APS105
Winter 2012

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

Lecture 13
February 13, 2012

Today

- Random Numbers
- Swap

Random Numbers

- They aren’t random!
- Pseudorandom
- Why? Remember what computers do...
- Difference *really* matters for some things, but not for us

rand()

- Provides you with pseudorandom numbers
- Range is between `0` and `RAND_MAX`
- Lives in `stdlib.h`
  - `#include <stdlib.h>` (no `-l` needed)

```
int rand(void);
```

```c
#include <stdio.h>
#include <stdlib.h>

#define N 10

int main(void)
{
   printf("RAND_MAX is: %d \n", RAND_MAX);
   printf("Here are %d random numbers: \n", N);

   for (int i = 0; i < N; i++)
   {
       int random = rand();
       printf("%d\n", random);
   }
}
```

Pseudorandom Numbers

- Start with a seed (an `int`)
- Generate a series of numbers
  - Look random
  - 100% predictable given the seed

Planting a Seed

- By default, `rand()` uses seed of 1
- You can change that with `srand()`

```
void srand(int seed);
```

Finding a Seed

- Often want different seed each time we run
- Convenient option is the current time

```
#include <time.h>

...

srand( time(NULL) );
```

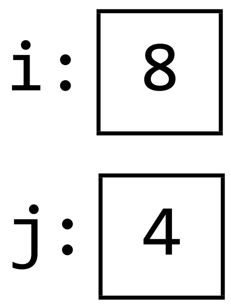
Smaller Range

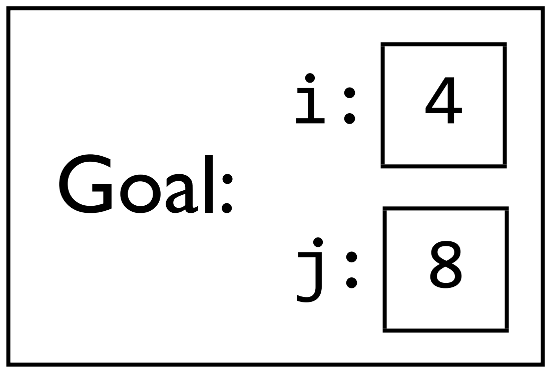
- Range is between `0` and `RAND_MAX`
- If you want a smaller range, just use some math

```
#define MAX 6
...

int random = rand() % MAX;     0 1 2 3 4 5

random = (rand() % MAX) + 1;  1 2 3 4 5 6

random = (rand() % MAX) * 2;  0 2 4 6 8 10
```

