C++ Text Nodes

Choose **Nodes> Node Dictionary> C++Text** from the menu bar to access the C++Text node dictionary. In this chapter:

Brackets

Brackets include parentheses, squares, and curly brackets:

`(,),[,],{,}`

(

Left parenthesis. In the following example, the left parenthesis has a body with a `i == 1` statement in it. The right parenthesis is directly next to the left one.

```
if (i == 1);
```

)

Right parenthesis. In the following example, the right parenthesis is directly next to the left parenthesis, which is in turn directly n ext to the `if` keyword (ignoring whitespace).

```
if (i == 1);
```

[

Left square bracket. In the following example, the left square bracket has a body with `10` constant in it. The right square bracket is directly next to the left square bracket.

```
int i[10];
```

]

Right square bracket. In the following example, the right square bracket is directly next to the left square bracket, which is in turn directly next to a identifier. The `10` constant is in the left square bracket's body.

```
int a[10];
```

{

Left curly bracket. In the following example, the left curly bracket has a body with function's `foo` statements. The right curly bracket is directly next to the left one.

```
int foo(){
 int i = 1;
 return i;
};
```

}

Right curly bracket. In the following example, the right curly bracket is directly next to the left curly bracket, which is in turn directly next to the right parenthesis. All statements in function's `foo` body are in the left curly bracket's body.

```
int foo(){
 int i = 1;
 return i;
};
```

Comments

C and C++ style comments:

/* , //

/*

C style comment. For example:

```
/* This is C style comment */
int i = 2; /*This is C style comment too */
```

//

C++ style comment. For example:

```
//This is C++ style comment
int i = 2; //This is C++ style comment too
```

Constants

The following constant expressions are included:

- Char Constant
- Identifier Constant
- Number Constant
- String Constant

Char Constant

Constant char expression. For example:

```
char one = '1'; // Char Constant " '1' "
...
if (type = 'd') { // Char Constant " 'd' "
...
```

Identifier Constant

Constant Identifier expression. For example:

```
static string msgQuit = "Quit message";
// Identifiers Constant: "static", "string", "msgQuit"
```

Number Constant

Constant Numerical expression. For example:

```
int one = 1; // Number Constant "1"
double pi = 3.14; // Number Constant "3.14"
```

String Constant

Constant string expression. For example:

```
char* one = "number one";
 // String Constant " "number one" "

string simpleText = "This is simple text";
 // String Constant " "This is simple text" "
```

Digraphs

Alternative token representations for some operators and punctuators.

%:, %>, :>, <%, <:

%:

Alternative token for #. For example:

```
%: define FOO // "%:" is an alternative for "#"
```

%>

Alternative token for }. For example:

```
void foo() {
%> // "%>" is an alternative for "}"
```

:>

Alternative token for]. For example:

```
int tab[10:>; // ":>" is an alternative for "]"
```

<%

Alternative token for {. For example:

```
void foo() <% // "<%" is an alternative for "{"
}
```

<:

Alternative token for [. For example:

```
int tab<:10];  // "<:" is an alternative for "["
```

Expressions

Includes all types of expression nodes.

- Assignment
- Bitwise
- |
- Comparison
- Logical

- Miscellaneous
- Numerical

Assignment

Assignment expressions include the following operators:

%=

Mod-equals operator. For example:

```
int i = 2;
int j = 3;
j %= i; // Mod-equals operator "%="
```

&=

Bitwise-and-equals operator. For example:

```
int i = 2;
int j = 3;
j &= i; // Bitwise-and-equals operator "&="
```

*=

Multiply-equals operator. For example:

```
int i = 2;
int j = 3;
j *= i; // Multiply-equals operator "*="
```

++

Increment operator. For example:

```
int i = 1;
i++; // increment operator "++"
++i; // increment operator "++"
```

+=

Plus-equals operator. For example:

```
int i = 1;
int j = 0;
j += i; // Plus-equals operator "+="
```

--

Decrement operator. For example:

```
int i = 10;
i--; // decrement operator "--"
--i; // decrement operator "--"
```

