Introduction to Software Design

Pointers

For now, please avoid using the following two C syntactic constructs:

- & (address of)
- `void *`

Dereferencing A Pointer


```
1 int *ip = malloc(sizeof(int));
2
3 // To set the value pointed to by ip to 10, do either of the following
4 ip[0] = 10;
5 *ip = 10;
6
7 free(ip);
```

Pixel in Java

```
1 public class Pixel {
2    public int red;
3    public int green;
4    public int blue;
5
6    public void zeroRed(Pixel p) {
7       p.red = 0;
8    }
9 }
```

pixel_t in C

```
1  typedef struct pixel {
2     uint8_t red;
3     uint8_t green;
4     uint8_t blue;
5  } pixel_t;
6
7
8  void pixel_zero_red(pixel_t *p) {
9     p->red = 0;
10 }
```

pixel_t in C

```
1  typedef struct pixel {
2     uint8_t red;
3     uint8_t green;
4     uint8_t blue;
5  } pixel_t;
6
7
8  void pixel_zero_red(pixel_t *p) {
9     p->red = 0;
10 }
```

So, what does this do?

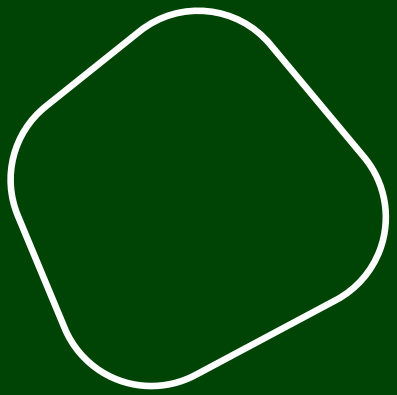
```
1  void pixel_zero_without_star(pixel_t p) {
2     p.red = 0;
3  }
```

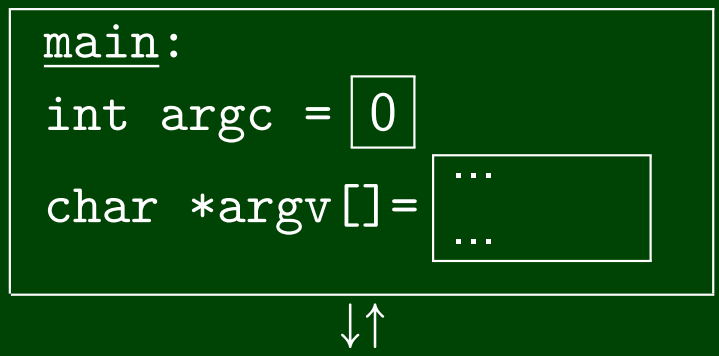
Investigating By Analogy to `int`

```
1  void f(int i) {
2     i = 5;
3  }
```

```
1  void g(int *pi) {
2     *pi = 5;
3  }
```

```
1  int main(int argc, char *argv[]) {
2     int d = 0;
3     f(d);
4     printf("%d\n", d);
5
6     int *p = malloc(sizeof(int));
7     *p = 0;
8     g(p);
9     printf("%d\n", *p);
10    free(p);
11 }
```



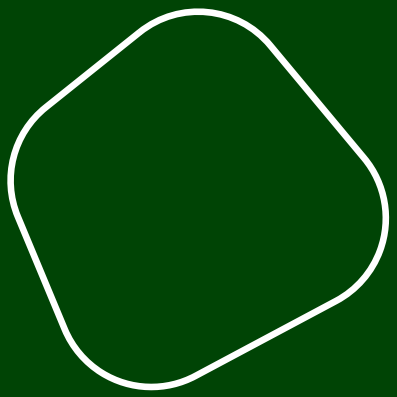


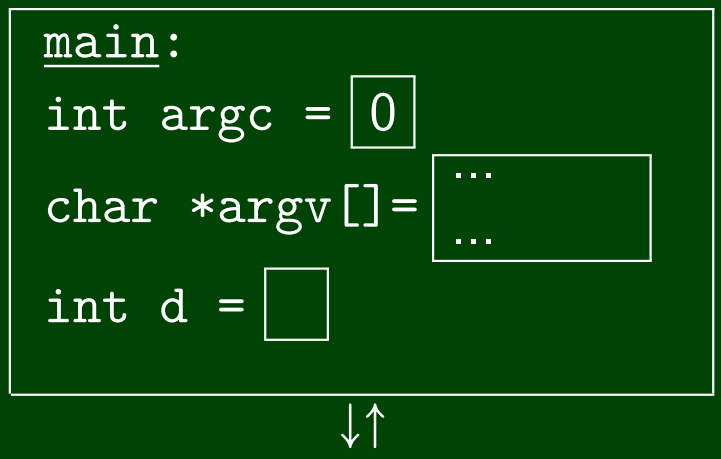
```c
void f(int i) {
    i = 5;
}
```

```c
void g(int *pi) {
    *pi = 5;
}
```

```c
int main(int argc, char *argv[]) {
    int d = 0;
    f(d);
    printf("%d\n", d);

    int *p = malloc(sizeof(int));
    *p = 0;
    g(p);
    printf("%d\n", *p);
    free(p);
}
```



