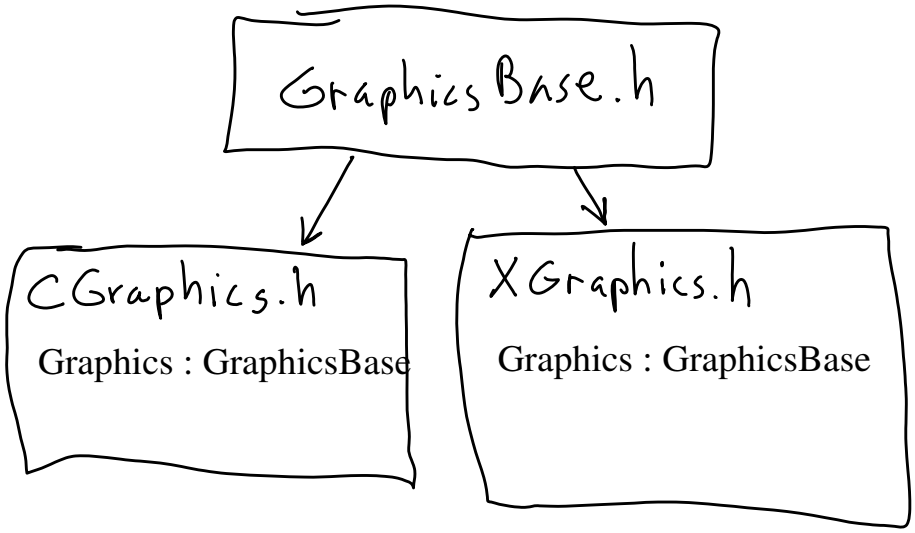


Syntax corner.----------
(see implicit.c)

```
float x;
x = 1;   ==>  x = (float) 1;

foo f;
f = 1;    ==>   f = foo(1) !?!;
```

If it's a bad idea to allow that, do this:

```
class foo {
   explicit foo( int x );
};
```

Makefile

```
simX:
        g++ -DXGRAPH sim.cpp ...
simC:
        g++ -DCGRAPH sim.cpp ...
```

```
% make simX
% make simC
```

sim.cpp

```
#ifdef XGRAPH
   #include "XGraphics.h"
#elif CGRAPH
   #include "CGraphics.h"
#else
   #error "Use C or X."
#endif
```

Q. Do we want both graphics types at once?

Q. How could we refactor to allow for this?

Holy Commandments:

- Thou shalt implement virtual methods (for thing.h
  - BUNNY
  - FOX)
- Thou shalt use vector, string
  multi-dimensional

graphicsBase.h
XGraphics.h → Implements Graphics ? → XGraphics
CGraphics.h → Implements Graphics ? → CGraphics

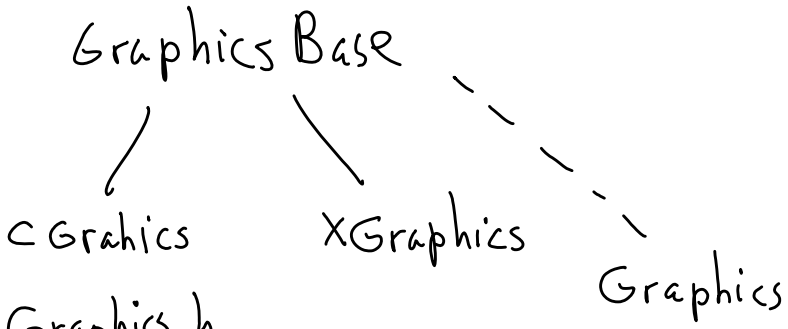
alternative conditional compilation

```
ifdef XGRAPH
   #include "XGraphics"
      ...
#if def XGRAPH
     XGraphics g;
#endif

#ifdef CGRAPH
     CGraphics g;
#endif
```

```
#ifdef BOTHGRAPH
     XGraphics xg;
     CGraphics cg;
#endif

#ifdef BOTHGRAPH
     init(xg, queue);
     init(cg, queue);
#endif
```

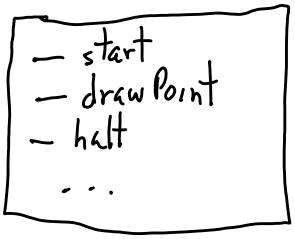
hmm.
Q. How bad is this?
Uses of "Graphics" in sim.cpp, others?

classes as wrappers?



