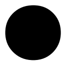
The Main Points of C++

Misha

Point



Point, as in:

4.2

1. Just one point (dot)

●

3-4

1. Just one point (dot)

●

- In floating-point literals

```
4.2
 .2
4.
```

followed by suffix
or exponent (e or **p**)

1. Just one point (dot)

●

- In floating-point literals
- Member access operators
  - Member of object

```
template<typename T>
struct S {
    template<typename U>
    void foo() {}
    in a;
};

template<typename T>
void bar() {
    S<T> x;
    x.a;
    s.template foo<T>();
}
```

1. Just one point (dot)

●

- In floating-point literals
- Member access operators
    - Member of object
    - Pointer to member of object

```
struct S {
    int mi;
    int f(int n) const;
};

auto pmi = &S::mi;
auto pf = &S::f;
S x{ 5 };
cout << x.*pmi << endl;
cout << (x.*pf)(6) << endl;
```

1. Just one point (dot)

.

- In floating-point literals
- Member access operators
  - Member of object
  - Pointer to member of object
- In module names

```
export module Main.SubM.SubSubM;
```

“Dots have no intrinsic meaning, however they are used informally to represent hierarchy.”

2. Two points -> colon

:

10-12

2. Two points -> colon

:

- Class inheritance

```
struct S : BaseClass {



};
```

2. Two points -> colon

:

- Class inheritance
- Member access specifiers

```
struct S : BaseClass {

private:
    int x;
};
```

2. Two points -> colon

:

- Class inheritance
- Member access specifiers
- Member initializer list

```
struct S : BaseClass {
    S(int xx) : x(xx) {}
private:
    int x;
};
```

2. Two points -> colon

:

- Class inheritance
- Member access specifiers
- Member initializer list
- Labels

```
switch (x) {

  if (y == 1)
case 1:
    cout << 7 << endl;
  else
case 2:
    cout << 8 << endl;
default:

}
```

2. Two points -> colon

:

- Class inheritance
- Member access specifiers
- Member initializer list
- Labels
  - but also **goto**

```
switch (x) {
label:
  if (y == 1)
case 1:
    cout << 7 << endl;
  else
case 2:
    cout << 8 << endl;
default:
  goto label;
}
```

2. Two points -> colon

:

- Class inheritance
- Member access specifiers
- Member initializer list
- Labels
  - but also **goto**
- Ternary

```
string str = (2+2 == 4)  ?  "OK"  :  throw logic_error("2+2 != 4");
```

2. Two points -> colon

:

- Class inheritance
- Member access specifiers
- Member initializer list
- Labels
    - but also **goto**
- Ternary
- Range **for**

```
for (auto it : container) {

}
```

2. Two points -> colon

:

- Class inheritance
- Member access specifiers
- Member initializer list
- Labels
  - but also **goto**
- Ternary
- Range **for**
- **enum** underlying type

```
enum Color : long long {
    red,
    green,
    blue, // <- extra comma is OK
};
```

2. Two points -> colon

:

- Class inheritance
- Member access specifiers
- Member initializer list
- Labels
  - but also **goto**
- Ternary
- Range **for**
- **enum** underlying type
- Bit-fields

```
struct S {
  int a : 3;
  int   : 0; // a new byte
  int   : 2; // skips 2 bits
  int b : 4 = 7; // can initialize
  int c : 5 {6}; // since C++20
};
```

2. Two points -> colon

:

- Class inheritance
- Member access specifiers
- Member initializer list
- Labels
    - but also **goto**
- Ternary
- Range **for**
- **enum** underlying type
- Bit-fields
- Attribute specifier

```
[[using gnu : always_inline, hot]]


// same as this:
[[gnu::always_inline, gnu::hot]]
```

2. Two points -> colon

:

- Class inheritance
- Member access specifiers
- Member initializer list
- Labels
  - but also **goto**
- Ternary
- Range **for**
- **enum** underlying type
- Bit-fields
- Attribute specifier
- In modules (private fragment and partitions)

```
module X:A; // in one file

module X:B; // another file

export module X; // third file
import :A;
export import :B;
```

```
module : private;
// starts the private part
```

2. Two points -> colon

:

- Class inheritance
- Member access specifiers
- Member initializer list
- Labels
  - but also **goto**
- Ternary
- Range **for**
- **enum** underlying type
- Bit-fields
- Attribute specifier
- In modules (private fragment and partitions)
- (asm declaration)

```
asm ("leal (%0,%0,4),%0“  :  "=r"(n)  :  "0"(n));
```

3. Three points -> ellipsis

• • •

6

3. Three points -> ellipsis

• • •

- Variadic functions

```
int printx(const char* fmt, ...);
int printy(const char* fmt ...);

#include <cstdarg>
// to access extra arguments

printx(“fs“, a, b, c);
printx(“fs“, a);

int foo(...);
// lowest priority in O.R.
// useful for SFINAE
// cannot access the arguments
```

3. Three points -> ellipsis

...

- Variadic functions
- Variadic templates (parameter pack)

```
template<class... Args> // template pp
void foo(Args... args) // function pp
{

}
```

3. Three points -> ellipsis

...

- Variadic functions
- Variadic templates (parameter pack)
- Pack expansion

```
template<class... Args> // template pp
void foo(Args... args) // function pp
{
    bar(args...);
}

// but it can happen in many different
// contexts, i.e. places, with many
// different useful properties
```

3. Three points -> ellipsis

• • •

- Variadic functions
- Variadic templates (parameter pack)
- Pack expansion
- **sizeof...**

```
template<typename... As>
auto average(As... as) {
  return sum(as...) / sizeof...(as);
}
```

3. Three points -> ellipsis

...

- Variadic functions
- Variadic templates (parameter pack)
- Pack expansion
- **sizeof...**
- **catch** all

```
try {


} catch (...) {


}
```

3. Three points -> ellipsis

…

- Variadic functions
- Variadic templates (parameter pack)
- Pack expansion
- **sizeof...**
- **catch** all
- Variadic macro

```
#define TWO(...) \
    foo(__VA_ARGS__); \
    bar(x_g __VA_OPT__(,) __VA_ARGS__)

TWO();
  foo();
  bar(x_g);

TWO(a);
  foo(a);
  bar(x_g, a);

TWO(a, b);
  foo(a, b);
  bar(x_g, a, b);
```

4. Four points -> double colon

∷

3-4

4. Four points -> double colon

::

- Scope resolution
  - Qualified names

```
Do I need an example?
```

4. Four points -> double colon

∷

- Scope resolution
  - Qualified names
  - Pointer to member

```
struct S {
    int mi;
    int f(int n) const;
};

int S::* pmi = &S::mi;
int (S::* pf)(int) = &S::f;

S x{ 5 };

cout << x.*pmi << endl;
cout << (x.*pf)(6) << endl;
```

4. Four points -> double colon

- Scope resolution
  - Qualified names
  - Pointer to member
- Nested namespace definition

```
namespace N1::N2::N3 {

}

// instead of
namespace N1 {
  namespace N2 {
    namespace N3 {

    }
  }
}
```

4. Four points -> double colon

∷

- Scope resolution
  - Qualified names
  - Pointer to member
- Nested namespace definition
- Attribute namespace

```
[[gnu::may_alias]]
```

4. Four points -> double colon

::

- Scope resolution
  - Qualified names
  - Pointer to member
- Nested namespace definition
- Attribute namespace

- Reflections -> splicer (P1240R2)

```
#include <meta>
template<Enum T>
std::string to_string(T value) {
  template for (
    constexpr auto e : meta::members_of(^T)) {

    if ([:e:] == value) {
      return string(meta::name_of(e));
    }
  }
  return "<unnamed>";
}
```