Lecture 3

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Expressions – C++ Primer Chapter 4

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

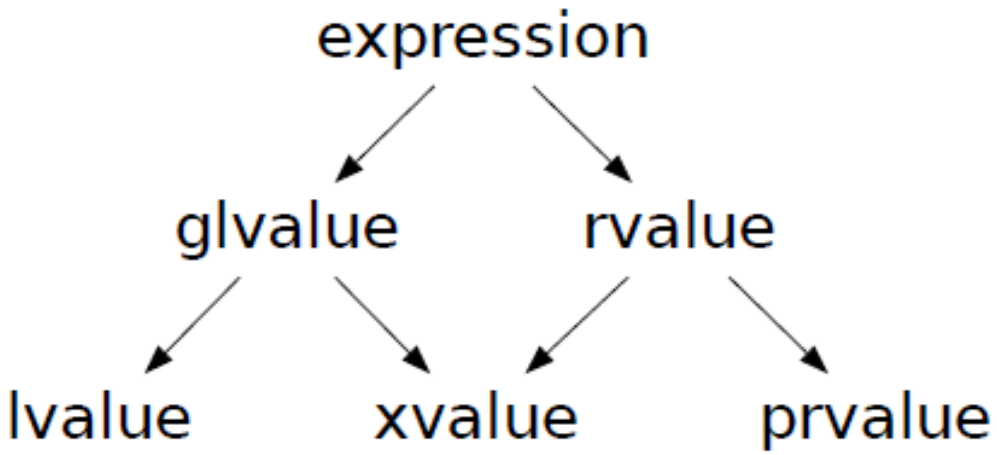
Moving objects (C++11)

- One of the major features in the new standard is the ability to move rather than copy an object!
- Moving instead of copying can provide a significant performance boost
- To support move operations a new kind of reference is introduced, an rvalue reference
- Example
  - int i = 42;
  - int &&rr = i * 42;

Rvalue reference (C++11)

- Rvalue references refer to objects that are about to be destroyed
- While lvalues have persistent state, rvalues are either literals or temporary objects created in the course of evaluating expressions
- Note that variables are lvalues!

Expression category taxonomy (std 3.10)



Expression category I

lvalue: An expression that yields an object or function.

A nonconst lvalue that denotes an object may be the left-hand operand of assignment.

xvalue: An xvalue (an "eXpiring" value) refers to an object, usually near the end of its lifetime (so that its resources may be moved).

Certain kinds of expressions involving rvalue references yield xvalues.

prvalue: A "pure" rvalue that is not an xvalue

glvalue: A "generalized" lvalue

rvalue: An expression that yields a value but not the associated location, if any, of that value, an xvalue or a temporary object.

Expression category II


```
int i;
i = 1;
```
lvalue

```
int* ip;
*ip = 1;
```
lvalue

```
int i;
i = 1;
```
prvalue

```
float& f() {
 float f;
 return f;
}
```
lvalue

```
float f() {
 return 2;
}
```
prvalue

```
float&& f() {
 return 2;
}
```
xvalue

Expressions

Expression: Lowest level of computation in C++ program.

- Expressions generally apply an operator to one or more operands.
- Each expression yields a result.
- Expressions can be used as operands, so we can write compound expressions requiring the evaluation of multiple operators.
- Example: i+j

Operands: Values on which an expression operates

Result: Value or object obtained by evaluating an expression.

Operators

Operator: Symbol that determines what action an expression performs.

- C++ defines a set of operators and specifies how many operands each operator takes.
- C++ also defines the precedence and associativity of each operator
- Operators may be overloaded and applied to values of class type.

Unary / binary operators: Operators that take a single / two operand.

Meaning of an operator – what operation is performed and the type of the result – depends on the types of ist operands

Arithmetic Operators

- List 7 operators
  - + (2), - (2), *, / , %
- Overflow and other arithmetic exceptions

Relational and Logical Operators

- List 9 operators
  - !, >, >=, < , <=, ==, !=, &&, ||
- Evaluate left operand before right
  - Example: (a <b) && (c == d)
- Short-circuit evaluation
  - Right operand is only evaluated if left operand does not determine result

Bitwise Operators I

- List 6 operators
  - ~, <<, >> , &, ^, |
- Use with unsigned types
- Use bitset container
- Shift operators for IO

Bitwise Operators II

^ operator: Bitwise exclusive or operator.