Graphics.h

```
#ifdef BOTHGRAPH
#include "CGraphics.h"
#include "XGraphics.h"

class Graphics : GraphicsBase
    XGraphics xg;
    CGrahics cg;
```



overrides

```
Graphics::start()
    xg.start();
    cg.start();
```

```
#ifdef XGRAPH
#include "XGraphics.h"

class Graphics: XGraphics
```

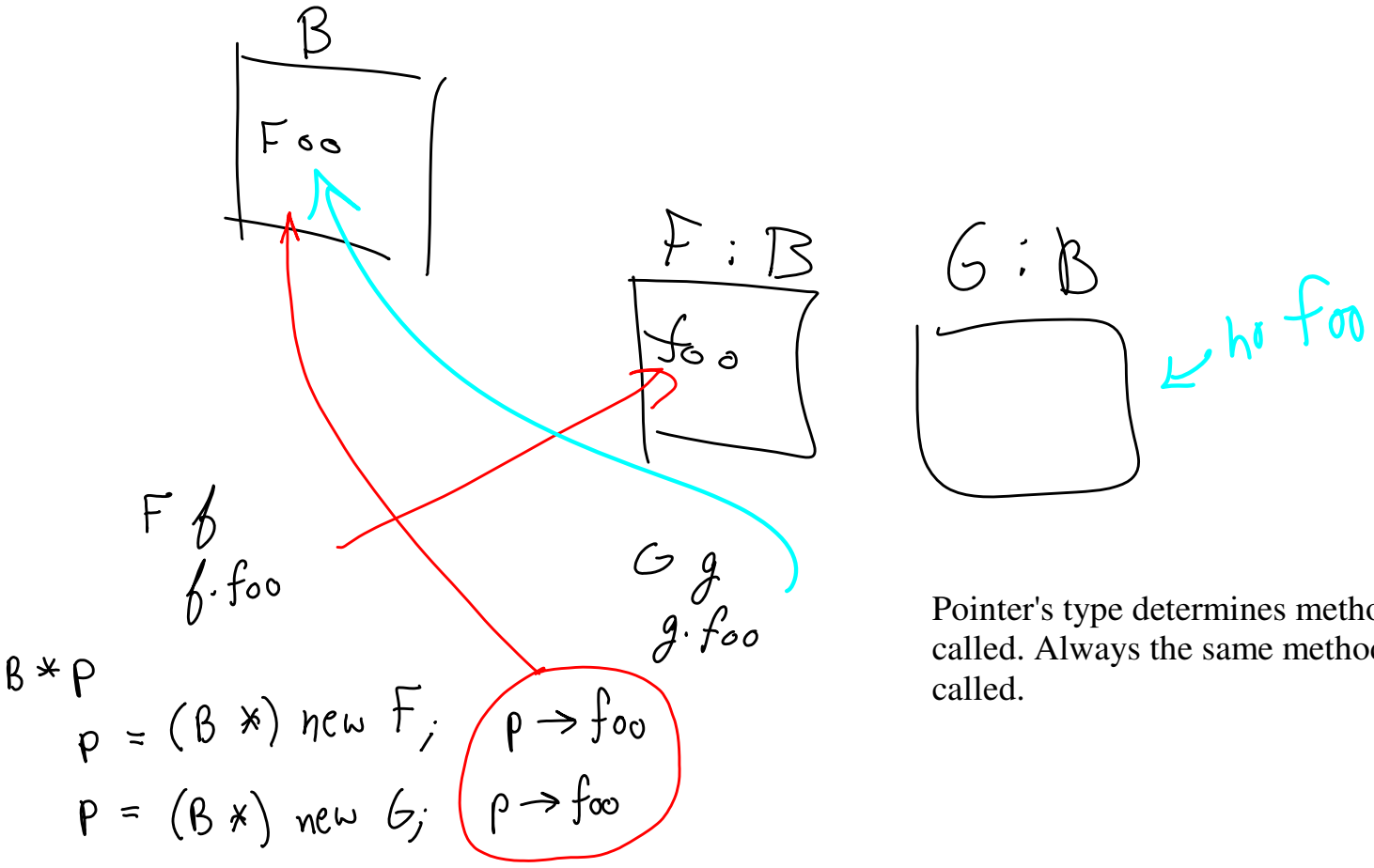
no overrides,

```
#ifdef CGRAPH
    class Graphics : CGraphics
        - no overrides
```

