APS105
Winter 2012

Jonathan Deber
jdeber -at- cs -dot- toronto -dot- edu

Lecture 14
February 15, 2012

Today

- More Pointers

Assignment 2

- Make sure you understand how the starter code works
- When working with arrays, draw pictures
- Don't change other parts of the starter code

Documentation

- In order to use a function you need to know how it works
- In order to write a function you need to know how it works

- Two parts:
  - Prototype (in C)
  - Specification (in English)

Prototype

- Tells us the return type and the number and type of parameters

```
int calculateXPosition(int xPos, char direction, int distance);
```

```
 int                  int      char
```

Prototype

- Tells us the return type and the number and type of parameters

```
int calculateXPosition(int xPos, char direction, int distance);
```

```
 int                  int       char       int
```

Specification

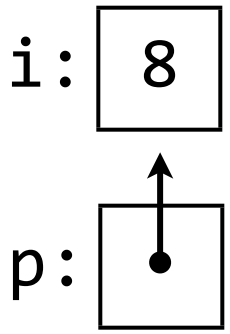
Then, fill in `calculateXPosition()`. Remember what this function does: given the current x position, the direction of travel, and a distance, it calculates the new x position after moving the pen. This means that we might need to add or subtract the distance from our current position, or do nothing to it if the movement is perpendicular to the x-axis. Also keep in mind that there should never be an x coordinate that is outside of the canvas; we stop when we hit the edge.

```
/* Calculates the new x position caused by a move of 'distance'
   in 'direction'. If the move is N or S, the x position will be
   unchanged.  The move will not cause the pen to move past the
   edge of the canvas; if the move would result in the pen
   moving off the left side (past position 0) or right side
   (past position COLS - 1), the new position is the edge of the
   canvas.  For more details, see the assignment handout. */
int calculateXPosition(int xPos, char direction, int distance);
```

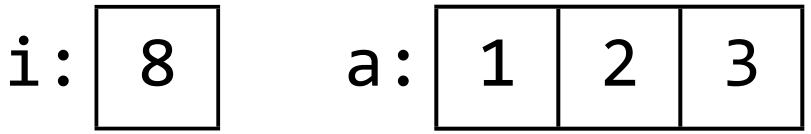
Prototype	Specification
`5`	5 squares North
`M_PI / 2`	`(M_PI / 2)` radians

Pointers

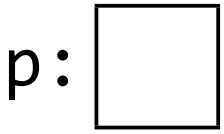
- A pointer is a variable that holds a memory address
- It "points at" another variable



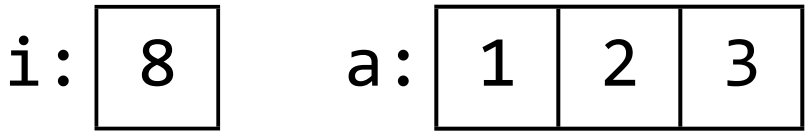
```
int i = 8;
int a[] = {1, 2, 3};
```



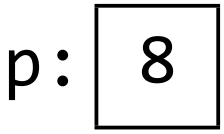

```
int *p;
```



```
int i = 8;
int a[] = {1, 2, 3};
```




```
int *p;
```

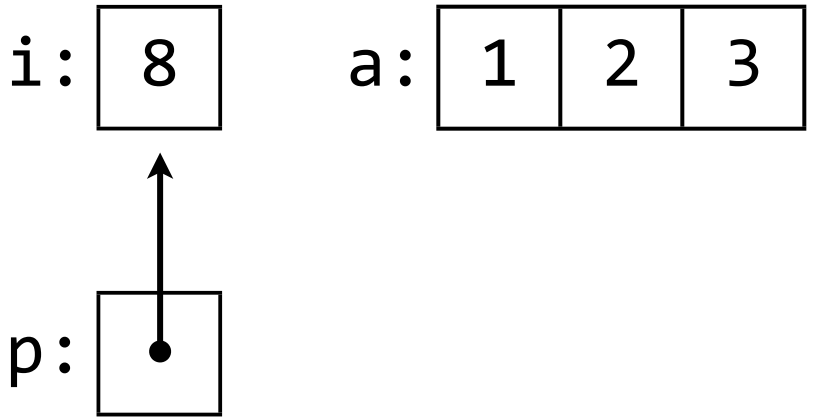



```
p = i;
```

Wrong

8 is an `int`, not an arrow!

```
int i = 8;
int a[] = {1, 2, 3};
```




```
int *p;
```

```
p = i;
```

Wrong

`8` is an `int`, not an arrow!

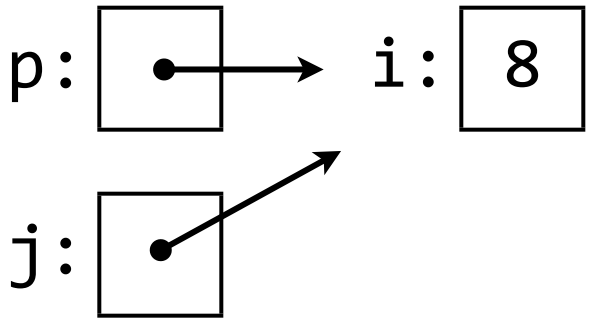
```
p = &i;
```

"address of" operator

(give me an arrow pointing to `i`)

Dereferencing

```
int i = 8;
int *p = &i;
```



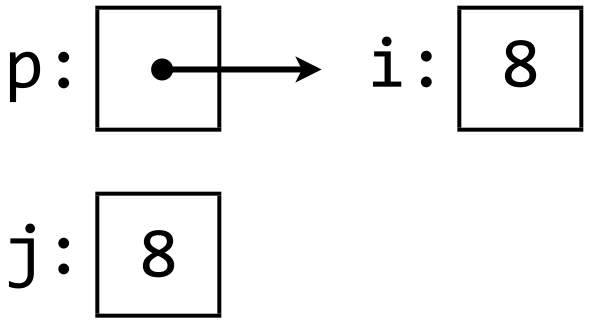

```
int j = p;
```

Wrong

Dereferencing

```
int i = 8;
int *p = &i;
```

p is of type `int *`




```
int j = p;
```

Wrong

```
int a[] = {1,2};
int k = a[1];
```

```
int j = *p;
```

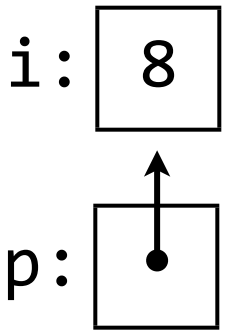
"indirection operator"
(follow the arrow in p and get the value stored there)