Generates a new integral value in which each bit position is 1 if either but not both operands contain a 1 in that bit position; otherwise, the bit is 0.

| operator: Bitwise OR operator.

Generates a new integral value in which each bit position is 1 if either operand has a 1 in that position; otherwise the bit is 0.

~ operator: Bitwise NOT operator.

Inverts the bits of its operand.

& operator: Bitwise AND operator.

Generates a new integral value in which each bit position is 1 if both operands have a 1 in that position; otherwise the bit is 0.

Bitwise Operators II

<< operator: The left-shift operator.

Shifts bits in the left-hand operand to the left. Shifts as many bits as indicated by the right-hand operand. The right-hand operand must be zero or positive and strictly less than the number of bits in the left-hand operand.

>> operator: The right-shift operator.

Like the left-shift operator except that bits are shifted to the right. The right-hand operand must be zero or positive and strictly less than the number of bits in the left-hand operand.

Increment and Decrement Operators

- **++ operator:** The increment operator.
  - The increment operator has two forms, prefix and postfix.
  - Prefix increment yields an lvalue. It adds one to the operand and returns the changed value of the operand.
  - Postfix increment yields an rvalue. It adds one to the operand and returns the original, unchanged value of the operand.
  - Use postfix only when necessary
- Brevity can be a virtue (*iter++)

Conditional Operators

- **?: operator:** The conditional operator. If-then-else expression of the form:
  - cond ? expr1 : expr2;
  - If the condition cond is true then expr1 is evaluated. Otherwise, expr2 is evaluated.

Sizeof Operator

- Returns value of type size_t
- Forms:
  - sizeof (type name), sizeof (int)
  - sizeof (expr)
  - sizeof expr

Other Operators

Arrow Operator: Synonym for expressions involving dot and dereference operator

Example: MyClass* obj; obj->fct() is equal to (*obj).fct()

, operator: The comma operator.

Expressions separated by a comma are evaluated left to right. Result of a comma expression is the value of the right-most expression.

Evaluating Compound Expressions

compound expression: Expression involving more than one operator.

Precedence: Order in which different operators in a compound expression are grouped.

Operators with higher precedence are grouped more tightly than operators with lower precedence.

Example: 2 + 3 / 4

Associativity: Determines how operators of the same precedence are grouped.

Operators can be either right associative (operators are grouped from right to left) or left associative (operators are grouped from left to right)

Example: i = j = k

http://en.cppreference.com/w/cpp/language/operator_precedence

Assignment Operators

- Left-hand operand must be a nonconst lvalue
- Assignment is right associative
  - Example: i = j = 0
- Assignment has low precedence
  - Example: i = j + 3
- Compound assignment operators
  - Example: +=, -=

Evaluating Compound Expressions

- **order of evaluation**: Order, if any, in which the operands to an operator are evaluated.
  - In most cases in C++ the compiler is free to evaluate operands in any order.
  - Example: if (ia[index++] < ia[index]) (behavior undefined!)
- Rules of thumb
  - When in doubt, parenthesize expressions
  - If you change the value of an operand, do not use it elsewhere in the same statement

Type Conversions

Conversion: Process transforming value of one type into value of another type.

The language defines conversions among the built-in types.

Conversions to and from class types are also possible.

implicit conversion: A conversion that is automatically generated by the compiler.

The compiler will automatically convert the operand to the desired type if an appropriate conversion exists.

Example: int i; double d = i;

arithmetic conversion: A conversion from one arithmetic type to another.

In the context of binary arithmetic operators, arithmetic conversions usually attempt to preserve precision by converting a smaller type to a larger type (e.g., small integral types, such as char and short, are converted to int).

Standard Conversions (std 4)

- A standard conversion sequence is a sequence of standard conversion in the order:
  - 0 or 1 conversion from the following set: lvalue-to-rvalue conversion, array-to-pointer, function-to-pointer
  - 0 or 1 conversion from the following set: integral promotions, floating point promotion, integral, floating point, floating-integral, pointer, pointer to member, and boolean conversions.
  - 0 or 1 qualification conversion (CV-qualifiers const / volatile)

Explicit Conversions

Cast: An explicit conversion.

static_cast: An explicit request for a type conversion that the compiler would do implicitly. Often used to override an implicit conversion that the compiler would otherwise perform.