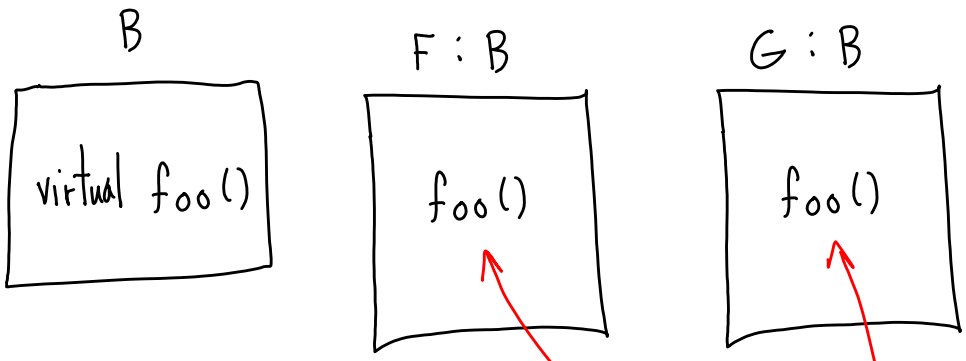
sim.cpp

```
#include "Graphics.h"

main
    Graphics g;
    g.start();
```

How should we override GraphicsBase class? Static or Virtual overrides?



Pointer's type determines method called. Always the same method called.



B * p

p = (B *) new F; ⇒ p->foo()

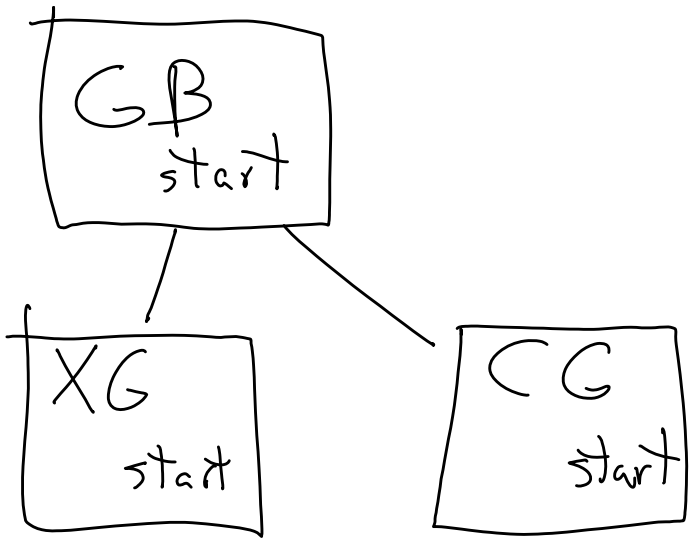
p = (B *) new G; ⇒ p->foo()

p is polymorphic : acts differently

class B now has a function pointer field:

(*foo())

Pointer filled in at instance creation. Points to F.foo or G.foo.



Multiple graphics objects of different types? Both accessed in same program at runtime? Via pointer? Is this what we need?