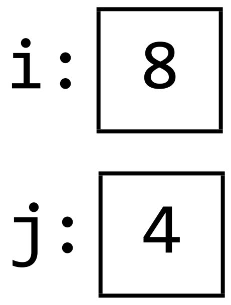
Swapping Values

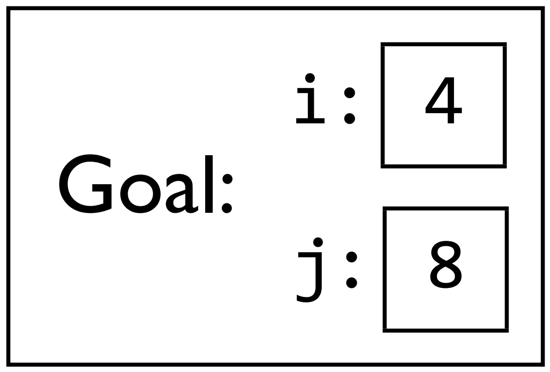
```
int i = 8;
int j = 4;
```





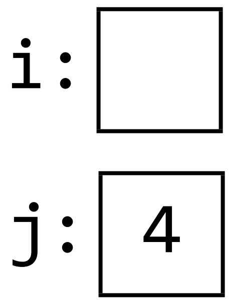
```
int i = 8;
int j = 4;
```

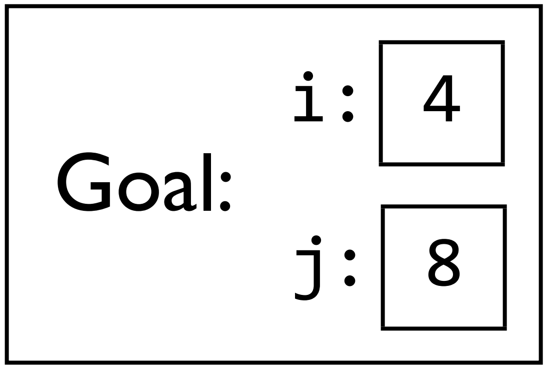




```
i = j;
j = i;
```

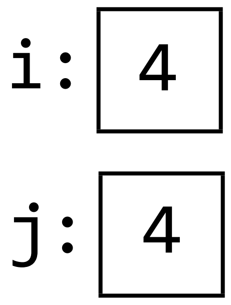
```
int i = 8;
int j = 4;
```

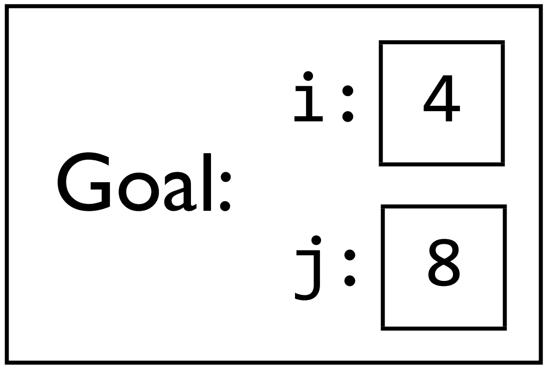




```
i = j;
j = i;
```

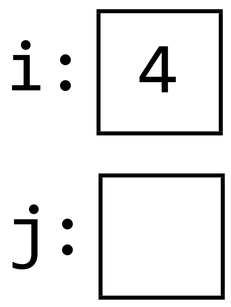
```
int i = 8;
int j = 4;
```

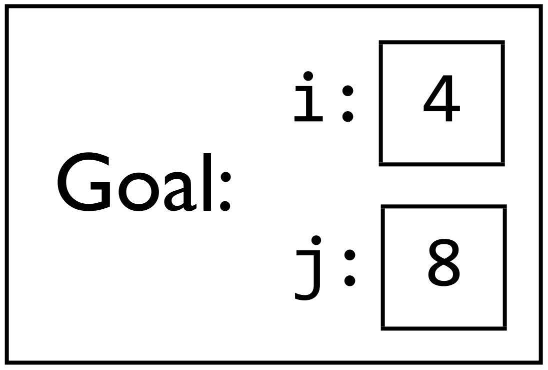




```
i = j;
j = i;
```

```
int i = 8;
int j = 4;
```





```
i = j;
j = i;
```

```
int i = 8;
int j = 4;
```

i: $\boxed{4}$

j: $\boxed{4}$

Goal: i: $\boxed{4}$ j: $\boxed{8}$

```
i = j;
j = i;
```

```
int i = 8;
int j = 4;
```

i: $\boxed{4}$

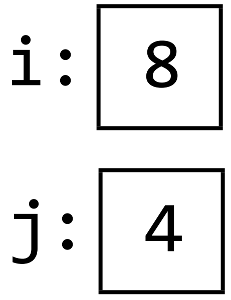
j: $\boxed{4}$

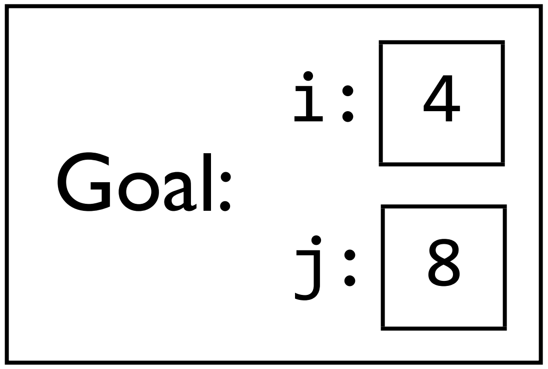
Goal: i: $\boxed{4}$ j: $\boxed{8}$

```
i = j;
j = i;
```

