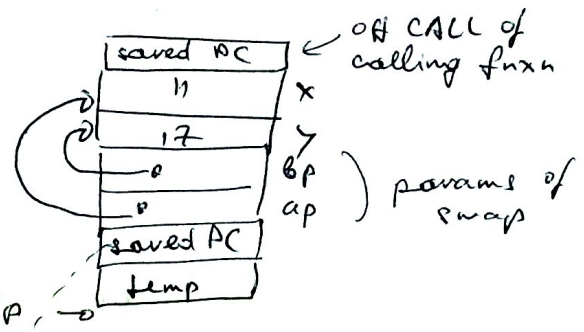
CS107 lecture 11 Assembly

```
void foo()
{
  int x;
  int y;
  x = 11;
  y = 17;
  swap(&x, &y);
}
```



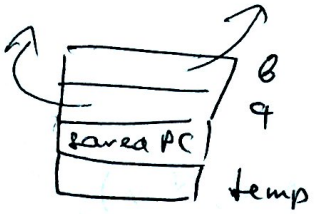

```
<foo>:
    SP = SP - 8;
    M[SP+4] = 11;
    M[SP] = 17;
    R1 = SP    ; &y
    R2 = SP+4  ; &x
    SP = SP - 8
    M[SP] = R2;
    M[SP+4] = R1
    CALL <swap>
    SP = SP + 8;
    SP = SP + 8;
    RET;
```

```
void swap(int *ap, int *bp)
{
  int temp = *ap;
  *ap = *bp;
  *bp = temp;
}
```

```
<swap>:
    SP = SP - 4    ;
    R1 = M[SP+8];
    R2 = M[R1]; # ap into
    M[SP] = R2 ; # in temp
    R1 = M[SP+12]; # bp
    R2 = M[R1] ; # bp into
    R3 = M[SP+8] ;
    M[R3] = R2 ;
    R1 = M[SP] ;
    R2 = M[SP+12] ;
    M[R2] = R1 ;
    SP = SP+4; # back to saved PC
    RET;
```

C++ version of swap

```
void swap (int &a, int &b)
{
  int temp = a;
  a = b;
  b = temp;
}
```



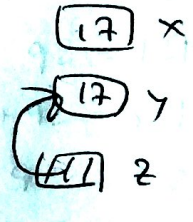
exactly the same Assembly code

```
int x;
int y;
x = 11;
y = 17
swap(x, y);
```

compiler interpolates [redacted] that it needs to get reference based on prototype

exactly the same Assembly

```
int x = 17;
int y = x;
int &z = y;
int *z = &y;
```



Structs & classes in C++

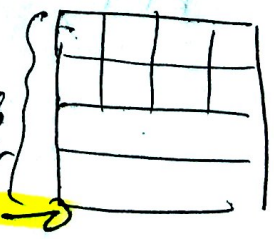
- layed out in memory exactly the same way
- both have constructors, destructors and methods
- default access modifier: struct → public
  class → private

```
class blinky {
public:
  int dunky (int x, int y);
  char * minky (int * z) {
    int w = *z;
    return slinky + dunky (winky, winky); }

private:
  int winky;
  char * blinky;
  char slinky [83];
};
```

```
blinky b;
b.minky (&n);
```

b and this pointer

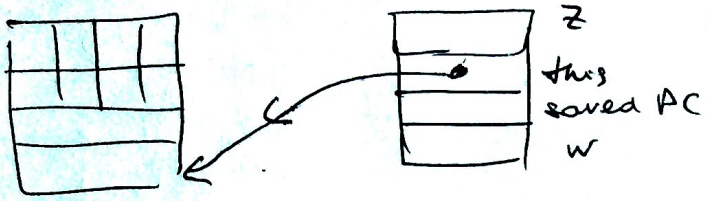


access to slinky and winky because have the pointer to the [redacted] figure / object

this pointer, implicitly passed

b.minky(&n); // binky::minky(&b, &n);

object reference passed implicitly



meaning of static method in a C++ class:
no need to [redacted] an instance of class to invoke it; no need for this pointer to be passed.
↳ actually no point being inside a class definition. → avoid

Compilation

Preprocessor:

```
# define  kWidth   480
# define  kHeight  720
          a        b
{
printf("Width is %d.\n", kWidth);
int area = kWidth * kHeight;
```

preprocessor only pays attention to # {define, include...}
associates a with b (as text) as a hashset and reads the code and substitutes a with b

output: .c file where all # - have been substituted as text.