GB * p

p -> start()

p -> start()

acts differently

Inheritance (public, private, protected)

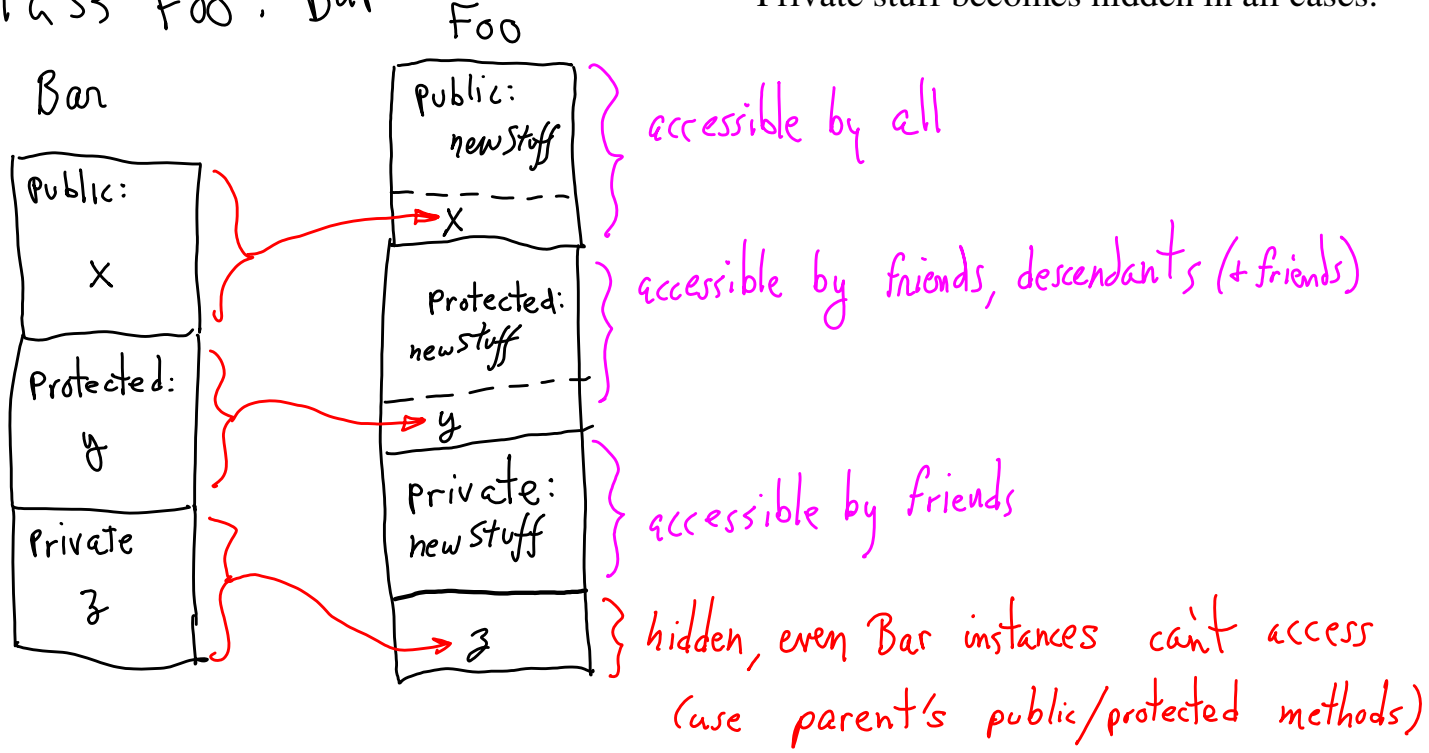
The question is,

Where does the public stuff go?
Where does the protected stuff go?