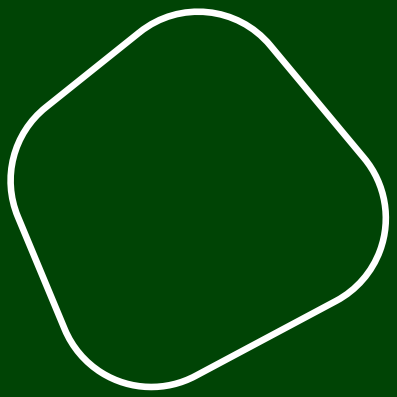


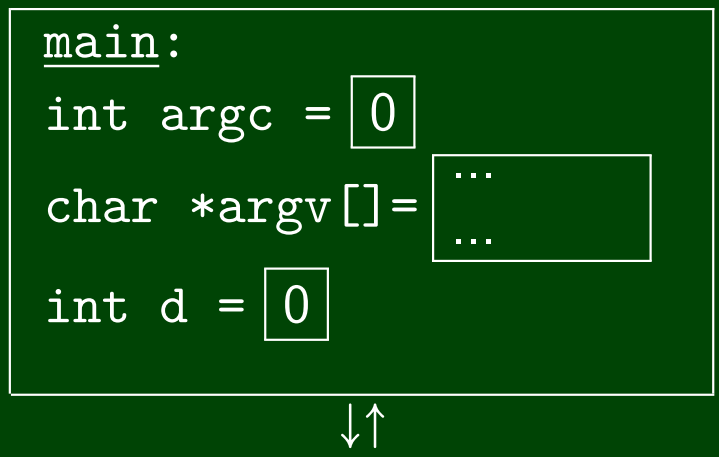
Investigating By Analogy to `int`

```
1  void f(int i) {
2     i = 5;
3  }
```

```
1  void g(int *pi) {
2     *pi = 5;
3  }
```

```
1  int main(int argc, char *argv[]) {
2     int d = 0;
3     f(d);
4     printf("%d\n", d);
5
6     int *p = malloc(sizeof(int));
7     *p = 0;
8     g(p);
9     printf("%d\n", *p);
10    free(p);
11 }
```



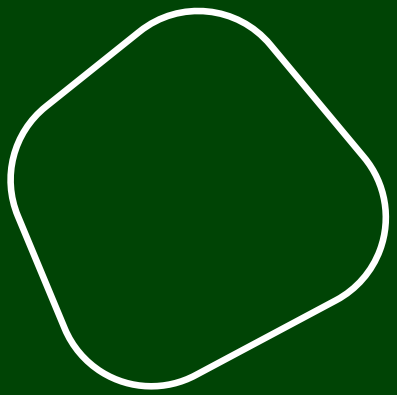


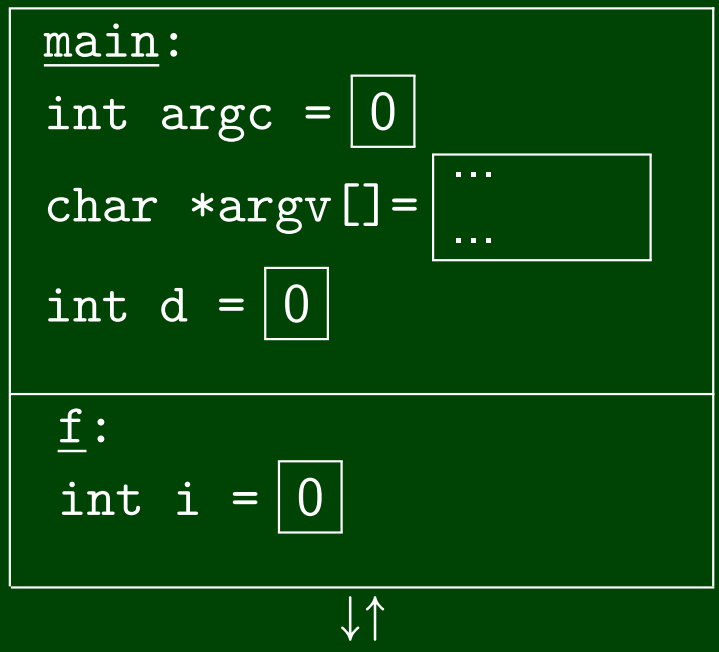
Investigating By Analogy to `int`

```
1  void f(int i) {
2     i = 5;
3  }
```

```
1  void g(int *pi) {
2     *pi = 5;
3  }
```

```
1  int main(int argc, char *argv[]) {
2     int d = 0;
3     f(d);
4     printf("%d\n", d);
5
6     int *p = malloc(sizeof(int));
7     *p = 0;
8     g(p);
9     printf("%d\n", *p);
10    free(p);
11 }
```