Example: int i; double d = static_cast<double>(i) / 2;

const_cast: A cast that converts a const object to the corresponding nonconst type.

dynamic_cast: Used in combination with inheritance and run-time type identification.

reinterpret_cast: Interprets the contents of the operand as a different type.

Inherently machine-dependent and dangerous.

Type Conversions

- Try to avoid old-style casts
  - Example: int i; double d = (double)i;

Constant expressions (C++14std 5.20)

- Expression that can be evaluated at compile time
- A literal is a constant expression
- Example:
  - constexpr int i = 10;

Statements – C++ Primer Chapter 5

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Statements

- **expression statement**: An expression followed by a semicolon.
  - An expression statement causes the expression to be evaluated.
- Null statement: ;
- Declaration statement: int i;
- Compound statements (Blocks): { … }
  - A block is not terminated by a semicolon, except e.g. classes!
- Statement scope
  - Variables defined in a condition must be initialized
  - Example: for(int i; ;) {}

If Statement


```
if (val <= a)
  if (val == a)
    ++count;
else
  count = 1;
```

- **dangling else**: how to process nested if statements with more ifs than elses?
  - In C++, an else is always paired with the closest preceding unmatched if
  - Curly braces can be used to hide an inner if

switch Statement

- **switch statement**: Conditional execution statement that starts by evaluating the expression that follows the switch keyword.
  - Control passes to the labeled statement with a case label that matches the value of the expression.
  - Case labels must be constant integral expressions
  - If there is no matching label, execution either branches to the default label, if there is one, or falls out of the switch if there is no default label.
  - Execution continues across case boundaries until the end of the switch statement or a break is encountered
  - Comment if no break at the end!
  - Be careful with variable definitions inside a switch statement

If and switch Statements

break statement: Terminates the nearest enclosing loop or switch statement.

Execution transfers to the first statement following the terminated loop or switch.

case label: Integral constant value that follows the keyword case in a switch statement.

- No two case labels in the same switch statement may have the same value.
- If the value in the switch condition is equal to that in one of the case labels, control transfers to the first statement following the matched label.
- Execution continues from that point until a break is encountered or it flows off the end of the switch statement.

default label: The switch case label that matches any otherwise unmatched value computed in the switch condition.

While and for Statements

- **While Statement**
- **Do While Statement**
  - Always ends with a semicolon
- **For loop statement**
  - Parts of for header are init-statement, condition, expression: for (; ;)
  - Multiple definitions in for header: for (int i = 0, int j = 1; ;)
  - Range based for

Range-based for (C++11)

Range-based for: Control statement that iterates through a collection of values.

Form: *for (declaration : expression) statement*

Other Statements

continue statement: Terminates the current iteration of the nearest enclosing loop.

Execution transfers to the loop condition in a while or do or to the expression in the for header.

goto statement: Do not use!

Exception Handling I

try block: block enclosed by the keyword try and one or more catch clauses.

- If code inside the try block raises an exception and one of the catch clauses matches the type of the exception, then the exception is handled by that catch.
- Otherwise, the exception is handled by an enclosing try block or the program terminates.

catch clause (exception handler) : The catch keyword, an exception specifier in parentheses, and a block of statements.

- The code inside a catch clause does whatever is necessary to handle an exception of the type defined in its exception specifier.

Exception Handling II

throw expression (or raise): Expression that interrupts the current execution path.

- Each throw throws an object and transfers control to the nearest enclosing catch clause that can handle the type of exception that is thrown.

exception specifier: The declaration of an object or a type that indicates the kind of exceptions a catch clause can handle.

Terminate: Library function that is called if an exception is not caught. Usually aborts the program.

C++ Standard library exceptions examples

exception	description
bad_alloc	thrown by new on allocation failure
bad_cast	thrown by dynamic_cast when fails with a referenced type
bad_exception	thrown when an exception type doesn't match any catch
bad_typeid	thrown by typeid
ios_base::failure	thrown by functions in the iostream library

Preprocessor debugging and assert

- Preprocessor constants
  - __FILE__, __LINE__, __TIME__, __DATE__
- **Assert:** Preprocessor macro taking a single expression, which it uses as condition.
  - If preprocessor variable NDEBUG is not defined, then assert evaluates condition.
  - If condition is false, assert writes a message and terminates program.