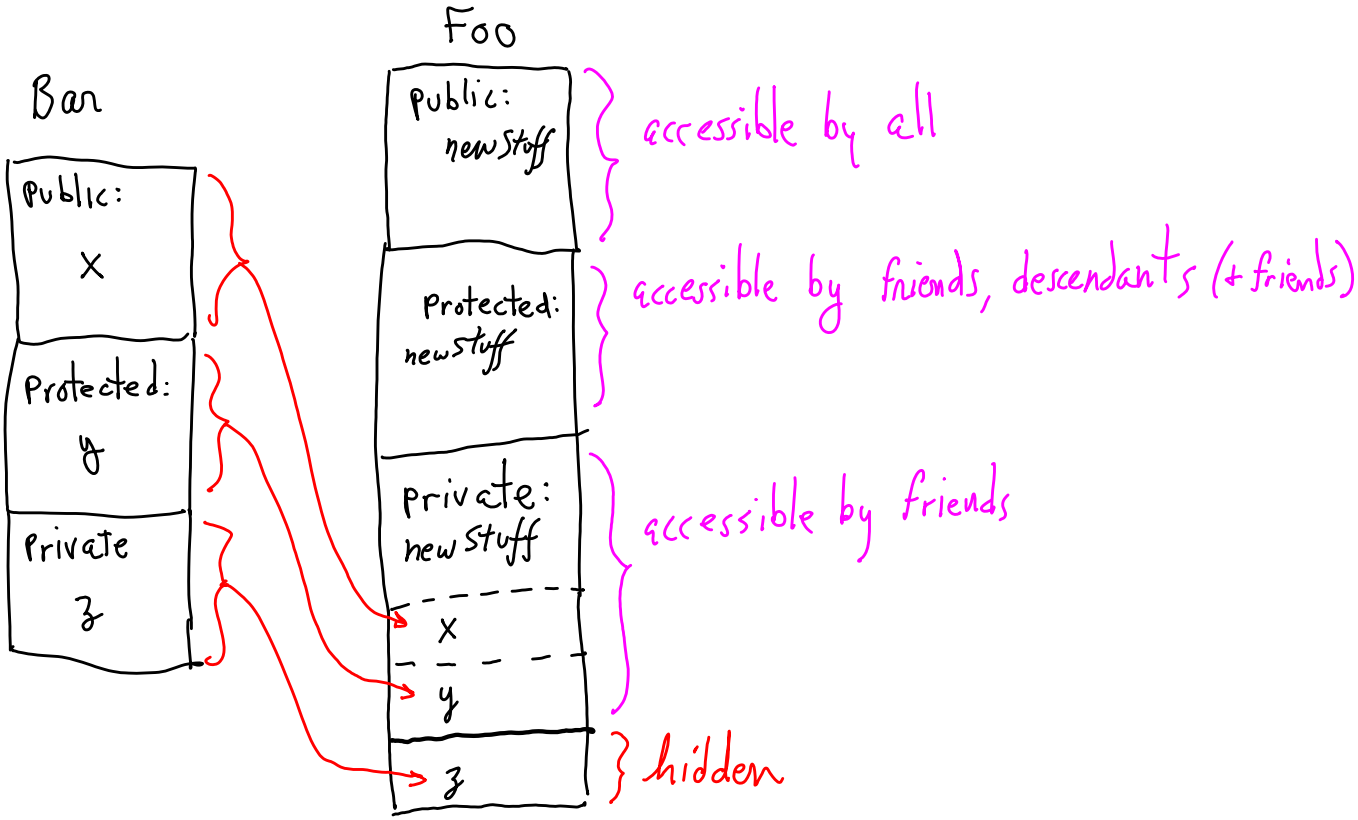
Private stuff becomes hidden in all cases.

