


Explicit constructor specifier



```
class Blob final
{
public:
    explicit Blob(int size);
    Blob(const Blob& other) = delete;
    ~Blob();
    Blob& operator=(const Blob& rhs) = delete;
private:
    int m_Size;
    int* m_pData;
};
```

Explicit constructor specifier

- The explicit specifier specifies that a constructor or conversion function (since C++11) doesn't allow **implicit conversions** or *copy-initialization*.
 - `Blob b1{5};`
 - `b1 = 30;` //implicit conversion from int to Blob
 - `Blob b2{b1};` //copy initialization
- A constructor with non-default parameters that is declared without the function specifier "explicit" is called a **converting constructor**.
- A **converting constructor** specifies an **implicit conversion** from the types of its arguments (if any) to the type of its class.

Without an explicit constructor specifier

```
void PrintBlob(const Blob& b)
{
    std::cout << "Blob" << b << '\n';
}
int main()
{
    PrintBlob(30);
    Blob b4 = 40;
}
```

```
class Blob final
{
public:
    Blob(int size);
    ~Blob();
private:
    int m_Size;
    int* m_pData;
};
```

PrintBlob(30):

- A Blob object is created, the constructor of Blob is fired with 30 as argument.
- The Blob object is passed as an argument to PrintBlob.
- When PrintBlob is finished, the object loses scope and is destroyed.

Without an explicit constructor specifier

```
void PrintBlob(const Blob& b)
{
    std::cout << "Blob" << b << '\n';
}
int main()
{
    PrintBlob(30);
    Blob b4 = 40;
}
```

Blob b4 = 40:

- A Blob object is created, the constructor of Blob is fired with 40 as argument.

```
class Blob final
{
public:
    Blob(int size);
    ~Blob();
private:
    int m_Size;
    int* m_pData;
};
```

Without an explicit constructor specifier

```
void PrintBlob(const Blob& b)
{
    std::cout << "Blob" << b << '\n';
}
int main()
{
    PrintBlob(30);
    Blob b4 = 40;
}
```

```
class Blob final
{
public:
    Blob(int size);
    ~Blob();
private:
    int m_Size;
    int* m_pData;
};
```

In both cases the integer is implicitly converted to an object of the class Blob.

In most cases this is not always desired behavior.

The explicit specifier inhibits implicit typecasting.

With an explicit constructor specifier

```
void PrintBlob(const Blob& b)
{
    std::cout << "Blob" << b << '\n';
}
int main()
{
    PrintBlob(30); // error
    Blob b4 = 40; //error
}
```

The explicit specifier inhibits implicit typecasting.

error C2664: cannot convert argument 1 from 'int' to 'const Blob &'

error C2440: 'initializing': cannot convert from 'int' to 'Blob'

```
class Blob final
{
public:
    explicit Blob(int size);
    ~Blob();
private:
    int m_Size;
    int* m_pData;
};
```

```
struct Vector final{
    Vector(int x, int y);
    int x, y;
};

Vector::Vector(int x, int y)
    :x{x}, y{y}
{}

void Func(const Vector& v)
{
    std::cout << v.x << v.y << "\n";
}

int main()
{
    Vector v1{ 1,2 };
    Func({ 5,6 }); // yes
    Func(Vector{ 5,6 }); // yes
}
```

This is not only for
single argument
constructors (since
C++11)

```
struct Vector final{
    explicit Vector(int x, int y);
    int x, y;
};

Vector::Vector(int x, int y)
    :x{x}, y{y}
{}

void Func(const Vector& v)
{
    std::cout << v.x << v.y << "\n";
}

int main()
{
    Vector v1{ 1,2 };
    Func({ 5,6 }); // no
    Func(Vector{ 5,6 }); // yes
}
```

This is not only for
single argument
constructors (since
C++11)

Is an explicit copy constructor allowed?

- The explicit specifier specifies that a constructor or conversion function (since C++11) doesn't allow implicit conversions or *copy-initialization*.
- Copy initialization initializes an object from another object. Example:

```
explicit Blob(const Blob& other);
```

```
Blob b2 = b1; // error: Constructor for class 'Blob' is declared 'explicit'
```

```
Blob b3 = 30; // error: Constructor for class 'Blob' is declared 'explicit'
```

- Do **NOT** make the copy constructor explicit.

References

- <http://www.learncpp.com/>
- <https://msdn.microsoft.com/nl-be/library/dn457344.aspx>
- *The C++ Programming Language* (3 ed.). Addison-Wesley. 2000. pp. 283–4. [ISBN 978-0-201-70073-2](#)