These are two different meanings of * !

```
int i = 2;
int *p;

p = &i;
(int *)   &(int)
          (int *)

*p = 4;
*(int *)    (int)
 (int)

*p = &i;
*(int *)   &(int)
 (int)     (int *)
```

Wrong

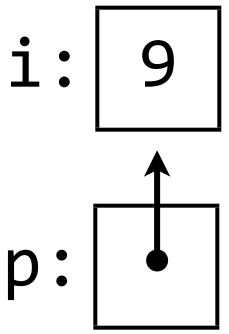
Aliasing




```
int i = 8;
int *p = &i;

printf("i: %d, *p: %d \n", i, *p);
```

```
i: 8, *p: 8
```

Aliasing



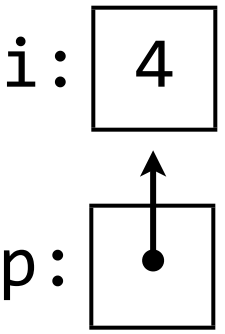

```
int i = 8;
int *p = &i;

printf("i: %d, *p: %d \n", i, *p);
```

```
i: 8, *p: 8
```

```
i = 9;
printf("i: %d, *p: %d \n", i, *p);
```

```
i: 9, *p: 9
```

Aliasing




```
int i = 8;
int *p = &i;

printf("i: %d, *p: %d \n", i, *p);
```

```
i: 8, *p: 8
```

```
i = 9;
printf("i: %d, *p: %d \n", i, *p);
```

```
i: 9, *p: 9
```

```
*p = 4;
printf("i: %d, *p: %d \n", i, *p);
```

```
i: 4, *p: 4
```

Aliasing

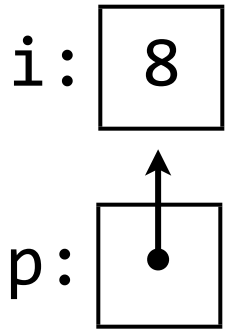
- When there are two (or more) ways to refer to the same chunk of data

```
int i = 8;
int *p = &i;
```

`i` and `*p` are aliases of each other

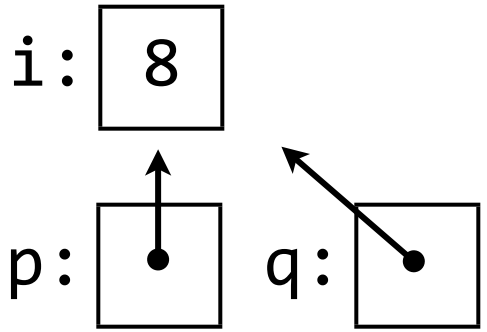
Aliasing

```
int i = 8;
int *p = &i;
```



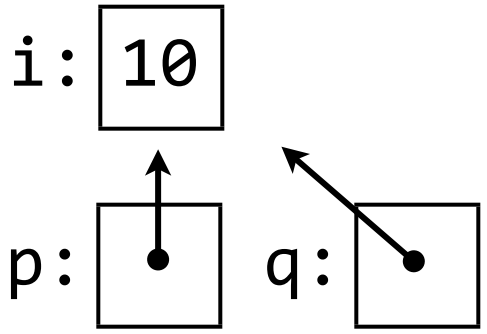
Aliasing

```
int i = 8;
int *p = &i;

int *q = &i;
```



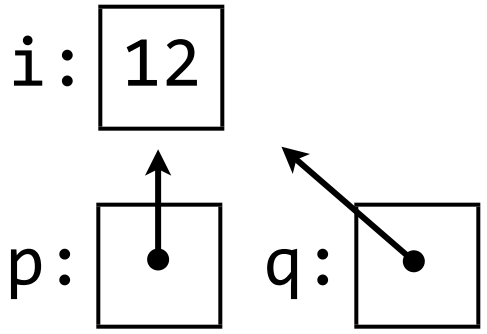
Aliasing

```
int i = 8;
int *p = &i;

int *q = &i;

i = 10;
```



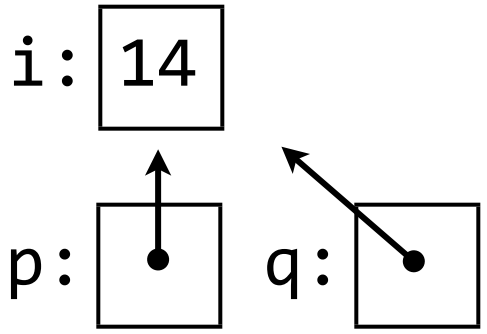
Aliasing

```
int i = 8;
int *p = &i;

int *q = &i;

i = 10;
*p = 12;
```



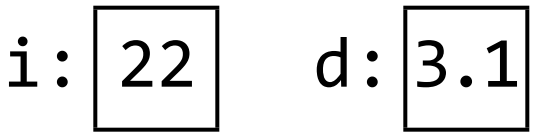
Aliasing

```
int i = 8;
int *p = &i;

int *q = &i;

i = 10;
*p = 12;
*q = 14;
```



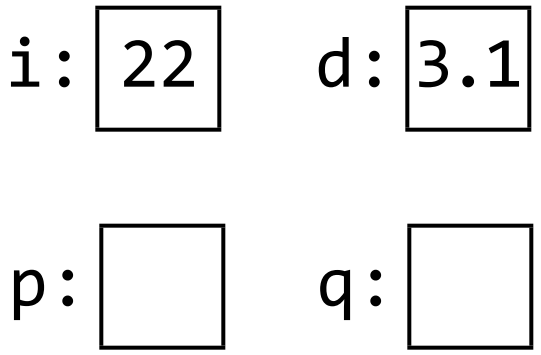
Pointer Types

```
int i = 22;
double d = 3.1;
```