class Foo : public Bar

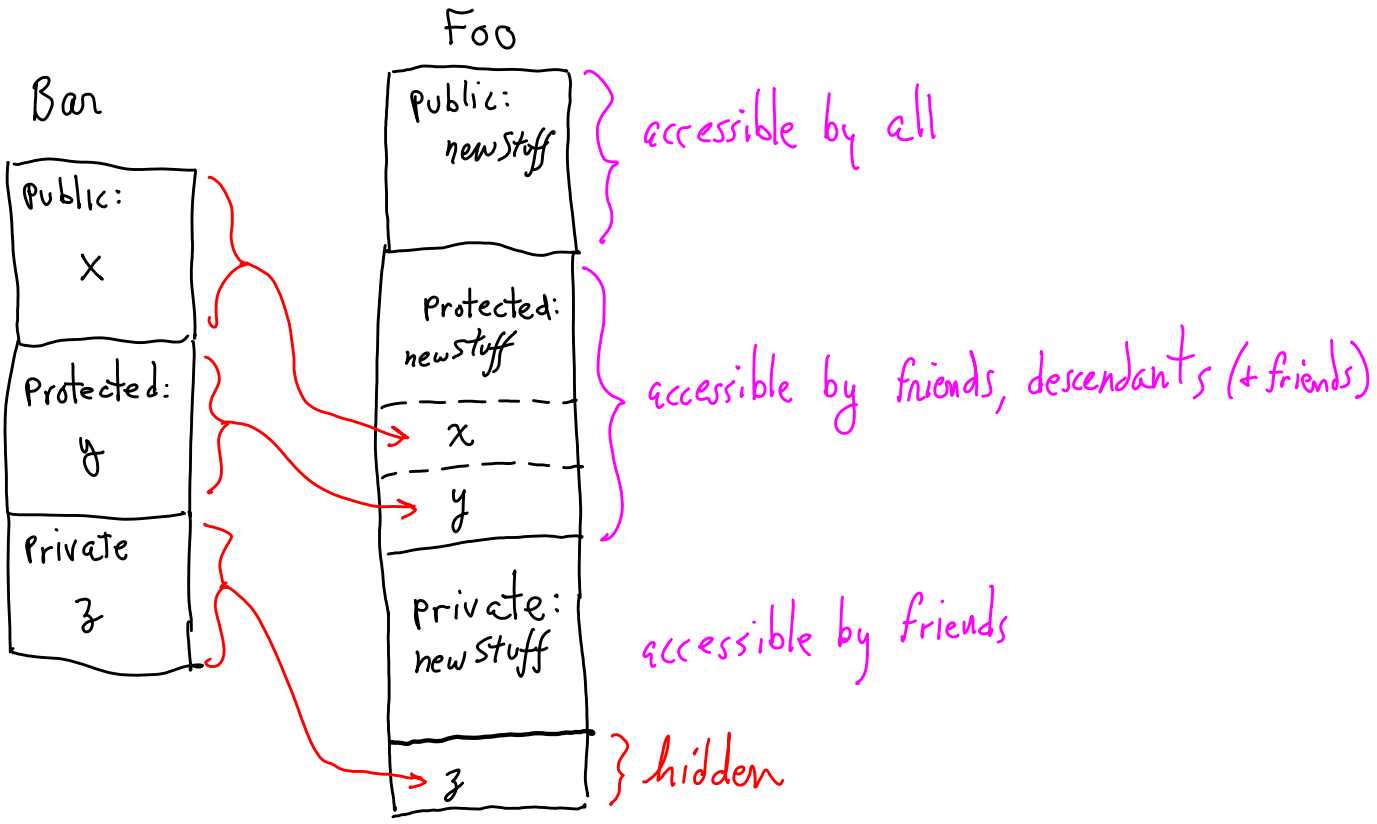
class Foo : Bar



class Foo : private Bar

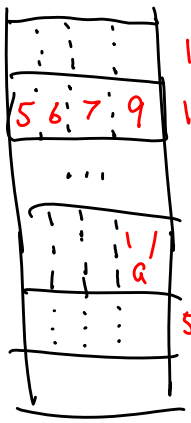


class Foo : protected Bar



Pointers, Arrays, Vectors, strings

```
int *p
char *c
char ch;
int n;
```



NB- byte addressing
8 bits/Byte
32-bit words (4 B)

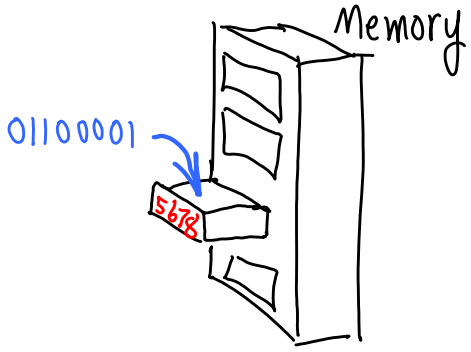
```
ch = 'a';
c = &ch;
c++;
*c = 'b';
```

