# **تقدم عمگرها**

زبان C++، عملگرها بر اساس اولویت وابستگی‌شان تعریف می‌شوند. اولویت عملگرها نشان دهنده‌ی این است که کدام عملگر در یک عبارت ابتدا اعمال می‌شود. برای مثال، عملگرهای جمع و تفریق اولویت کمتری نسبت به ضرب و تقسیم دارند.

برای توصیف گرامر C++ و تعیین اولویت وابستگی عملگرها، از مفهوم اولویت عملگر (precedence) و اولویت گروه‌های مشابه (associativity) استفاده می‌شود. برخی از اولویت‌های بالا و پایین‌تر عبارت‌ها به شرح زیر هستند:

* پرانتز: بالاترین اولویت
* \* و / و % اولویت بالاتری از + و - دارند.
* + و - اولویت بالاتری از << و >> دارند.
* << و >> اولویت بالاتری از < و > دارند.
* < و > و <= و =< اولویت بالاتری از == و != دارند.
* == و != اولویت بالاتری از & دارند.
* & اولویت بالاتری از ^ دارند.
* ^ اولویت بالاتری از | دارند.
* | اولویت بالاتری از && دارند.
* && اولویت بالاتری از || دارند.
* || اولویت بالاتری از : ؟ (conditional operator) دارند.

برای توصیف اولویت وابستگی، از گروه‌های چپ به راست (left-to-right) یا راست به چپ (right-to-left) استفاده می‌شود. به عنوان مثال،

= یک عملگر راست به چپ است که برای انتساب مقدار به یک متغیر استفاده می‌شود.  
  
در زیر یک گرامر ساده ترکیب‌های اصلی برای جمع، تفریق، ضرب و تقسیم را تعریف می‌کند. گرامر زبان C++ به شکل مستندات رسمی (formal grammar) توصیف شده است. این گرامر شامل اصطلاحات نظیر ترمینال‌ها، نوع‌های داده، عبارات و دستورات است.

<expression> ::= <term> | <expression> + <term> | <expression> - <term>  
<term> ::= <factor> | <term> \* <factor> | <term> / <factor>  
<factor> ::= <number> | (<expression>)  
<number> ::= [0-9]+

در این توصیف:

<expression> یک عبارت را نمایش می‌دهد که می‌تواند شامل یک <term> باشد یا یک <expression>

همراه با + یا - و یک <term> باشد.

<term> یک عبارت حاوی یک <factor> است، یا یک <term> همراه با \* یا / و یک <factor> است .

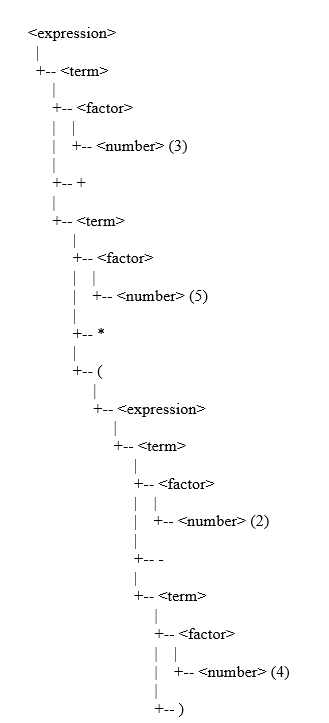
<factor> می‌تواند یک <number> باشد یا یک عبارت دیگر که درون پرانتز قرار دارد.

<number>یک عدد صحیح است.

یک مثال از استفاده این گرامر:

3 + 5 \* (2 - 4)

این عبارت با توجه به گرامر مشخص شده، به شکل زیر تجزیه و تحلیل می‌شود. این نمودار نشان می‌دهد که چگونه عبارت اولیه تجزیه و تحلیل می‌شود و هر قسمت از آن به کدام قاعده از گرامر متناظر می‌شود.



البته گرامر زبان C++ بسیار جزئی و پیچیده تر از گرامر بالاست. مثال پیچیده تری در ادامه آمده است. این گرامر برخی از عملگرهای اصلی را نشان می‌دهد:

<expression> ::= <assignment-expression>  
<assignment-expression> ::= <conditional-expression> | <logical-or-expression> assignment-operator <initializer-clause>  
<conditional-expression> ::= <logical-or-expression> | <logical-or-expression> ? <expression> : <assignment-expression>  
<logical-or-expression> ::= <logical-and-expression> | <logical-or-expression> || <logical-and-expression>  
<logical-and-expression> ::= <inclusive-or-expression> | <logical-and-expression> && <inclusive-or-expression>  
<inclusive-or-expression> ::= <exclusive-or-expression> | <inclusive-or-expression> | <exclusive-or-expression>  
<exclusive-or-expression> ::= <and-expression> | <exclusive-or-expression> ^ <and-expression>  
<and-expression> ::= <equality-expression> | <and-expression> & <equality-expression>  
<equality-expression> ::= <relational-expression> | <equality-expression> == <relational-expression> | <equality-expression> != <relational-expression>  
<relational-expression> ::= <shift-expression> | <relational-expression> < <shift-expression> | <relational-expression> > <shift-expression> | <relational-expression> <= <shift-expression> | <relational-expression> >= <shift-expression>  
<shift-expression> ::= <additive-expression> | <shift-expression> << <additive-expression> | <shift-expression> >> <additive-expression>  
<additive-expression> ::= <multiplicative-expression> | <additive-expression> + <multiplicative-expression> | <additive-expression> - <multiplicative-expression>  
<multiplicative-expression> ::= <unary-expression> | <multiplicative-expression> \* <unary-expression> | <multiplicative-expression> / <unary-expression> | <multiplicative-expression> % <unary-expression>  
<unary-expression> ::= <postfix-expression> | ++ <unary-expression> | -- <unary-expression> | unary-operator <cast-expression> | sizeof <unary-expression> | sizeof <type-id>  
<postfix-expression> ::= <primary-expression> | <postfix-expression> [ <expression> ] | <postfix-expression> ( <expression-list>? ) | <postfix-expression> . <identifier> | <postfix-expression> -> <identifier> | <postfix-expression> ++ | <postfix-expression> --  
<primary-expression> ::= <identifier> | <literal> | ( <expression> ) | <id-expression>  
<id-expression> ::= <unqualified-id> | <qualified-id>  
<unqualified-id> ::= <identifier> | <operator-function-id> | <literal-operator-id> | <template-id>  
<qualified-id> ::= <nested-name-specifier>? <unqualified-id>  
<nested-name-specifier> ::= <type-name> :: | <namespace-name> :: | decltype-specifier  
<literal> ::= <integer-literal> | <floating-point-literal> | <character-literal> | <string-literal> | <boolean-literal> | <pointer-literal>

این توصیف گرامر نشان می‌دهد چگونه عبارات در C++ ساخته می‌شوند، از جمله عبارات انتساب، شرطی، منطقی، بیتی، مقایسه، نقطه‌ویرگشتی و غیره. این فقط یک قسمت از گرامر C++ است و برای گرامر کامل‌تر، به منابع رسمی زبان C++ مانند استاندارد ISO C++ مراجعه می‌توان کرد.