Wrong

```
int i = 8;
int j = 4;
```





```
~~i = j;
j = i;~~
```

Wrong

```
int temp = i;
i = j;
j = temp;
```

```
int i = 8;
int j = 4;
```

i: $\boxed{8}$

j: $\boxed{4}$

temp: $\boxed{8}$

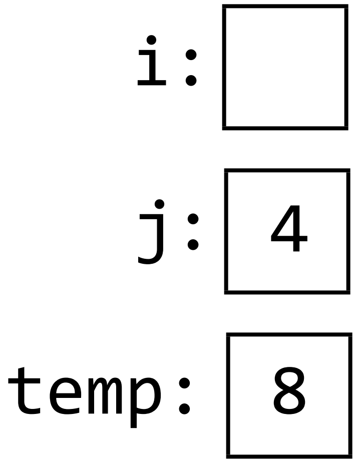
Goal: i: $\boxed{4}$ j: $\boxed{8}$

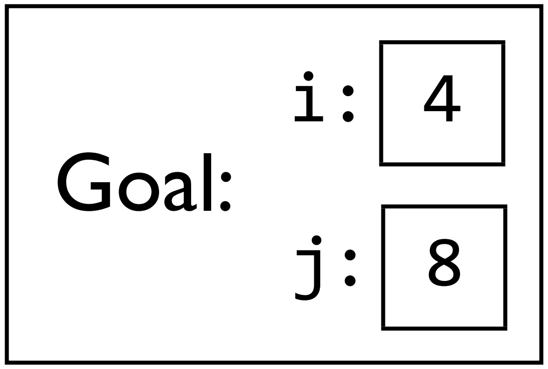
~~`i = j;`~~
~~`j = i;`~~

Wrong

```
int temp = i;
i = j;
j = temp;
```

```
int i = 8;
int j = 4;
```



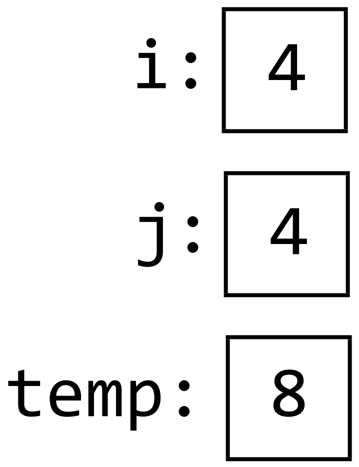


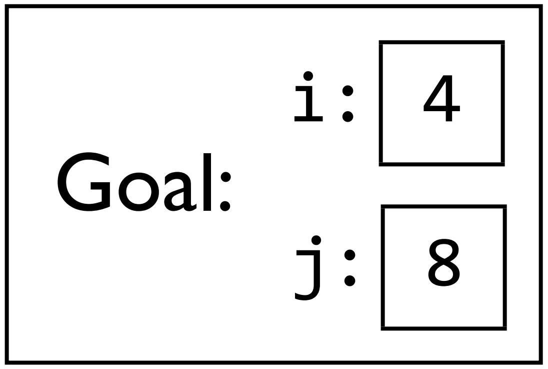
~~i = j;~~
~~j = i;~~

Wrong

```
int temp = i;
i = j;
j = temp;
```

```
int i = 8;
int j = 4;
```



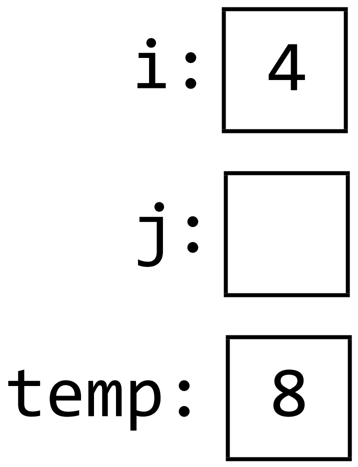


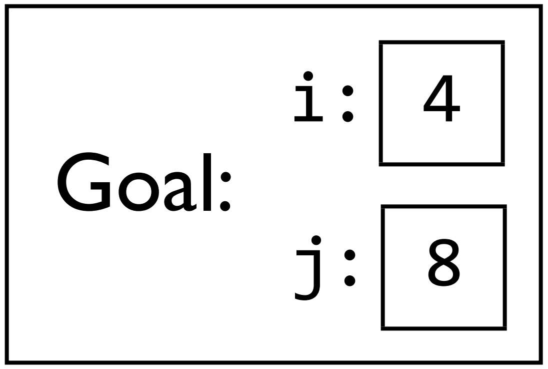
~~i = j;~~
~~j = i;~~

Wrong

```
int temp = i;
i = j;
j = temp;
```

```
int i = 8;
int j = 4;
```





~~i = j;~~
~~j = i;~~

Wrong

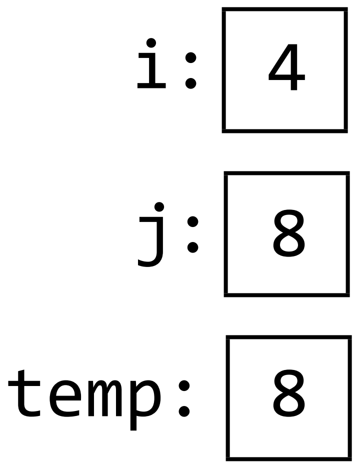
```
int temp = i;
i = j;
j = temp;
```

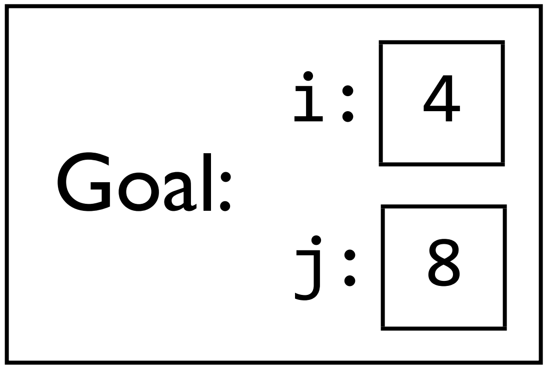
```
int i = 8;
int j = 4;
```

~~i = j;~~
~~j = i;~~

Wrong

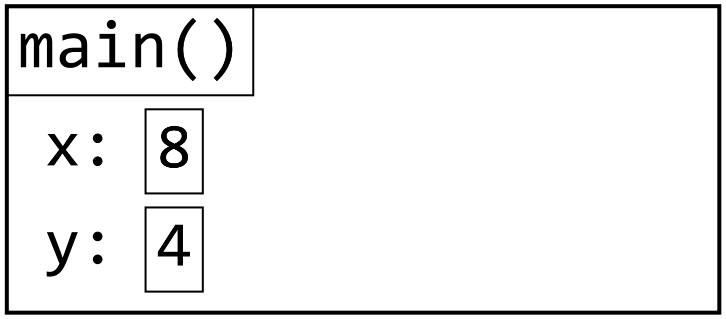
```
int temp = i;
i = j;
j = temp;
```





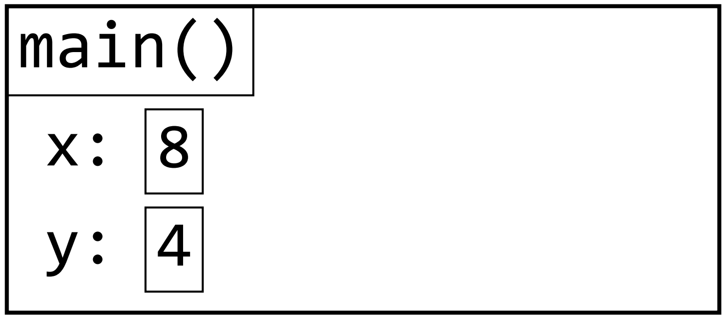
```
void swap(int a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}
```

```
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;
}
```



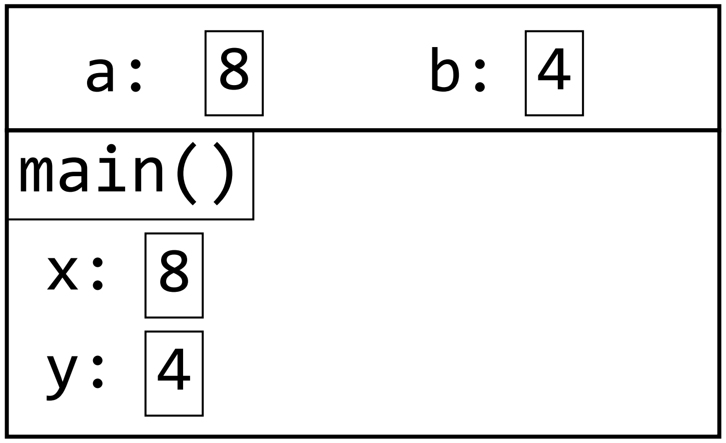
```c
void swap(int a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}

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
```