Mem[5678] ⇐ 0x61
Mem[1238] ⇐ 5678
Mem[1238] ⇐ 5679
Mem[5679] ⇐ 0x62

ASCII codes

'a' = 0110 0001 = 0x61
1 Byte
'b' = 01100010 = 0x62

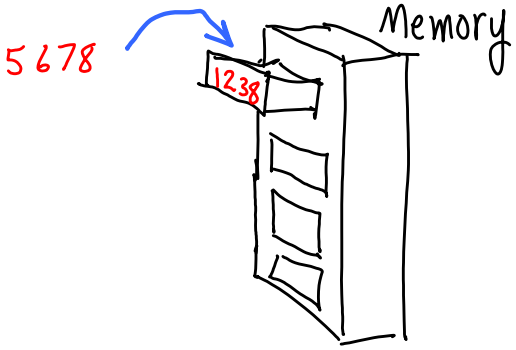
ch = 'a';
⇓
*(5678)
Mem[5678] ⇐ write 01100001



Symbol Table

name	address
p	1234
c	1238
ch	5678
n	5682

c = &ch;
⇓
*(1238)
Mem[1238] ⇐ write 5678



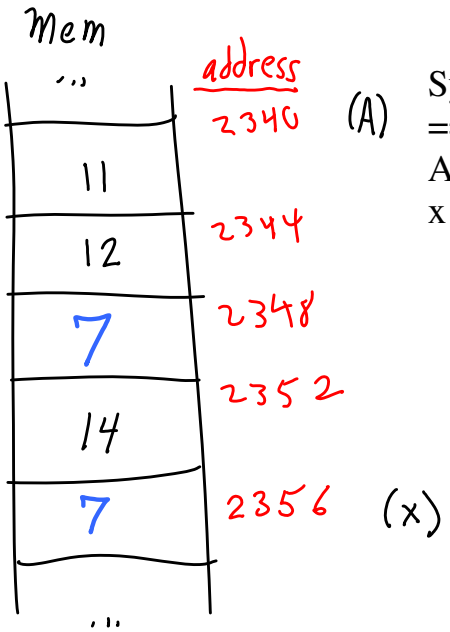
```
int x;
int A[4];

A[2]      =      x;
```

(2340) + (2 * sizeof(int))
4B

*(2348)

*(2356)

Mem[2348] ⟸ Mem[2356]
write read



Symbol Table
==========
A 2340
x 2356

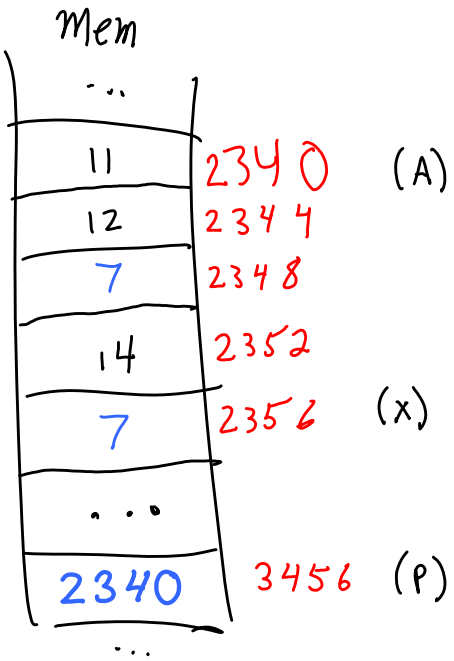
```
int *p;

p      =      A;
```

*(3456)

Mem[3456] ⟸ 2340
write

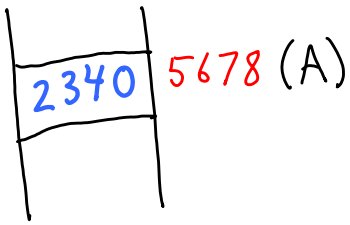
*(5678)

