라 왼쪽에서 오른쪽으로 계산.
- `x = a * b + c * d;`
  - `a * b`와 `c * d`가 덧셈보다 먼저 계산되어야 함.
  - 그렇다면 둘 중 어떤 것을 먼저 계산해야 하는가?
  - 동일한 우선순위의 연산자들에서 명확히 먼저 계산해야 할 이유가 없는 경우, 컴파일러의 선택에 맡겨짐.

Summary

- C++다양한 연산자들
- 여러 데이터형과 연산자의 사용
- 연산자의 우선 순위