-=

Minus-equals operator. For example:

```
int i = 1;
int j = 0;
j -= i; // Minus-equals operator "-="
```

/=

Divide-equals operator. For example:

```
int i = 4;
int j = 2;
j /= i; // Divide-equals operator a/=b
```

<<=

Left-shift-equals operator. For example:

```
int i = 2;
int j = 3;
j <<= i; // Left-shift-equals operator "<<="
```

=

Assignment operator. For example:

```
int i = 1; // assignment operator "="
int j = 0; // assignment operator "="
j = i; // assignment operator "="
```

>>=

Right-shift-equals operator. For example:

```
int i = 2;
int j = 3;
j >>= i; // Right-shift-equals operator >>=
```

^=

Bitwise-xor-equals operator. For example:

```
int i = 2;
int j = 3;
j ^= i; // Bitwise-xor-equals operator "^="
```

|=

Bitwise-or-equals operator. For example:

```
int i = 2;
int j = 3;
j |= i; // Bitwise-or-equals operator "|="
```

Bitwise

The following bitwise operators are included:

&

Bitwise 'AND' operator. For example:

```
int z = 73, y = 0;
y = z & 0x0f; // Bitwise 'AND' operator "&"
```

^

Bitwise 'XOR' operator. For example:

```
int z = 73, y = 0;
y = z ^ 0x0f; // Bitwise 'XOR' operator "^"
```

|

Bitwise 'OR' operator. For example:

```
int x = 73, y = 0;
y = x | 4; // Bitwise 'OR' operator "|"
```

~

Bitwise 'NOT' operator. For example:

```
int a = ~3; // Bitwise 'NOT' operator "~"
```

Comparison

The following comparison operators are included:

!=

'Not-equal' operator. For example:

```
bool func(int x) {
 if (x!=3) { // 'Not-equal' operator "!="
 return false;
 }
 return true;
}
```

<

'Less than' operator. For example:

```
bool func(int x) {
 if (x<3) { // 'Less than' operator "<"
 return false;
 }
 return true;
}
```

<=

'Less than or equals' operator. For example:

```
bool func(int x) {
 if (x<=3) {
 // 'Less than or equals' operator "<="
 return false;
 }
 return true;
}
```

==

'Equality' operator. For example:

```
bool func(int x) {
 if (x==3) { // 'Equality' operator "=="
 return false;
 }
 return true;
}
```

>

'Greater than' operator. For example:

```
bool func(int x) {
 if (x > 3) { // 'Greater than' operator ">"
 return false;
 }
 return true;
}
```

>=

'Greater than or equals' operator. For example:

```
bool func(int x) {
 if (x >= 3) { // 'Greater than or equals' operator ">="
 return false;
 }
 return true;
}
```

Logical

The following logical operator expressions are included:

!

Logical 'NOT' operator. For example:

```
bool func(int x) {
 if (!x) {
 // Logical 'NOT' operator "!"
 return false;
 }
 return true;
}
```

&&

Logical 'AND' operator. For example:

```
bool func(int x, int y) {
 if (x && y) { // Logical 'AND' operator "&&"
 return false;
 }
 return true;
}
```

||

Logical 'OR' expression. For example:

```
bool func(int x, int y) {
 if (x || y) { // Logical 'OR' expression "||"
 return false;
 }
 return true;
}
```

Miscellaneous

The following miscellaneous expressions are included:

->

Dereference operator. For example:

```
class A {
public:
 int field;
};

main() {
 A *a = new A;
 a->field = 1; // Dereference operator "->"
}
```

->*

Arrow-star operator. For example:

```
class Foo {
public:
 void func();
};
typedef void(Foo::*mpf)();
void bar(Foo *f) {
 mpf ampf = &Foo::func;
 (f->*ampf)(); // Arrow-star operator "->*"
}
```

.

Dot operator. For example:

```
class A {
public:
 int foo;
}

[...]

a.foo = 1; //Dot operator.
```

::

Double colon operator. For example:

```
class A {
 int foo();
};

int A::foo() { // Double colon operator (A::foo())
 return 1;
}
```

?

