**Unnamed struct vs Anonymous struct.**

*Unnamed struct* is allowed (type has no name)

void foo()

{

struct /\* No name \*/ {

float x;

float y;

} point;

point.x = 42;

}

or

struct Circles

{

struct /\* No name \*/ {

float x;

float y;

} center; // but a member name

float radius;

};

and later

Circle circle;

circle.center.x = 42.f;

but NOT *anonymous struct* (unnamed type and unnamed object)

struct InvalidCircle

{

struct /\* No name \*/ {

float centerX;

float centerY;

}; // No member either.

float radius;

};

Note: Some compilers allow *anonymous struct* as *extension*.

C++11

* *lamdba* can be seen as a special *unnamed struct*.
* decltype allows to retrieve the type of *unnamed struct*:
* decltype(circle.point) otherPoint;
* *unnamed struct* instance can be parameter of template method:
* void print\_square\_coordinates()
* {
* const struct {float x; float y;} points[] = {
* {-1, -1}, {-1, 1}, {1, -1}, {1, 1}
* };
* // for range relies on `template <class T, std::size\_t N> std::begin(T (&)[N])`
* for (const auto& point : points) {
* std::cout << "{" << point.x << ", " << point.y << "}\n";
* }
* decltype(points[0]) topRightCorner{1, 1};
* auto it = std::find(points, points + 4, topRightCorner);
* std::cout << "top right corner is the "
* << 1 + std::distance(points, it) << "th\n";
* }

All the standard text refers to creating an "unnamed struct":

struct {

int hi;

int bye;

} bar;

Just a nice friendly type, with no accessible name.

In a standard way, it could be instantiated as a member like this:

struct Foo {

struct {

int hi;

int bye;

} bar;

};

int main()

{

Foo f;

f.bar.hi = 3;

}

But an "anonymous struct" is subtly different — it's the combination of an "unnamed struct" and the fact that you magically get members out of it in the parent object:

struct Foo {

struct {

int hi;

int bye;

}; // <--- no member name!

};

int main()

{

Foo f;

f.hi = 3;

}

Converse to intuition, this does not merely create an unnamed struct that's nested witin Foo, but also automatically gives you an "anonymous member" of sorts which makes the members accessible within the parent object.

It is this functionality that is non-standard. GCC [*does* support it](http://gcc.gnu.org/onlinedocs/gcc/Unnamed-Fields.html), and so does Visual C++. Windows API headers make use of this feature by default, but you can specify that you don't want it by adding #define NONAMELESSUNION before including the Windows header files.

Compare with the *standard* functionality of "anonymous unions" which do a similar thing:

struct Foo {

union {

int hi;

int bye;

}; // <--- no member name!

};

int main()

{

Foo f;

f.hi = 3;

}

It appears that, though the term "unnamed" refers to the type (i.e. "the class" or "the struct") itself, the term "anonymous" refers instead to the actual instantiated member (using an older meaning of "the struct" that's closer to "an object of some structy type"). This was likely the root of your initial confusion.