Pointer Types

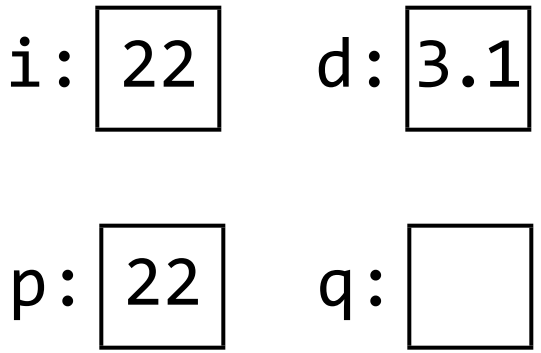
```
int i = 22;
double d = 3.1;

int *p;
double *q;
```



Pointer Types

```
int i = 22;
double d = 3.1;

int *p;
double *q;

p = i;
```

Wrong

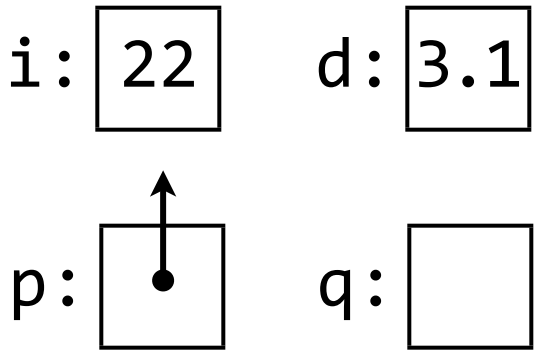


Pointer Types

```
int i = 22;
double d = 3.1;

int *p;
double *q;

p = i;
```

Wrong

```
p = &i;
```

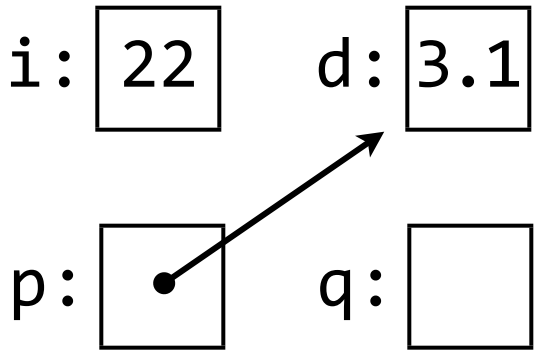


Pointer Types

```
int i = 22;
double d = 3.1;

int *p;
double *q;

p = i;
p = &i;
p = &d;
```

Wrong

Wrong

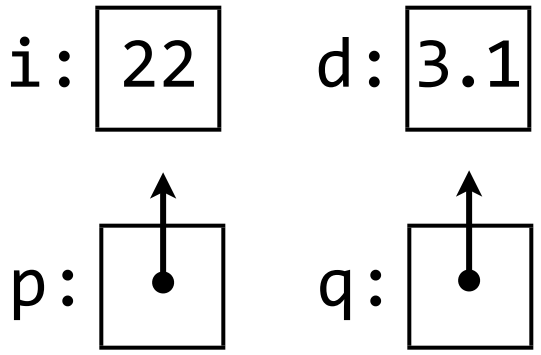


Pointer Types

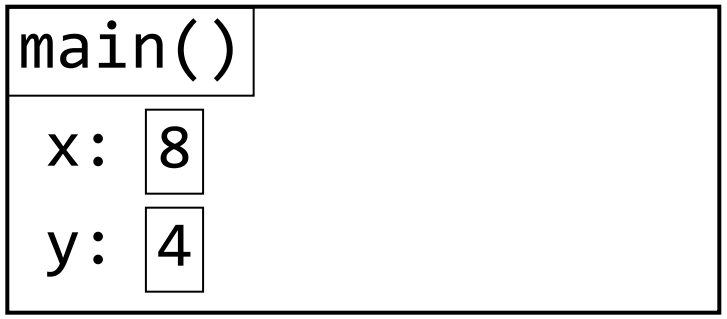
```
int i = 22;
double d = 3.1;

int *p;
double *q;

p = i;
p = &i;
p = &d;
q = &d;
```

Wrong

Wrong

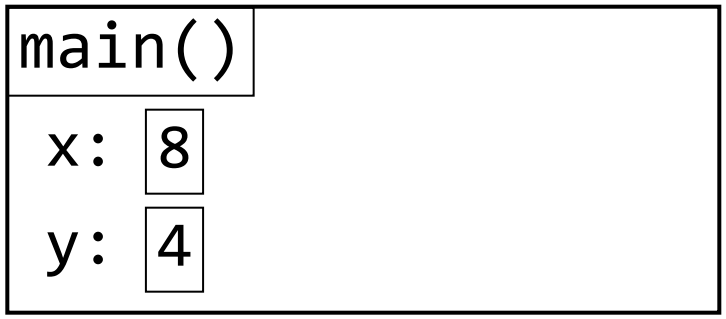


```
void swap(int a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(x, y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```



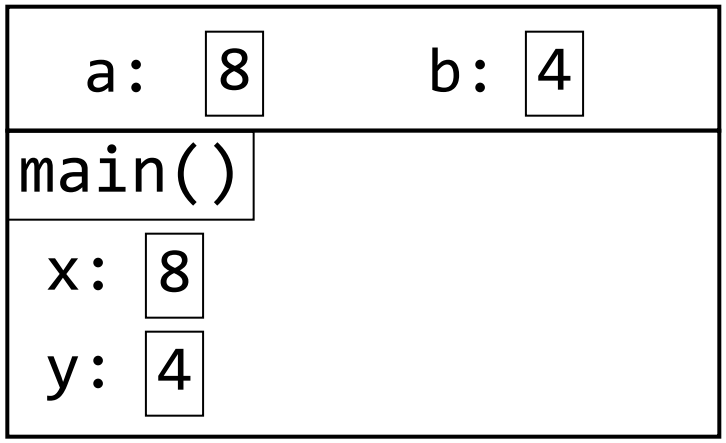
```
void swap(int a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(x, y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```

```
x: 8, y: 4
```