Ternary operator. For example:

```
bool func(int x) {
 return x ? true : false; // Ternary operator "?"
}
```

.*

Pointer-to-member operator. For example:

```
class A {
public:
void foo();
};
typedef void (A::*PtrToMember)();
void bar(A& a, PtrToMember ptr) {
 (a.*ptr)(); // ".*" operator
}
```

...

Ellipsis. For example:

```
void foo(...); /* ellipsis */
```

Numerical

The following numerical expressions are included:

%

Mod operator. For example:

```
int j = 11;
int i = j % 2; //Mod operator (j % 2)
```

***

Asterisk operator. For example:

```
int plus1 = 2 * 2;
 //Asterisk operator (multiplication)
int *plus2 = &plus1;
 //Asterisk operator (pointer type)
```

+

Plus operator. For example:

```
int j = 2;
int plus1 = 1 + j; //Plus operator (addition)
int plus2 = +1; //Plus operator (meaningless)
```

-

Minus operator. For example:

```
int j = 2;
int minus1 = 1 - j; //Minus operator (substraction)
int minus2 = -1; //Minus operator (negation)
```

/

Division operator. For example:

```
float f = 3;
float r = f / 2; //Division operator
```

<<

Binary shift-left operator. For example:

```
int i = 2;
int j = i << 1; //shift-left operator (i << 1)
```

>>

Binary shift-right operator. For example:

```
int i = 2;
int j = i >> 1; //shift-right operator (i >> 1)
```

General

General nodes

File

File containing source code.

Lines

General line nodes.

Line

Single line. For example:

```
//first line (#line = 1)

//third line (as second is empty)
//access this comment using "All" line's property
```

Preprocessor Directives

The following preprocessor directives are included.

- #define
- #elif
- #else
- #endif
- #if
- #ifdef
- #ifndef
- #include
- #pragma
- #undef

#define

The `#define` preprocessor directive. For example:

```
#define PI 3.14159 //"#define" keyword
```

#elif

The `#elif` preprocessor directive. For example:

```
#if defined ASSERT_STDERR
#define ASSERT(str) fprintf(stderr, str);

#elif defined ASSERT_STDOUT //"#elif" keyword

#define ASSERT(str) fprintf(stdout, str);
#else
#define ASSERT(str)
#endif
```

#else

The `#else` preprocessor directive. For example:

```
#ifdef DEBUG
#define ASSERT(str) fprintf(stderr, str);

#else //"#else" keyword

#define ASSERT(str)
#endif
```

#endif

The `#endif` preprocessor directive. For example:

```
#if defined DEBUG
#define ASSERT(str) fprintf(stderr, str);
#else
#define ASSERT(str)

#endif //"#endif" keyword
```

#if

The `#if` preprocessor directive. For example:

```
#if defined DEBUG //"#if" keyword

#define ASSERT(str) fprintf(stderr, str);
#else
   #define ASSERT(str)
#endif
```

#ifdef

"#ifdef" preprocessor directive. For example:

```
#ifdef PI_NEEDED //"#ifdef" keyword
#define PI 3.14159
#endif
```

#ifndef

The `#ifndef` preprocessor directive. For example:

```
#ifndef PI //"#ifndef" keyword
#define PI 3.14159
#endif
```

#include

The `#include` preprocessor directive. For example:

```
#include <stdio.h> //#include" preprocessor directive.
```

#pragma

The `#pragma` preprocessor directive. For example:

```
#pragma align 8 //"#pragma" keyword
```

#undef

The `#undef` preprocessor directive. For example:

```
#ifdef PI
#undef PI //"#undef" keyword
#endif
```

Statements

General node of all types of statements.