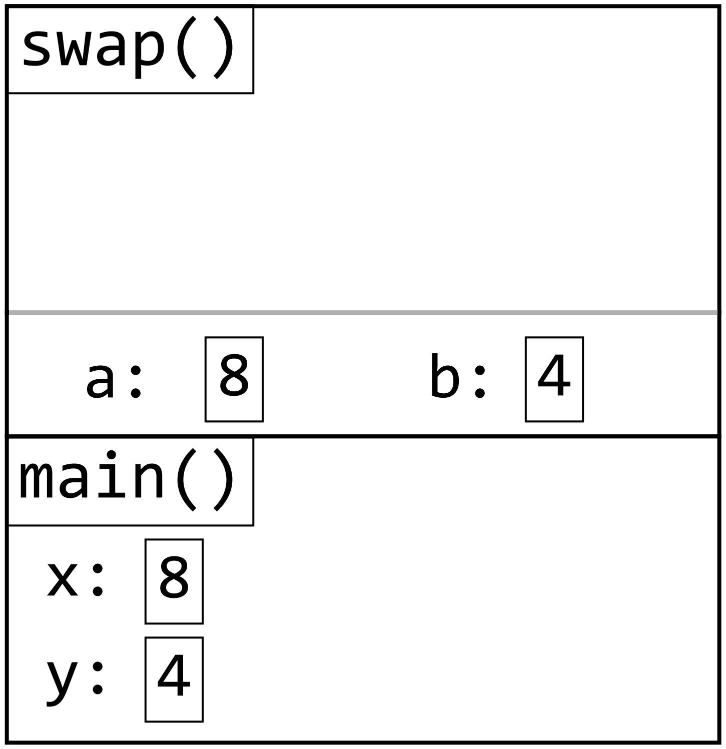
```
void swap(int a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}

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
```



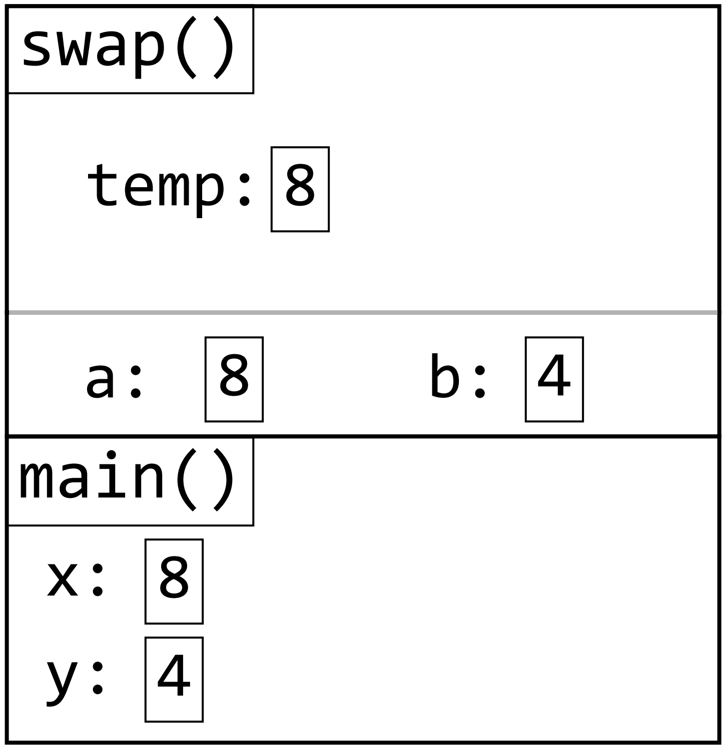
```
void swap(int a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}

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
```



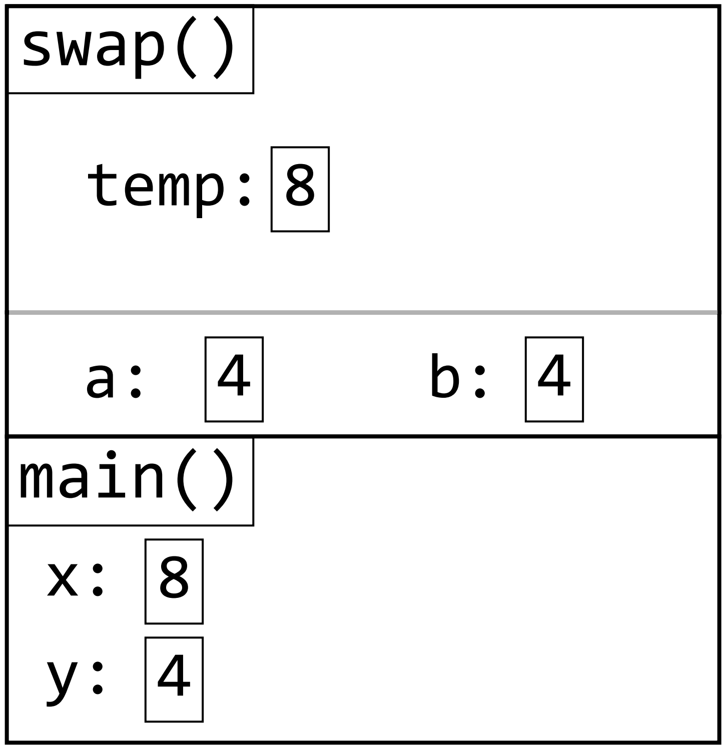
```
void swap(int a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}

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
```