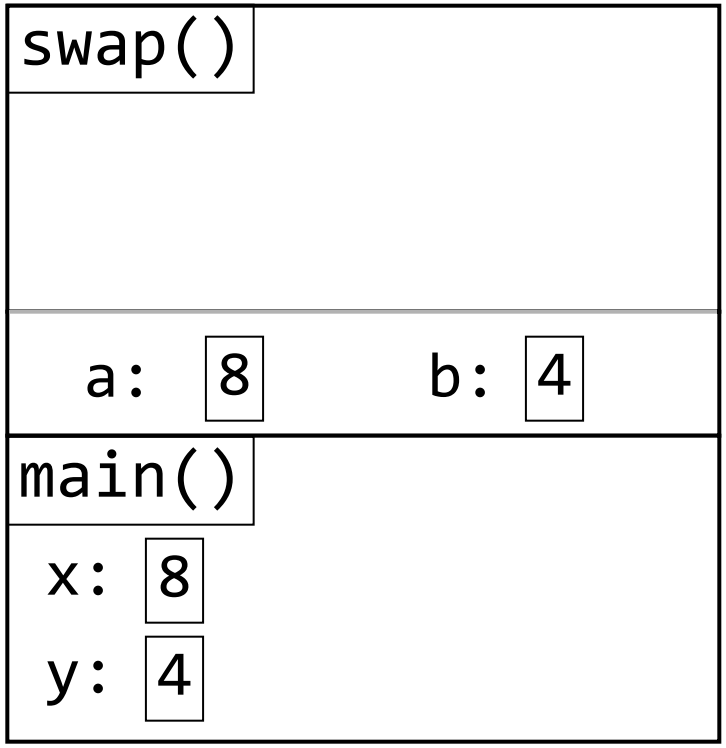
```
void swap(int a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(x, y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```

```
x: 8, y: 4
```



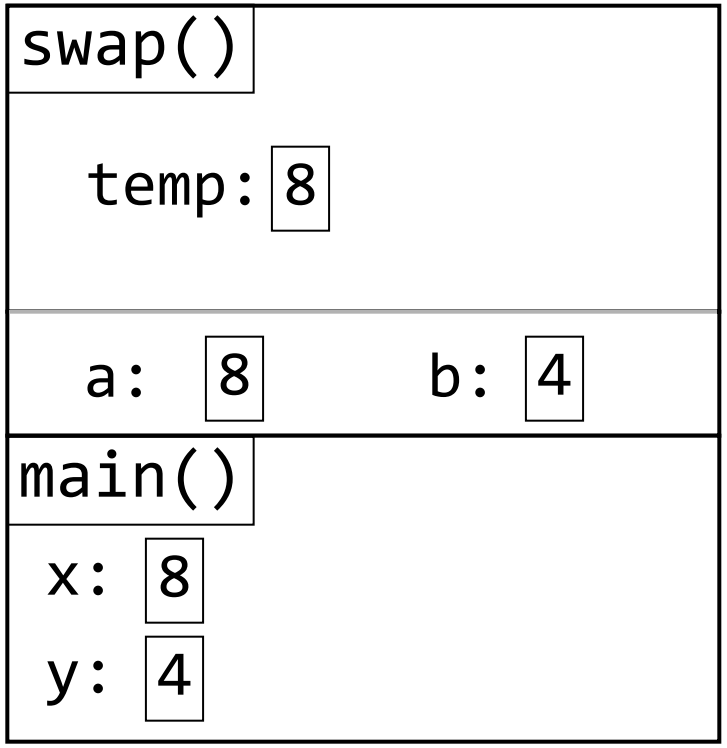
```
void swap(int a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(x, y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```

```
x: 8, y: 4
```



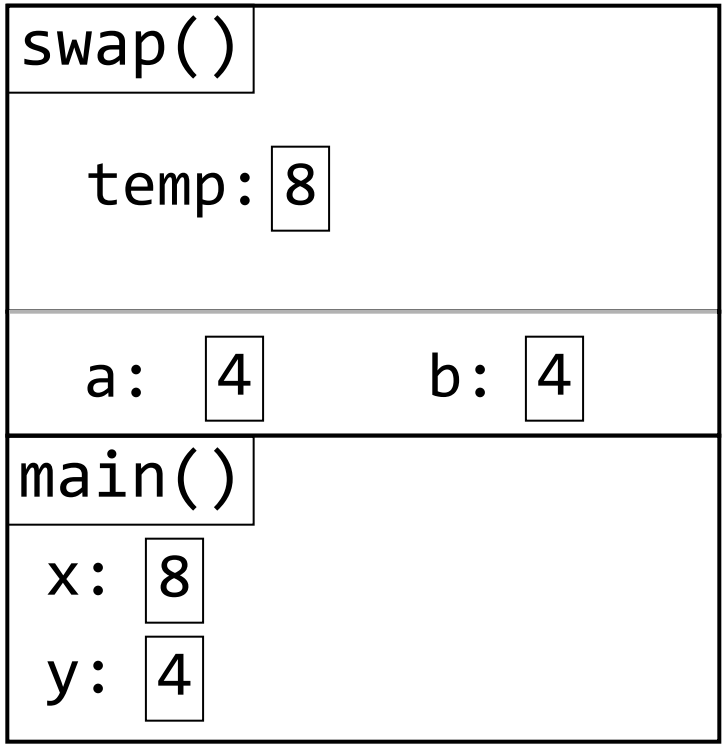
```
void swap(int a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(x, y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```

