C++: Pointer to class data member

I came across this strange code snippet which compiles fine:

```
class Car
{
    public:
    int speed;
};

int main()
{
    int Car::*pSpeed = &Car::speed;
    return 0;
}
```

Why does C++ have this pointer to a non-static data member of a class? **What** is the use of this strange pointer in real code?

c++ class pointers

edited Mar 22 '09 at 9:38

asked Mar 22 '09 at 9:03
Ashwin
12.9k 23 81 151

Here's where I found it, confused me too...but makes sense now: stackoverflow.com/a/982941/211160 – HostileFork Jul 12 '12 at 6:03

10 Answers

It's a "pointer to member" - the following code illustrates its use:

```
#include <iostream>
using namespace std;

class Car
{
    public:
    int speed;
};

int main()
{
    int Car::*pSpeed = &Car::speed;

    Car c1;
    c1.speed = 1;       // direct access
    cout << "speed is " << c1.speed << endl;
    c1.*pSpeed = 2;     // access via pointer to member
    cout << "speed is " << c1.speed << endl;
    return 0;
}
```

As to *why* you would want to do that, well it gives you another level of indirection that can solve some tricky problems. But to be honest, I've never had to use them in my own code.

Edit: I can't think off-hand of a convincing use for pointers to member data. Pointer to member functions can be used in pluggable architectures, but once again producing an example in a small space defeats me. The following is my best (untested) try - an Apply function that would do some pre &post processing before applying a user-selected member function to an object:

```
void Apply( SomeClass * c, SomeClass::*func() ) {
    // do hefty pre-call processing
    c->*func();  // call user specified function
    // do hefty post-call processing
}
```

edited Apr 12 at 9:50
Krizz
6,304 7 25

answered Mar 22 '09 at 9:13
anon

1 Could you show an example of a tricky situation where this is useful? Thanks. – Ashwin Mar 22 '09 at 9:31

I have an example of using pointer-to-member in a Traits class in another SO answer. – Mike DeSimone Apr 13 '11 at 19:08

An example is writing a "callback"-type class for some event-based system. CEGUI's UI event subscription system, for example, takes a templated callback that stores a pointer to a member function of your choosing, so that you can specify a method to handle the event. – Benji XVI Dec 28 '12 at 21:03

Another application are intrusive lists. The element type can tell the list what its next/prev pointers are. So the list does not use hard-coded names but can still use existing pointers:

```
// say this is some existing structure. And we want to use
// a list. We can tell it that the next pointer
// is apple::next.
struct apple {
    int data;
    apple * next;
};

// simple example of a minimal intrusive list. Could specify the
// member pointer as template argument too, if we wanted:
// template<typename E, E *E::*next_ptr>
template<typename E>
struct List {
    List(E *E::*next_ptr):head(0), next_ptr(next_ptr) { }

    void add(E &e) {
        // access its next pointer by the member pointer
        e.*next_ptr = head;
        head = &e;
    }

    E * head;
    E *E::*next_ptr;
};

int main() {
    List<apple> lst(&apple::next);

    apple a;
    lst.add(a);
}
```

edited Mar 23 '09 at 0:25

answered Mar 23 '09 at 0:19
Johannes Schaub - litb
202k 27 406 751

If this is truly a linked list wouldn't you want something like this: void add(E* e) { e->*next_ptr = head; head = e; } ?? – eeeeaaii Aug 25 '11 at 16:56

@eee I recommend you to read about reference parameters. What I did is basically equivalent to what you did. – Johannes Schaub - litb Aug 25 '11 at 18:55

+1 for your code example, but I didn't see any necessity for the use of pointer-to-member, any other example? – Alcott Aug 14 '12 at 8:32

You can later access this member, on any instance:

```
int main()
{
  int Car::*pSpeed = &Car::speed;
  Car myCar;
  Car yourCar;

  int mySpeed = myCar.*pSpeed;
  int yourSpeed = yourCar.*pSpeed;

  assert(mySpeed > yourSpeed); // ;-)

  return 0;
}
```

Note that you do need an instance to call it on, so it does not work like a delegate.
It is used rarely, I've needed it maybe once or twice in all my years.