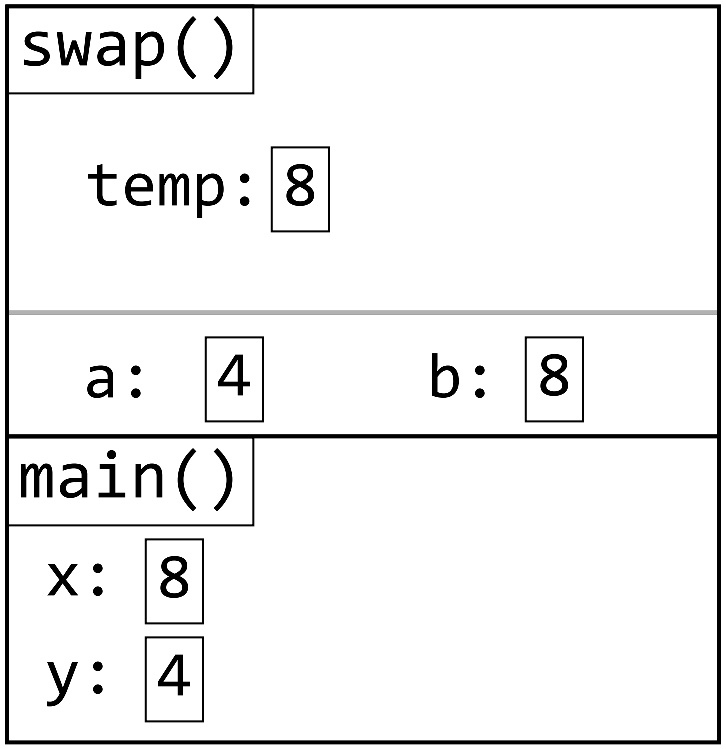
```
void swap(int a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}

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
```



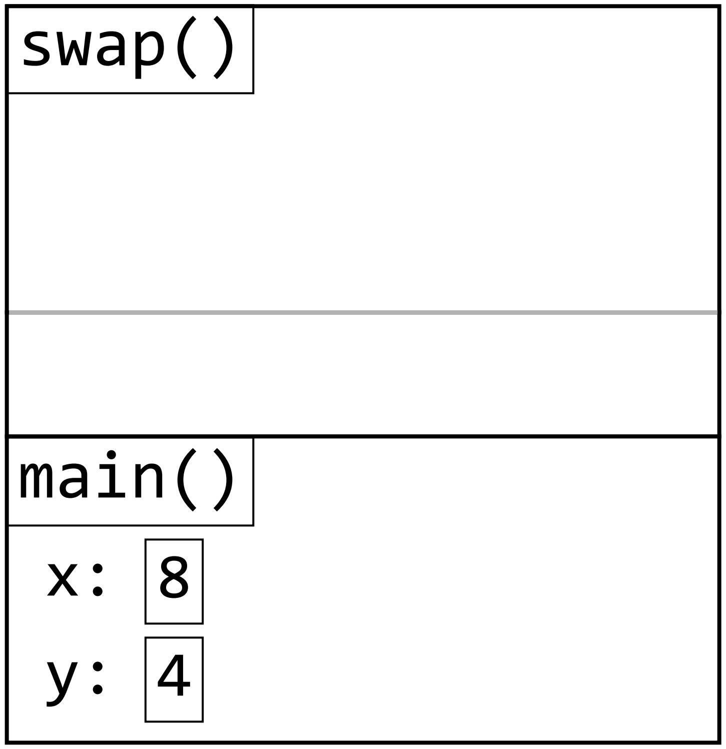
```
void swap(int a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}

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
```



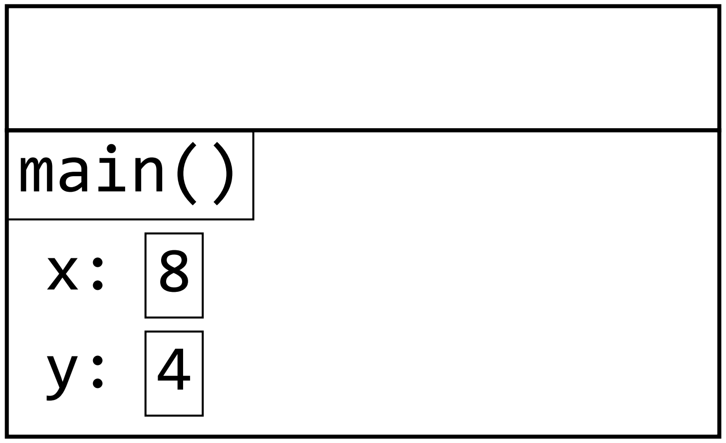
```
void swap(int a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}

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
```



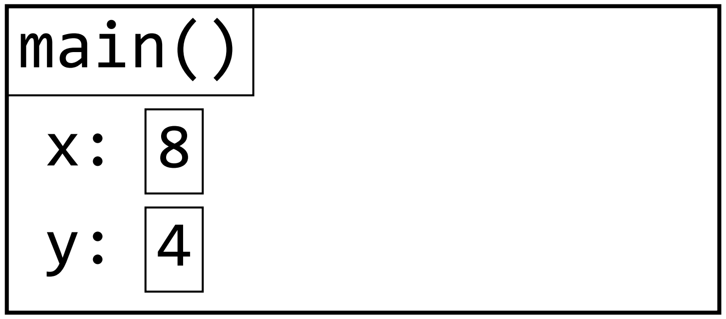
```
void swap(int a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}

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;
}
```

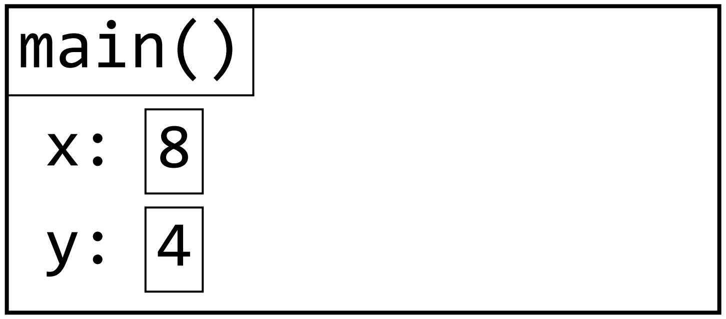
```
x: 8, y: 4
```



```
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
```



This is a bit weird, but it will make sense when we get to ...

Just put & there for now...