```
x: 8, y: 4
```



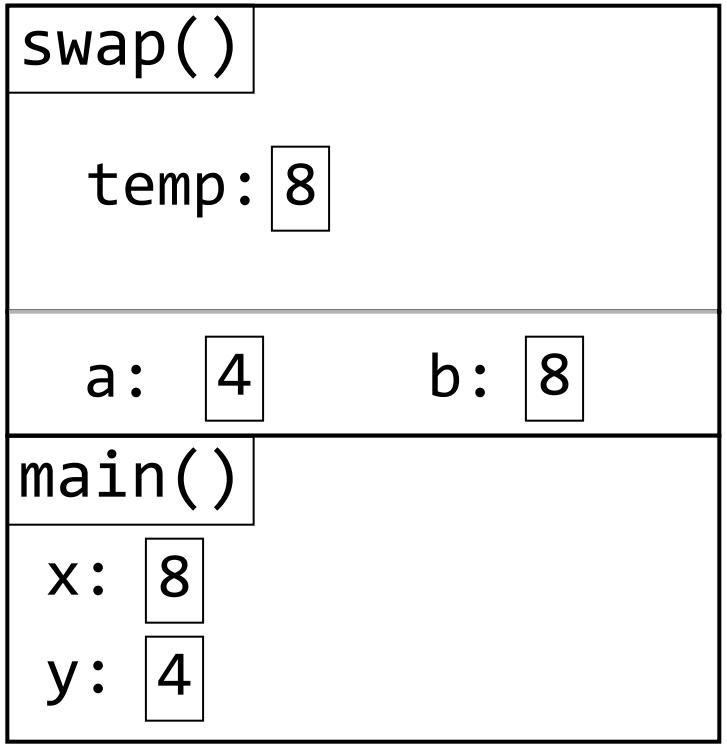
```
void swap(int a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(x, y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```

```
x: 8, y: 4
```



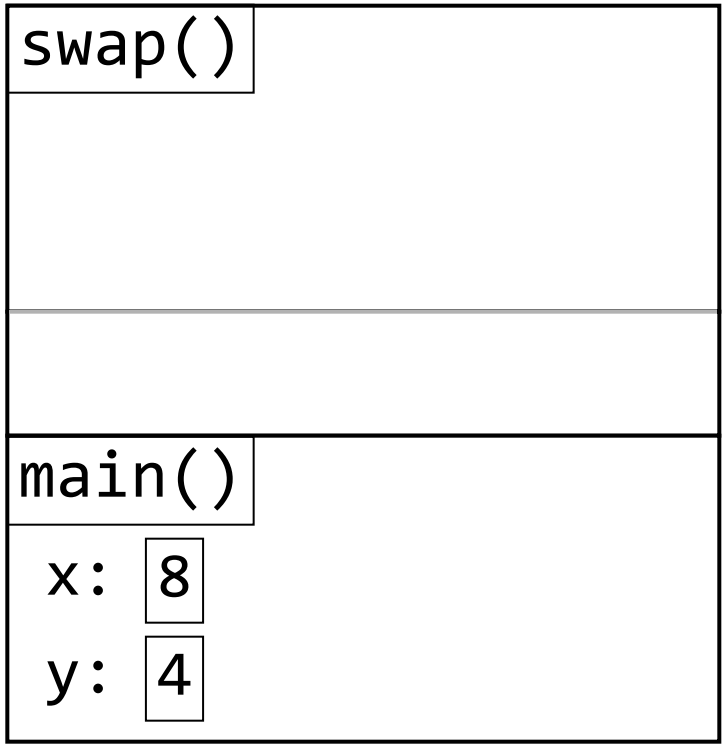
```
void swap(int a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(x, y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```

```
x: 8, y: 4
```



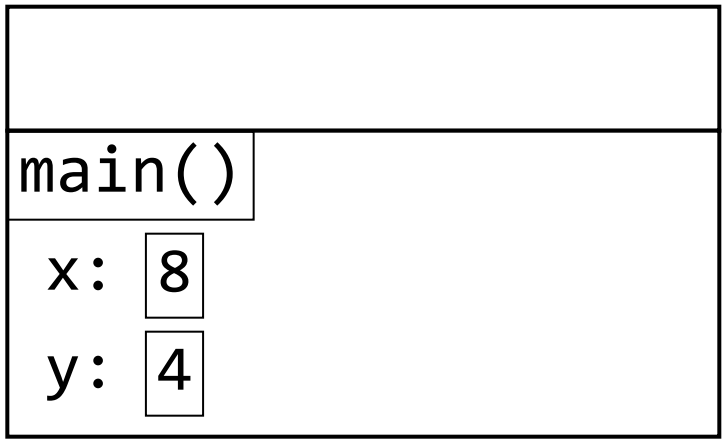
```
void swap(int a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(x, y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```

```
x: 8, y: 4
```



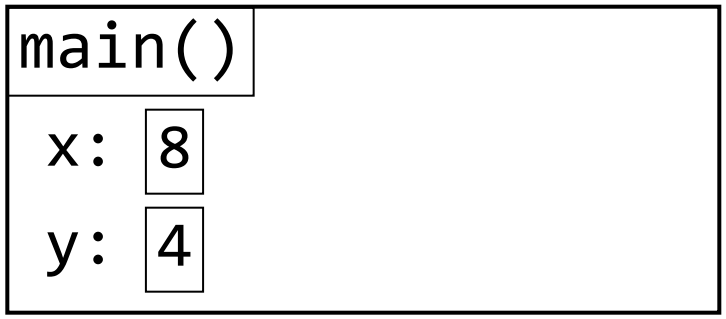
```
void swap(int a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(x, y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```

```
x: 8, y: 4
```



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

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(x, y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```

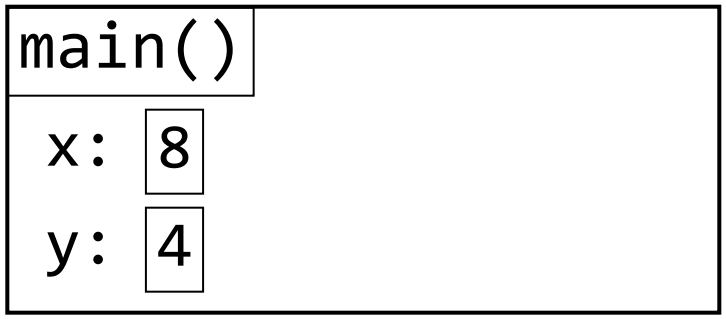
```
x: 8, y: 4
```



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

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(x, y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```

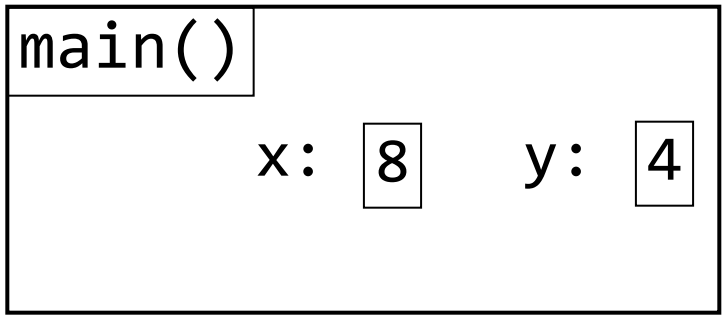
```
x: 8, y: 4
```