Normally using an interface (i.e. a pure base class in C++) is the better design choice.

answered Mar 22 '09 at 9:10
peterchen
18.6k 6 36 99

But surely this is just bad practice? should do something like youcar.setspeed(mycar.getpspeed) – thecoshman Oct 6 '10 at 21:08

1 @thecoshman: entirely depends - hiding data members behind set/get methods is not encapsulation and merely a milkmaids attempt at interface abstraction. In many scenarios, "denormalization" to public members is a reasonable choice. But that discussion probably exceeds the confines of the comment functionality. – peterchen Oct 12 '10 at 15:21

IBM has some more documentation on how to use this. Briefly, you're using the pointer as an offset into the class. You can't use these pointers apart from the class they refer to, so:

```
  int Car::*pSpeed = &Car::speed;
  Car mycar;
  mycar.*pSpeed = 65;
```

It seems a little obscure, but one possible application is if you're trying to write code for deserializing generic data into many different object types, and your code needs to handle object types that it knows absolutely nothing about (for example, your code is in a library, and the objects into which you deserialize were created by a user of your library). The member pointers give you a generic, semi-legible way of referring to the individual data member offsets, without having to resort to typeless void * tricks the way you might for C structs.

edited Feb 18 at 4:54

answered Mar 22 '09 at 9:13
AHelps
1,021 5 8

Could you share a code snippet example where this construct is useful? Thanks. – Ashwin Mar 22 '09 at 9:32

1 +1 for finding a reasonable use case – dmckee Mar 22 '09 at 16:24

I'm currently doing alot of this due to doing some DCOM work and using managed resource classes which involves doing a bit of work before each call, and using data members for internal representation to send off to com, plus templating,makes a lot of boiler plate code *much* smaller – Dan Aug 10 '09 at 21:30

+1 for the key point: "Briefly, you're using the pointer as an offset into the class." – Samaursa Nov 29 '12 at 3:53

It makes it possible to bind member variables and functions in the uniform manner. The following is example with your Car class. More common usage would be binding `std::pair::first` and `::second` when using in STL algorithms and Boost on a map.

```
#include <list>
#include <algorithm>
#include <iostream>
#include <iterator>
#include <boost/lambda/lambda.hpp>
#include <boost/lambda/bind.hpp>


class Car {
public:
    Car(int s): speed(s) {}
    void drive() {
        std::cout << "Driving at " << speed << " km/h" << std::endl;
    }
    int speed;
};

int main() {

    using namespace std;
    using namespace boost::lambda;

    list<Car> l;
    l.push_back(Car(10));
    l.push_back(Car(140));
    l.push_back(Car(130));
    l.push_back(Car(60));

    // Speeding cars
    list<Car> s;

    // Binding a value to a member variable.
    // Find all cars with speed over 60 km/h.
    remove_copy_if(l.begin(), l.end(),
```

edited Mar 22 '09 at 13:08

answered Mar 22 '09 at 13:02
Alex B
26.6k 6 79 157

This is the simplest example I can think of that conveys the rare cases where this feature is pertinent:

```
#include <iostream>

class bowl {
public:
    int apples;
    int oranges;
};

int count_fruit(bowl * begin, bowl * end, int bowl::*fruit)
{
    int count = 0;
    for (bowl * iterator = begin; iterator != end; ++ iterator)
        count += iterator->*fruit;
    return count;
}

int main()
{
    bowl bowls[2] = {
        { 1, 2 },
        { 3, 5 }
    };
    std::cout << "I have " << count_fruit(bowls, bowls + 2, & bowl::apples) << " a
    std::cout << "I have " << count_fruit(bowls, bowls + 2, & bowl::oranges) << "
    return 0;
}
```

The thing to note here is the pointer passed in to count_fruit. This saves you having to write separate count_apples and count_oranges functions.

answered Apr 29 '11 at 16:05
JMcF
436 6 6

Here's a real-world example I am working on right now, from signal processing / control systems:

Suppose you have some structure that represents the data you are collecting:

```
struct Sample {
    time_t time;
    double value1;
    double value2;
    double value3;
};
```

Now suppose that you stuff them into a vector:

```
std::vector<Sample> samples;
... fill the vector ...
```

Now suppose that you want to calculate some function (say the mean) of one of the variables over a range of samples, and you want to factor this mean calculation into a function. The pointer-to-member makes it easy:

```
double Mean(std::vector<Sample>::const_iterator begin,
    std::vector<Sample>::const_iterator end,
    double Sample::* var)
{
    float mean = 0;
    int samples = 0;
    for(; begin != end; begin++) {
        const Sample& s = *begin;
        mean += s.*var;
        samples++;
    }
    mean /= samples;
    return mean;
}

...
double mean = Mean(samples.begin(), samples.end(), &Sample::value2);
```

And, of course, you can template it, though it gets a bit messy (I've recast it as a struct to get the typedefs in; I guess there'd be a way with a template function, but its a bit more readable this way):

```
template<typename T>
struct Mean {
    typedef std::vector<T> Tvector;
    typedef typename std::vector<T>::const_iterator Titer;
    double operator()(Titer begin, Titer end, double T::* var) {
        float sum = 0;
        int samples = 0;
        for( ; begin != end; begin++ ) {
            const T& s = *begin;
            sum += s.*var;
            samples++;
        }
        return sum / samples;
    }
};

...

Mean<Sample> m;
double mean = m(samples.begin(), samples.end(), &Sample::value2);
```

answered Nov 2 '10 at 13:17
Tom
453 4 11

This is excellent. I'm about to implement something very similar, and now I don't have to figure out the strange syntax! Thanks! – SchighSchagh Mar 26 at 2:47

You can use an array of pointer to (homogeneous) member data to enable a dual, named-member (i.e. x.data) and array-subscript (i.e. x[idx]) interface.

```
#include <cassert>
#include <cstddef>

struct vector3 {
    float x;
    float y;
    float z;

    float& operator[](std::size_t idx) {
        static float vector3::*component[3] = {
                &vector3::x, &vector3::y, &vector3::z
        };
        return this->*component[idx];
    }
};

int main()
{
    vector3 v = { 0.0f, 1.0f, 2.0f };

    assert(&v[0] == &v.x);
    assert(&v[1] == &v.y);
    assert(&v[2] == &v.z);

    for (std::size_t i = 0; i < 3; ++i) {
        v[i] += 1.0f;
    }

    assert(v.x == 1.0f);
    assert(v.y == 2.0f);
    assert(v.z == 3.0f);

    return 0;
}
```

edited Mar 23 '09 at 17:33 answered Mar 23 '09 at 4:24
Functastic
396 1 10

One way I've used it is if I have two implementations of how to do something in a class and I want to choose one at run-time without having to continually go through an if statement i.e.

```
class Algorithm
{
public:
    Algorithm() : m_impFn( &Algorithm::implementationA ) {}
    void frequentlyCalled()
    {
        // Avoid if ( using A ) else if ( using B ) type of thing
        (this->*m_impFn)();
    }
private:
    void implementationA() { /*...*/ }
    void implementationB() { /*...*/ }

    typedef void ( Algorithm::*IMP_FN ) ();
    IMP_FN m_impFn;
};
```

Obviously this is only practically useful if you feel the code is being hammered enough that the if statement is slowing things done eg. deep in the guts of some intensive algorithm somewhere. I still think it's more elegant than the if statement even in situations where it has no practical use but that's just my opnion.

edited Mar 23 '09 at 18:40 answered Mar 22 '09 at 11:57
Troubadour
8,467 1 10 21

I think you'd only want to do this if the member data was pretty large (e.g., an object of another pretty hefty class), and you have some external routine which only works on references to objects of that class. You don't want to copy the member object, so this lets you pass it around.

answered Mar 22 '09 at 10:47
Andrew Jaffe
6,758 12 22

Not the answer you're looking for? Browse other questions tagged c++ class pointers or ask your own question.