- break

break

`break` keyword. For example:

```
switch(i) {
case 1:
 break; //"break" keyword
}
```

case

`case` keyword. For example:

```
switch(i) {
case 1: //"case" keyword
 break;
}
```

catch

`catch` keyword. For example:

```
try {

 do_something();

} catch(...) { //"catch" keyword.
}
```

continue

`continue` keyword. For example:

```
while (i < 10)
{
 i++;
 if (i == 5)
 continue; //"continue" keyword

 do_something();
}
```

default

`default` keyword. For example:

```
switch(i) {
case 1:
 do_something();
 break;
default: //"default" keyword
 do_something_else();
 break;
}
```

do

`do` keyword. For example:

```
do { //"do" keyword

 i++;

} while (i < 10);
```

else

`else` keyword. For example:

```
if (i == 0) {

 do_something();

} else { //"else" keyword

 do_something_else();

}
```

for

`for` keyword. For example:

```
for (i = 0; i < 10; i++) { //"for" keyword
 do_something();
}
```

goto

`goto` keyword. For example:

```
if (i == 0) {
 goto label1; //"goto" keyword.
}

label1:
 do_something_else();
```

if

`if` keyword. For example:

```
if (i == 0) { //"if" keyword

 do_something();

} else {

 do_something_else();

}
```

return

`return` keyword. For example:

```
int foo() {
 return 1; //"return" keyword.
}
```

switch

`switch` keyword. For example:

```
switch(i) { //"switch" keyword.

case 1:
 do_something();
 break;
default:
 do_something_else();
 break;
}
```

try

`try` keyword. For example:

```
try { //"try" keyword.

 do_something();

} catch(...) {
}
```

while

`while` keyword. For example:

```
do {
 i++;
} while (i < 10); //"while" keyword

while (i < 10) { //"while" keyword
 i++;
}
```

Tokens

Tokens are basic language elements.

Token

This is a basic lexical object recognized by lexer. Example statement:

```
int Table[10]; // this is statement
```

The following tokens are objects in this example statement:

- `int`
- *(single space)*
- `Table`
- `[`
- `10`
- `]`
- `;`
- `// this is statement.`

Notice that the comment (`// this is statement`) is a single token.

Trigraphs

Trigraphs are special sequences of three characters. Trigraphs are replaced with a corresponding single character when they appear in a source file.

- ??/
- ??)
- ??'
- ??<
- ??!
- ??>
- ??-

??/

Trigraph sequence replaced with `\`. For example:

```
char c = '??/a'; // "??/" is replaced with "\"
```

??)

Trigraph sequence replaced with `]`. For example:

```
int tab [ 10 ??); // "??)" is replaced with "]"
```

??'

Trigraph sequence replaced with `^`. For example:

```
int x = 10 ??' 2; // "??'" is replaced with "^"
```

??<

Trigraph sequence replaced with `{`. For example:

```
void foo() ??< } // "??<" is replaced with "{"
```

??!

Trigraph sequence replaced with `|`. For example:

```
int x = 10 ??! 2; // "??!" is replaced with "|"
```

??>

Trigraph sequence replaced with `}`. For example:

```
void foo() { ??> // "??>" is replaced with "}"
```

??-

Trigraph sequence replaced with `~`. For example:

```
int x = ??- 10; // "??-" is replaced with "~"
```

Various

Various nodes not fitting in other categories.

`,`,`:`,`;`,`\`

,

Comma operator. For example:

```
void func() {
int a,b; // comma operator "," in declaration
for (int i = 0; i >=0, i < 10; i++) {
// comma operator "," inside for statement condition
}
}
```

:

Colon operator. For example:

```
class My {
public: // colon operator ":"
bool abs(int val) {
return (val >= 0) ? val : -val;
// colon operator ":"
}
};
```

;

Semicolon operator. For example:

```
void func() {
for (int i = 0; i < 10; i++) {
// Semicolon operator ";"
// inside for statement condition
}
}
```

\

Backslash. For example:

```
#define SUM(A,B) A\
// BackSlash "\" in macro definition +B
```

Whitespace

Tab and space characters.

Space

Space character. For example:

```
 do_something( );

/*
The call above has single space at start of line
and single space in parentheses body
*/
```

Tab

Tab character. For example:

```
        int i = 1;

/*
The statement above has a single tab at start of line
*/
```