```
x: 8, y: 4
```



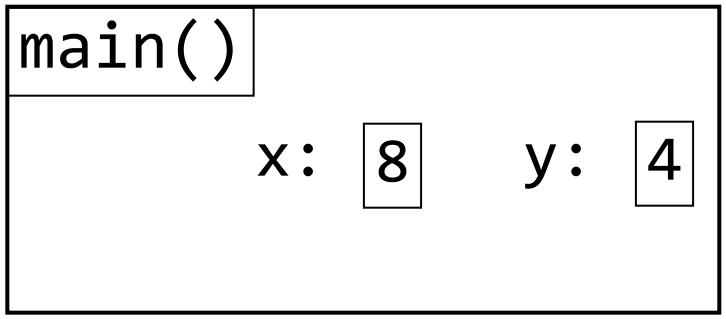
```
void swap(int *a, int *b)
{
  int temp = *a;
  *a = *b;
  *b = temp;
}

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(&x, &y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```

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

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(&x, &y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```



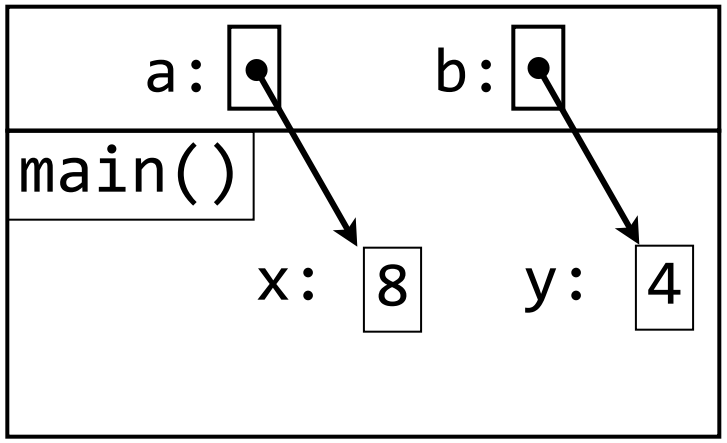
```
void swap(int *a, int *b)
{
  int temp = *a;
  *a = *b;
  *b = temp;
}

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(&x, &y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```

```
x: 8, y: 4
```



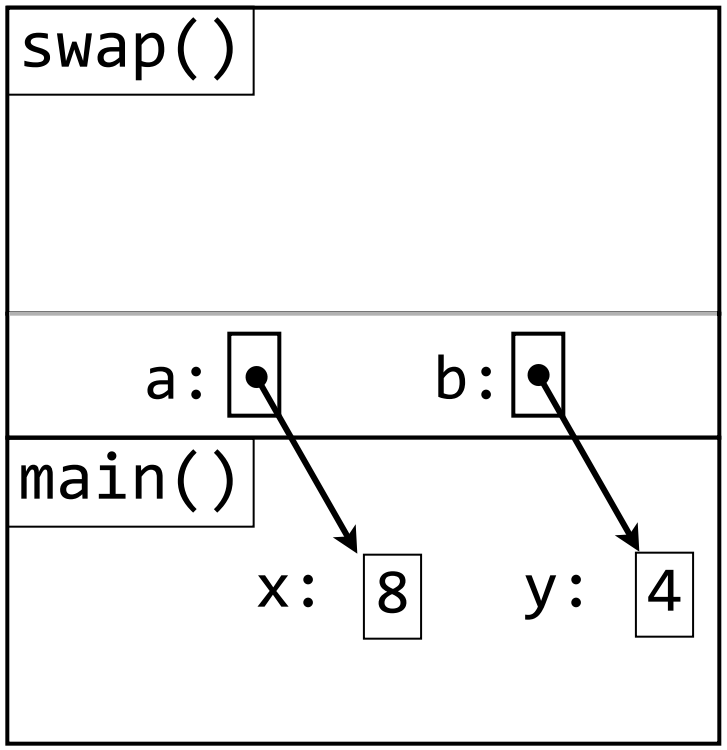
```
void swap(int *a, int *b)
{
  int temp = *a;
  *a = *b;
  *b = temp;
}

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(&x, &y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```

```
x: 8, y: 4
```



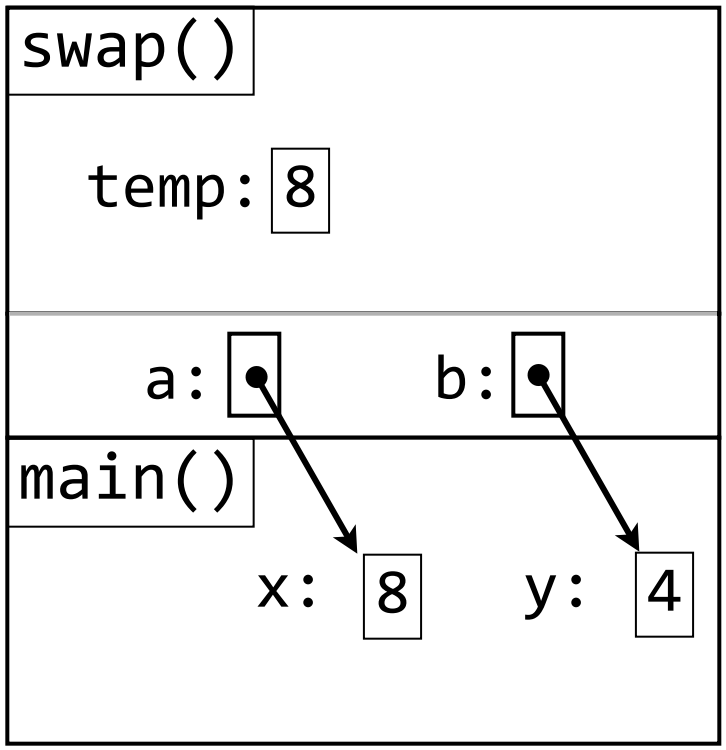
```
void swap(int *a, int *b)
{
  int temp = *a;
  *a = *b;
  *b = temp;
}

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(&x, &y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```