Mem[3456] ⟸ Mem[5678]
write read

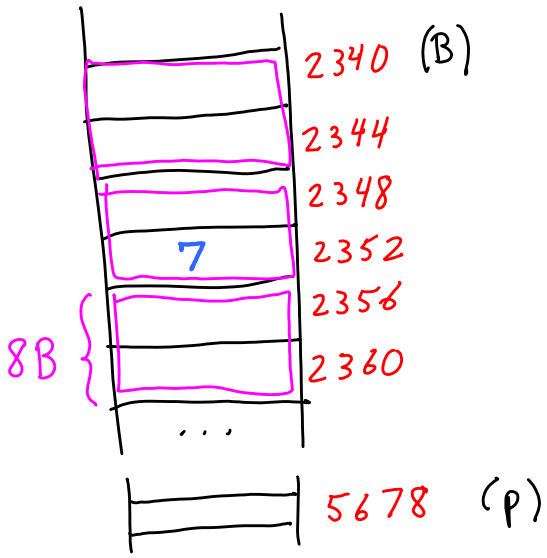


? shouldn't this be *(2340) ???

"A" is treated AS IF it were a pointer variable whose content was 2340. So, to be consistent, we could ASSUME it is:



```
int B[3][2]
int *p;
```




```
B[1][1]  =  7;
```

(2340) + (1 * 8B)

(2348) + (1 * 4B)

*(2352)

Mem[2352] ⇐ 7

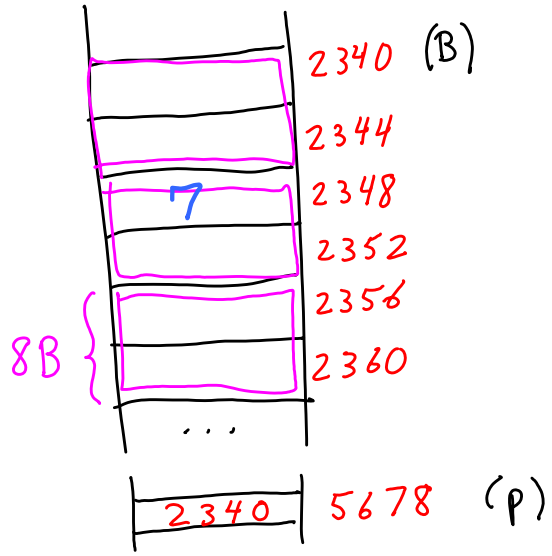
write

```
p  =  B;
```

*(5678)

Mem[5678] ⇐ 2340

write




```
*(p+2)  =  7;
```

*(5678)

Mem[5678]

read

(2340) + (2 * 4B)

*(2348)

Mem[2348] ⇐ 7

write

Bottom line:

Array names are not pointer variables, but they are sort of, syntactically.

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

int main()
{
    int A[4] = {1,2,3,4};
    int *p;
    int B[4][2] = {{1,2}, {3,4}, {5,6}, {7,8}};

    cout << "&(A[0])==" << (long) &(A[0]) << endl;
    cout << "&(A[1])==" << (long) &(A[1]) << endl;
    cout << "&(A[2])==" << (long) &(A[2]) << endl;
    cout << "&(A[3])==" << (long) &(A[3]) << endl;
    cout << endl;

    cout << "(A+0)==" <<  (long) (A+0) << endl;
    cout << "(A+1)==" <<  (long) (A+1) << endl;
    cout << "(A+2)==" <<  (long) (A+2) << endl;
    cout << "(A+3)==" <<  (long) (A+3) << endl;
    cout << endl;

    p = A;
    cout << "p = A" << endl;
    cout << "p==" << (long) p << endl;
```

```
&(A[0])==2280736
&(A[1])==2280740
&(A[2])==2280744
&(A[3])==2280748

(A+0)==2280736
(A+1)==2280740
(A+2)==2280744
(A+3)==2280748

p = A
p==2280736
&p==2280732
A==2280736
&A==2280736

(p+0)==2280736
(p+1)==2280740
(p+2)==2280744
(p+3)==2280748

*p==1
*A==1
*(p+1)==2
*(A+1)==2

B==2280688
&(B[0][0])==2280688

B[0]==2280688
B[1]==2280696
B[2]==2280704
B[3]==2280712

&(B[0])==2280688
&(B[1])==2280696
&(B[2])==2280704
&(B[3])==2280712

&(B[1][0])==2280696
&(B[2][0])==2280704
&(B[3][0])==2280712
```