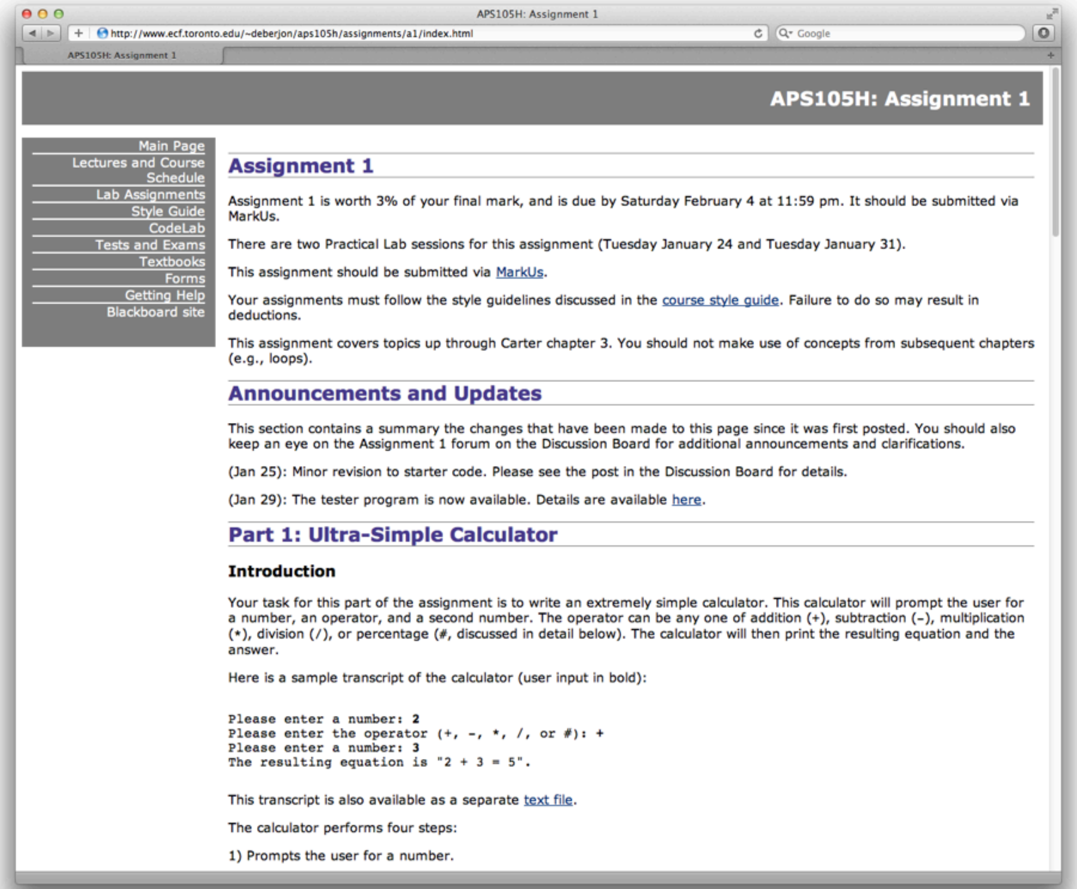
Pointers

We'll talk about that when we get to...

You can't return an array, but we'll see how to do that when we get to...

Indirection

~~Pointers~~



APS105H: Assignment 1

Assignment 1

Assignment 1 is worth 3% of your final mark, and is due by Saturday February 4 at 11:59 pm. It should be submitted via MarkUs.

There are two Practical Lab sessions for this assignment (Tuesday January 24 and Tuesday January 31).

This assignment should be submitted via MarkUs.

Your assignments must follow the style guidelines discussed in the course style guide. Failure to do so may result in deductions.

This assignment covers topics up through Carter chapter 3. You should not make use of concepts from subsequent chapters (e.g., loops).

Announcements and Updates

This section contains a summary the changes that have been made to this page since it was first posted. You should also keep an eye on the Assignment 1 forum on the Discussion Board for additional announcements and clarifications.

(Jan 25): Minor revision to starter code. Please see the post in the Discussion Board for details.

(Jan 29): The tester program is now available. Details are available here.

Part 1: Ultra-Simple Calculator

Introduction

Your task for this part of the assignment is to write an extremely simple calculator. This calculator will prompt the user for a number, an operator, and a second number. The operator can be any one of addition (+), subtraction (-), multiplication (*), division (/), or percentage (#, discussed in detail below). The calculator will then print the resulting equation and the answer.

Here is a sample transcript of the calculator (user input in bold):

```
Please enter a number: 2
Please enter the operator (+, -, *, /, or #): +
Please enter a number: 3
The resulting equation is "2 + 3 = 5".
```

This transcript is also available as a separate text file.

The calculator performs four steps:

1) Prompts the user for a number.