```
x: 8, y: 4
```



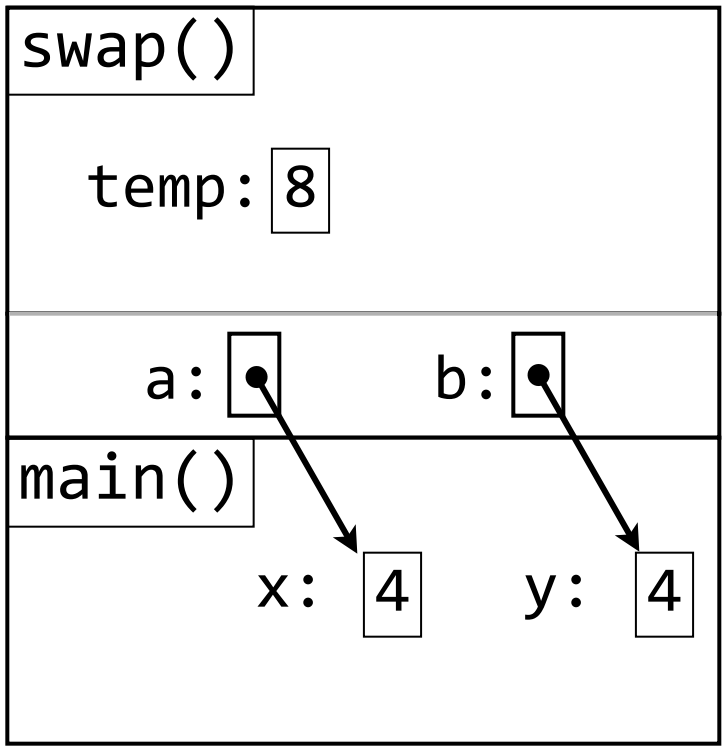
```
void swap(int *a, int *b)
{
  int temp = *a;
  *a = *b;
  *b = temp;
}

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(&x, &y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```

```
x: 8, y: 4
```



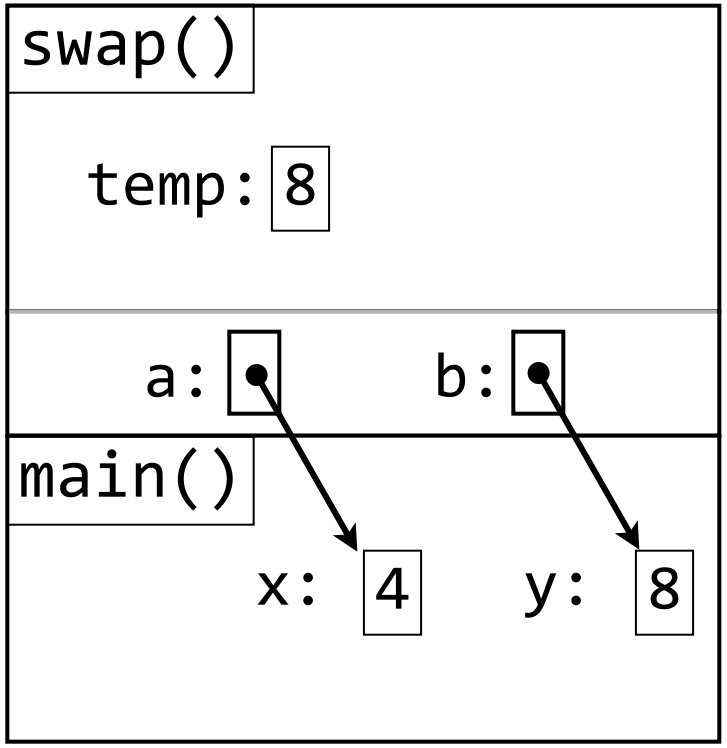
```
void swap(int *a, int *b)
{
  int temp = *a;
  *a = *b;
  *b = temp;
}

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(&x, &y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```

```
x: 8, y: 4
```



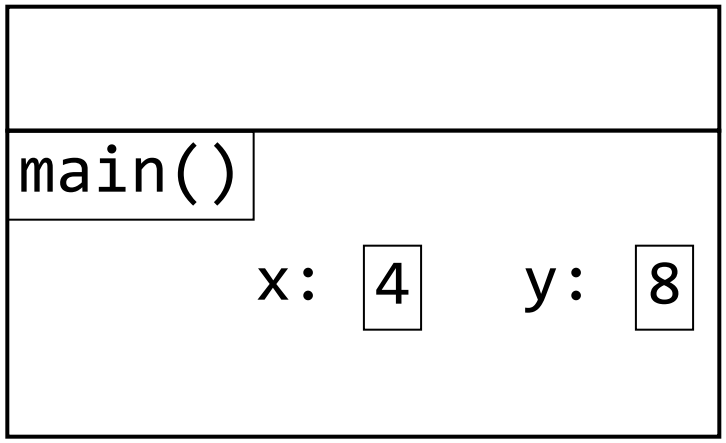
```
void swap(int *a, int *b)
{
  int temp = *a;
  *a = *b;
  *b = temp;
}

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(&x, &y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```

x: 8, y: 4



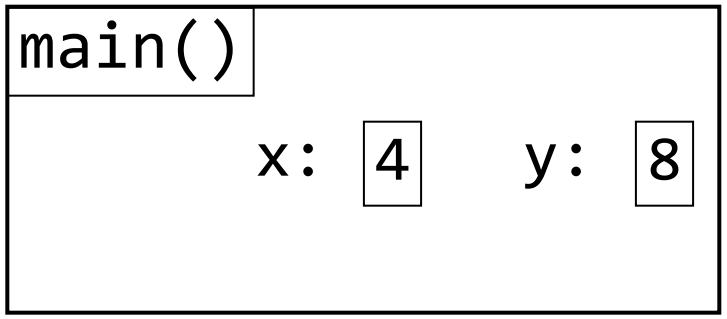
```
void swap(int *a, int *b)
{
  int temp = *a;
  *a = *b;
  *b = temp;
}

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(&x, &y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```

