Section Overview

Expressions, Statements and Operators

- Expressions
- Statements and block statements
- Operators
  - Assignment
  - Arithmetic
  - Increment and decrement
  - Equality
  - Relational
  - Logical
  - Compound assignment
  - Precedence

Expressions and Statements

Expressions

An expression is:

- The most basic building block of a program
- “a sequence of operators and operands that specifies a computation”
- Computes a value from a number of operands
- There is much, much more to expressions – not necessary at this level

Expressions and Statements

Expressions - examples

```
34                    // literal

favorite_number             // variable

1.5 + 2.8             // addition

2 * 5                       // multiplication

a > b                       // relational

a = b                       // assignment
```

Expressions and Statements

Statements

A statement is:

- A complete line of code that performs some action
- Usually terminated with a semi-colon
- Usually contain expressions
- C++ has many types of statements
  - expression, null, compound, selection, iteration,
  - declaration, jump, try blocks, and others.

Expressions and Statements

Statements - examples

```
int x;                          // declaration

favorite_number = 12;           // assignment

1.5 + 2.8;                            // expression

x = 2 * 5;                              // assignment

if (a > b) cout << "a is greater than b";     // if

;
```

Using Operators

- C++ has a rich set of operators
  - unary, binary, ternary
- Common operators can be grouped as follows:
  - assignment
  - arithmetic
  - increment/decrement
  - relational
  - logical
  - member access
  - other

Assignment Operator (=)

```
lhs = rhs
```

- `rhs` is an expression that is evaluated to a value
- The value of the `rhs` is stored to the `lhs`
- The value of the `rhs` must be type compatible with the `lhs`
- The lhs must be assignable

- Assignment expression is evaluated to what was just assigned
- More than one variable can be assigned in a single statement

Mixed Type Expressions

•C++ operations occur on same type operands

•If operands are of different types, C++ will convert one

•Important! since it could affect calculation results
•C++ will attempt to automatically convert types (coercion).

If it can’t, a compiler error will occur

Mixed Type Expressions

Conversions

- Higher vs. Lower types are based on the size of the values the type can hold
  - `long double,double,float,unsigned long,long,unsigned int,int`
  - `short` and `char` types are always converted to `int`
- Type Coercion: conversion of one operand to another data type
- Promotion: conversion to a higher type
  - Used in mathematical expressions
- Demotion: conversion to a lower type
  - Used with assignment to lower type

Mixed Type Expressions

Examples

- `lower` **op** `higher` **the** `lower` **is promoted to a** `higher`

```
2 * 5.2
```

2 is promoted to 2.0

- `lower = higher;` **the** `higher` **is demoted to a** `lower`

```
int num {0};
num  = 100.2;
```

Mixed Type Expressions

Explicit Type Casting – static_cast<type>

```
int total_amount {100};
int total_number {8};
double average {0.0};

average = total_amount / total_number;
cout << average << endl;                    // displays 12

average = static_cast<double>(total_amount) / total_number;
cout << average << endl;                    // displays 12.5
```

Testing for Equality

The == and != operators

- Compares the values of 2 expressions
- Evaluates to a Boolean (True or False, 1 or 0)
- Commonly used in control flow statements

```
expr1 == expr2

expr1 != expr2

100 == 200
num1 != num2
```

Testing for Equality

The == and != operators

```
bool result {false};

result = (100 == 50+50);

result = (num1 != num2);

cout << (num1 == num2) << endl;     // 0 or 1
cout << std::boolalpha;
cout << (num1 == num2) << endl;     // true or false
cout << std::noboolalpha;
```

Relational Operators

`expr1` **op** `expr2`

Operator	Meaning
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
<=>	three-way comparison (C++20)

Logical Operators

Operator	Meaning
`not` `!`	negation
`and` `&&`	logical and
`or` `||`	logical or

Logical Operators

not (!)

expression a	not a !a
true	false
false	true

Logical Operators

and (&&)

expression a	expression b	a and b a && b
true	true	**true**
true	false	false
false	true	false
false	false	false

Logical Operators

or (||)

expression a	expression b	a or b a || b
true	true	**true**
true	false	**true**
false	true	**true**
false	false	false

Logical Operators

Precedence

- `not` has higher precedence than `and`
- `and` has higher precedence than `or`

- `not` is a unary operator
- `and` and `or` are binary operators

Logical Operators

Examples

```
num1 >= 10 && num1 < 20
num1 <= 10 || num1 >=20

!is_raining && temperature > 32.0

is_raining || is_snowing

temperature > 100 && is_humid || is_raining
```

Logical Operators

Short-Circuit Evaluation

- When evaluating a logical expression C++ stops as soon as the result is known

```
expr1 && expr2 && expr3

expr1 || expr2 || expr3
```

Compound Assignment

op=

Operator	Example	Meaning
+=	`lhs += rhs;`	`lhs = lhs + (rhs);`
-=	`lhs -= rhs;`	`lhs = lhs - (rhs);`
*=	`lhs *= rhs;`	`lhs = lhs * (rhs);`
/=	`lhs /= rhs;`	`lhs = lhs / (rhs);`
%=	`lhs %= rhs;`	`lhs = lhs % (rhs);`
>>=	`lhs >>= rhs;`	`lhs = lhs >> (rhs);`
<<=	`lhs <<= rhs;`	`lhs = lhs << (rhs);`
&=	`lhs &= rhs;`	`lhs = lhs & (rhs);`
^=	`lhs ^= rhs;`	`lhs = lhs ^ (rhs);`
|=	`lhs |= rhs;`	`lhs = lhs | (rhs);`

Logical Operators

Examples

```
lhs op= rhs; // lhs = lhs op (rhs);

a += 1;          // a = a + 1;
a /= 5;          // a = a / 5;
a *= b + c;      // a = a * (b + c);

cost += items * tax;
                 //cost = cost + (items * tax);
```

Operator Precedence (not a complete list)

Higher to lower

Operator	Associativity
`[]  ->  .  ()`	left to right
`++  --  not  -(unary)  *(de-ref)  &  sizeof`	right to left
`*  /  %`	left to right
`+  -`	left to right
`<<  >>`	left to right
`<  <=  >  >=`	left to right
`==  !=`	left to right
`&`	left to right
`^`	left to right
`|`	left to right
`&&`	left to right
`||`	left to right
`=  op=  ?:`	right to left

Operator Precedence

What is associativity?

- Use precedence rules when adjacent operators are different

```
expr1 op1 expr2 op2 expr3 // precedence
```

- Use associativity rules when adjacent operators have the same precedence

```
expr1 op1 expr2 op1 expr3 // associativity
```

- Use parenthesis to absolutely remove any doubt

Operator Precedence

Example

```
result = num1 + num2 * num3;
result = ( num1 + (num2 * num3) );

result1 = num1 + num2 - num3;
result1 = ( (num1 + num2) - num3 );
```