Save As... $\swarrow$ $\searrow$ Bookmark

Alice Bob

16 KB file called
index.html

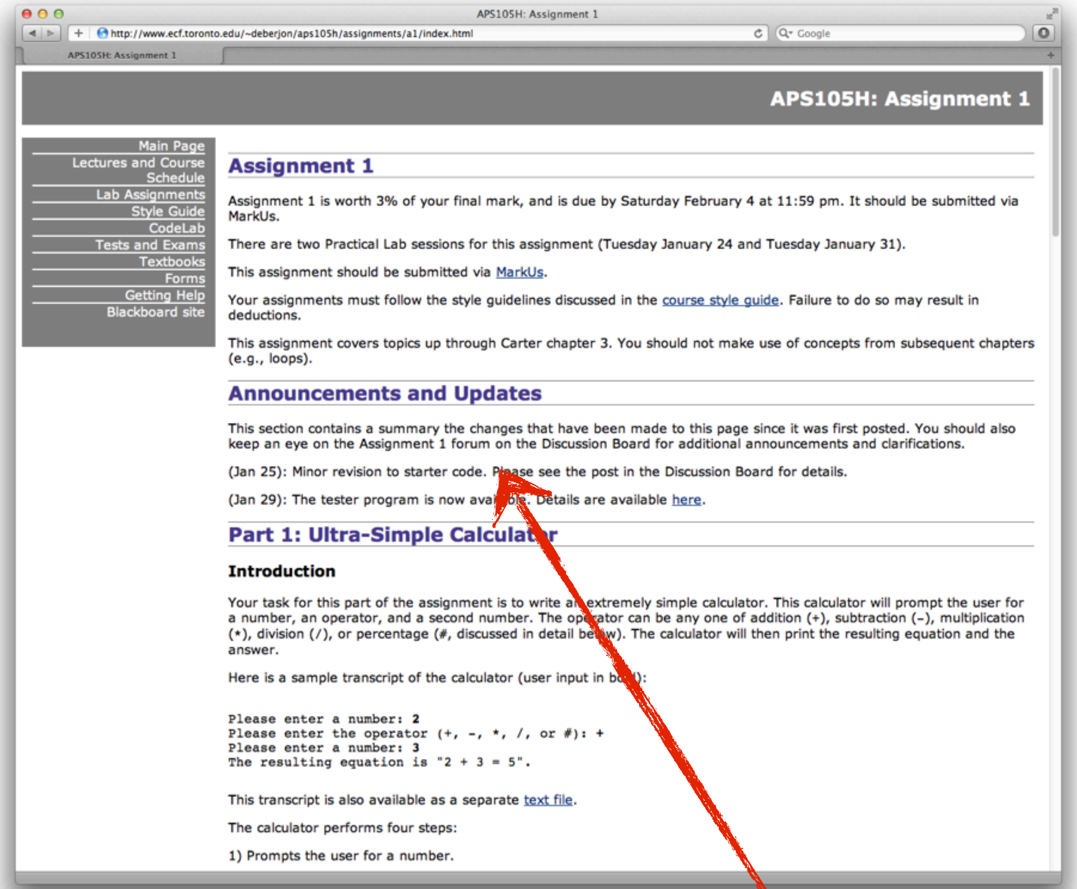
Title:	APS105: Assignment 1
URL:	`http://www.ecf.toronto.edu/~deberjon/aps105h/assignments/a1/index.html`

Entry in a bookmarks database

Alice

Bob

Icon: mozilla.org



APS105H: Assignment 1

http://www.ecf.toronto.edu/~deberjon/aps105h/assignments/a1/index.html

APS105H: Assignment 1

Main Page
Lectures and Course Schedule
Lab Assignments
Style Guide
CodeLab
Tests and Exams
Textbooks
Forms
Getting Help
Blackboard site

Assignment 1

Assignment 1 is worth 3% of your final mark, and is due by Saturday February 4 at 11:59 pm. It should be submitted via MarkUs.

There are two Practical Lab sessions for this assignment (Tuesday January 24 and Tuesday January 31).

This assignment should be submitted via MarkUs.

Your assignments must follow the style guidelines discussed in the course style guide. Failure to do so may result in deductions.

This assignment covers topics up through Carter chapter 3. You should not make use of concepts from subsequent chapters (e.g., loops).

Announcements and Updates

This section contains a summary the changes that have been made to this page since it was first posted. You should also keep an eye on the Assignment 1 forum on the Discussion Board for additional announcements and clarifications.

(Jan 25): Minor revision to starter code. Please see the post in the Discussion Board for details.

(Jan 29): The tester program is now available. Details are available here.

Part 1: Ultra-Simple Calculator

Introduction

Your task for this part of the assignment is to write an extremely simple calculator. This calculator will prompt the user for a number, an operator, and a second number. The operator can be any one of addition (+), subtraction (-), multiplication (*), division (/), or percentage (#, discussed in detail below). The calculator will then print the resulting equation and the answer.

Here is a sample transcript of the calculator (user input in bold):

```
Please enter a number: 2
Please enter the operator (+, -, *, /, or #): +
Please enter a number: 3
The resulting equation is "2 + 3 = 5".
```

This transcript is also available as a separate text file.

The calculator performs four steps:

1) Prompts the user for a number.

Updated starter code available

Title:	APS105: Assignment 1
URL:	`http://www.ecf.toronto.edu/~deberjon/aps105h/assignments/a1/index.html`

16 KB file called index.html

Entry in a bookmarks database

Alice

Bob

Icon: mozilla.org

Indirection

- Something that tells you where to find something else

All problems in computer science can be solved by another level of indirection.

Except for the problem of too many layers of indirection.

David Wheeler

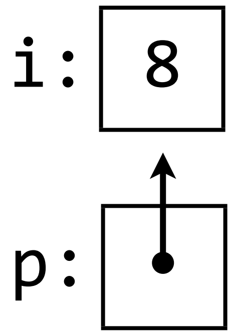
Indirection

- Something that tells you where to find something else

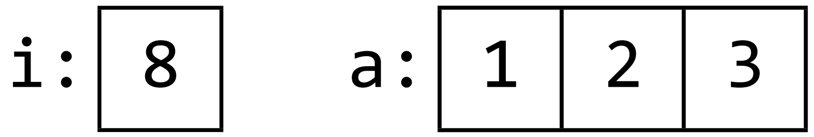
Something	*Something that tells you where to find it*
Contents of webpage	URL
A house	Street address
A person	Phone number
Variable	Pointer

Pointers

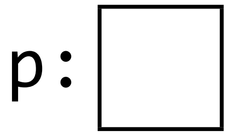
- A pointer is a variable that holds a memory address
- It "points at" another variable