```
x: 8, y: 4
```



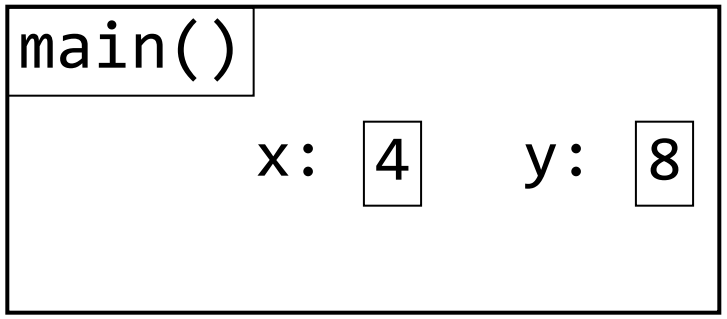
```
void swap(int *a, int *b)
{
  int temp = *a;
  *a = *b;
  *b = temp;
}

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(&x, &y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```

```
x: 8, y: 4
```



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

int main(void)
{
  int x = 8;
  int y = 4;

  printf("x: %d, y: %d \n",x,y);

  swap(&x, &y);

  printf("x: %d, y: %d \n",x,y);

  return 0;
}
```

```
x: 8, y: 4
x: 4, y: 8
```



Passing Parameters

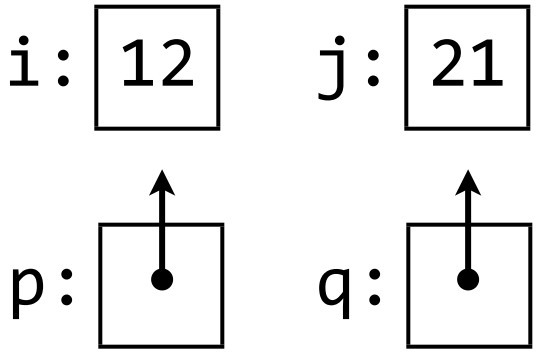
- Pass-by-value `void swap (int a, int b);`
  - Make a copy of what's in a variable

- Pass-by-reference `void swap (int *a, int *b);`
  - Uses indirection
  - Provide a way to find another variable
  - Still a copy, but copy of a different thing

Copying Pointers

- You can use =, as long as the types match

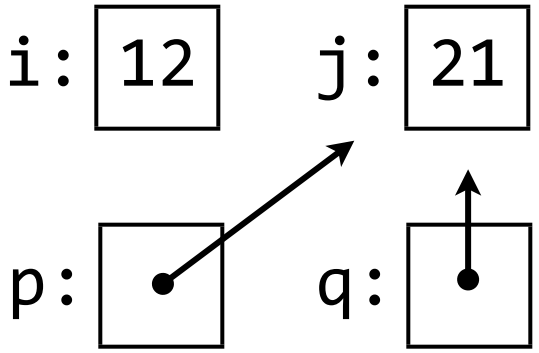
```
int i = 12;
int j = 21;

int *p = &i;
int *q = &j;
```



Copying Pointers

- You can use =, as long as the types match

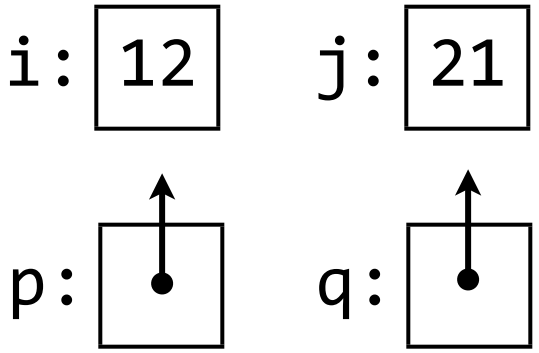
```
int i = 12;
int j = 21;

int *p = &i;
int *q = &j;

p = q;
```



Copying Pointers

- You can use =, as long as the types match

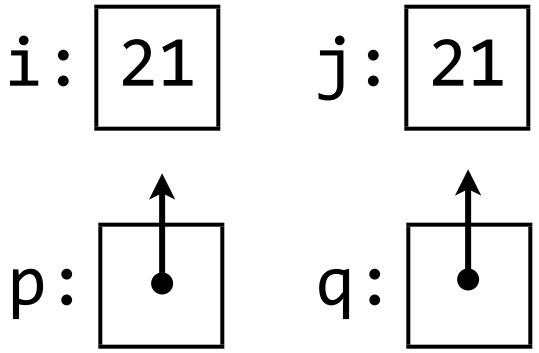
```
int i = 12;
int j = 21;

int *p = &i;
int *q = &j;

p = q;
```



Copying Pointers

- You can use =, as long as the types match

```
int i = 12;
int j = 21;

int *p = &i;
int *q = &j;

p = q;
*p = *q;
```

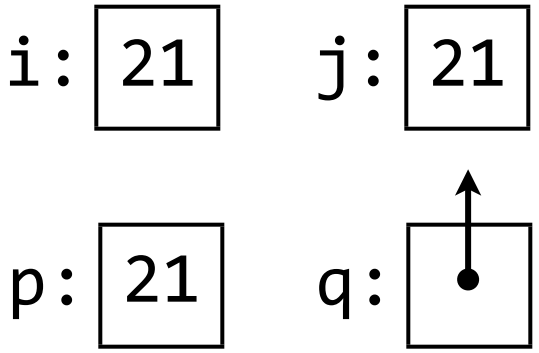


Copying Pointers

- You can use =, as long as the types match

```
int i = 12;
int j = 21;

int *p = &i;
int *q = &j;

p = q;
*p = *q;
 p = *q;
```

Wrong



Watch Your Precedence

```
int i = 9;
int *p = &i;
*p++;
```

```
*(p++);
```

```
(*p)++;
```

When in doubt, use parentheses