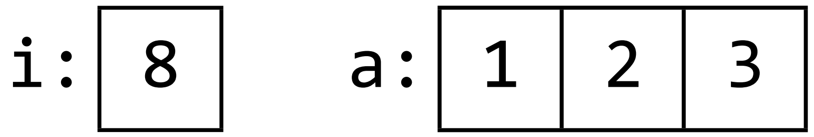
```
int i = 8;
int a[] = {1, 2, 3};
```



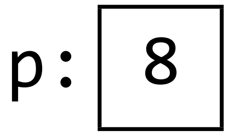
```
int *p;
```



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



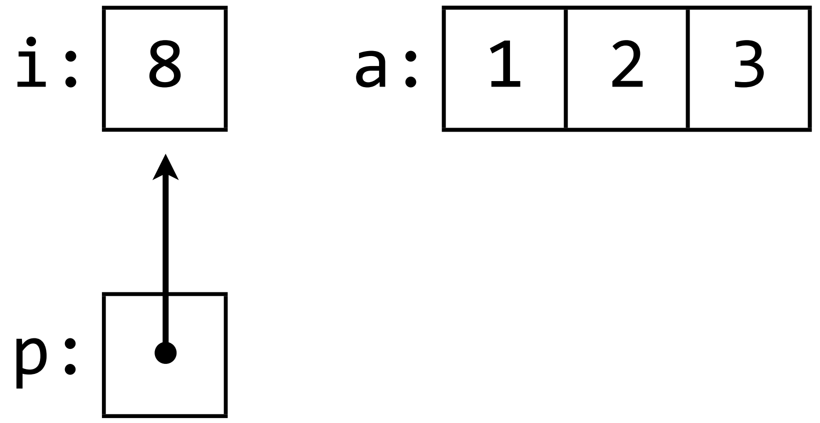
```
int *p;
```



```
p = i;
```

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!

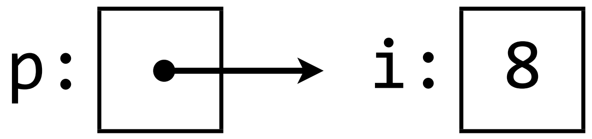
```
p = &i;
```

"address of" operator

(give me an arrow pointing to `i`)

Dereferencing

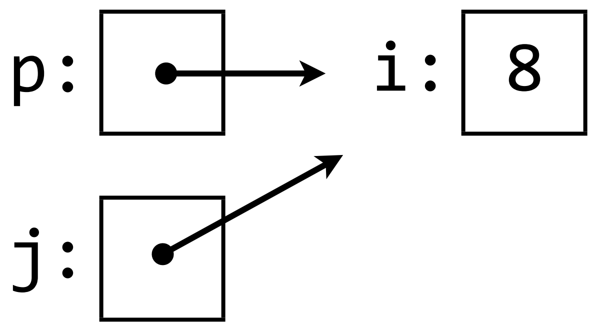
```
int i = 8;
int *p = &i;
```



Dereferencing

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


int j = p;
```

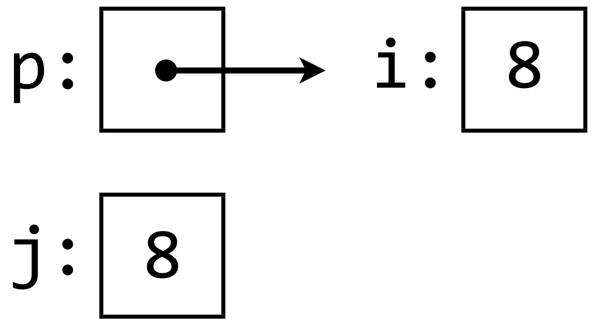
Wrong



Dereferencing

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

p is of type `int *`



```
int j = p;
```

Wrong

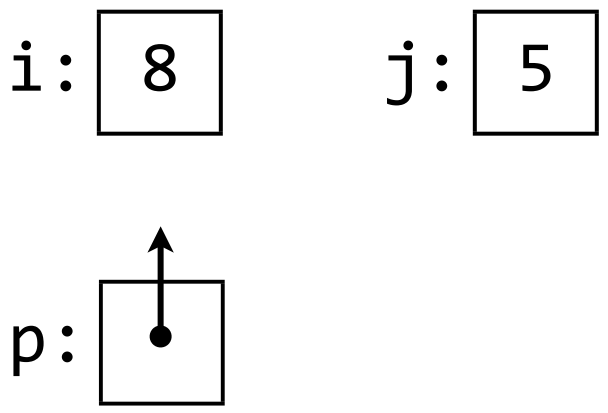
```
int a[] = {1,2};
int k = a[1];
```

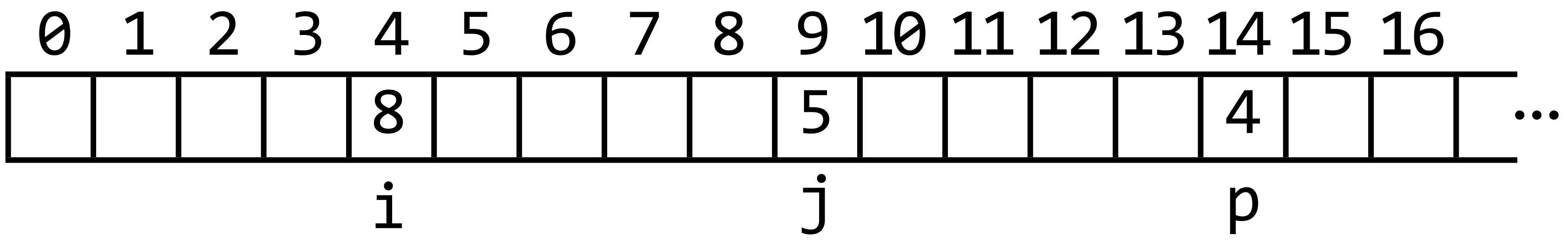
```
int j = *p;
```

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

These are two different meanings of * !

```
int i = 8;
int j = 5;

int *p = &i;
```





Conversion Specifier

- You can print a pointer with %p
- Generally not that useful

- Addresses look like this: `0x7fff649a09c4`