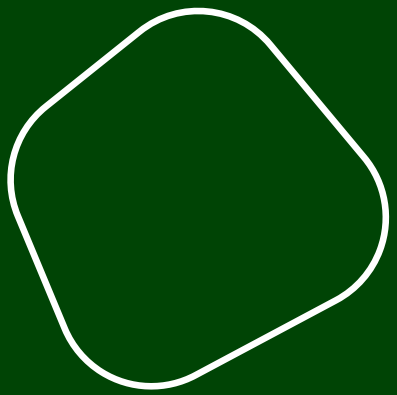


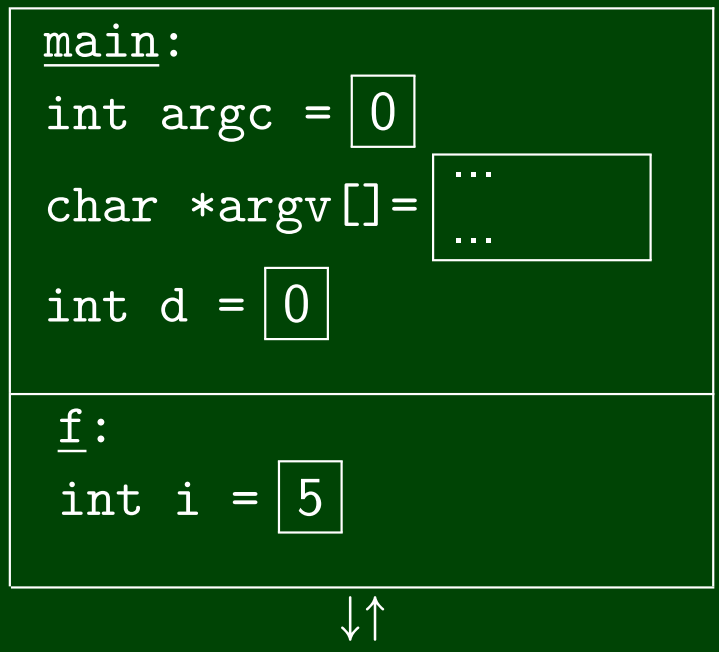
Investigating By Analogy to `int`

```
1  void f(int i) {
2     i = 5;
3  }
```

```
1  void g(int *pi) {
2     *pi = 5;
3  }
```

```
1  int main(int argc, char *argv[]) {
2     int d = 0;
3     f(d);
4     printf("%d\n", d);
5
6     int *p = malloc(sizeof(int));
7     *p = 0;
8     g(p);
9     printf("%d\n", *p);
10    free(p);
11 }
```



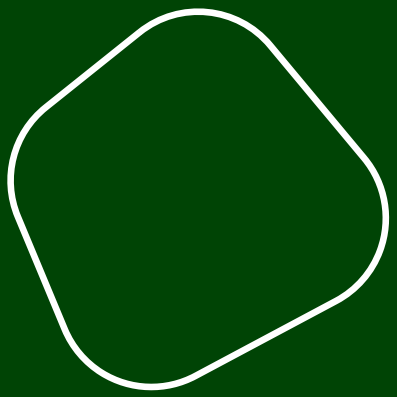


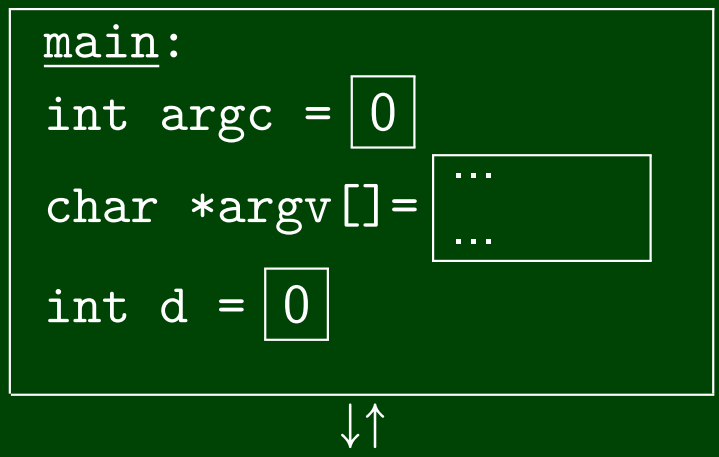
Investigating By Analogy to `int`

```
1  void f(int i) {
2     i = 5;
3  }
```

```
1  void g(int *pi) {
2     *pi = 5;
3  }
```

```
1  int main(int argc, char *argv[]) {
2     int d = 0;
3     f(d);
4     printf("%d\n", d);
5
6     int *p = malloc(sizeof(int));
7     *p = 0;
8     g(p);
9     printf("%d\n", *p);
10    free(p);
11 }
```



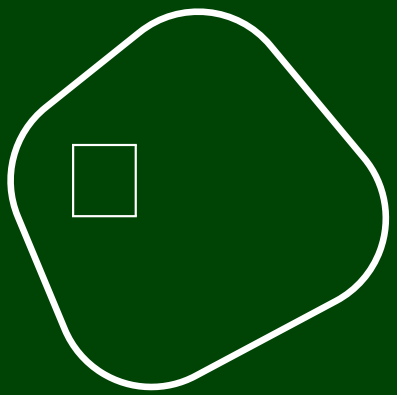


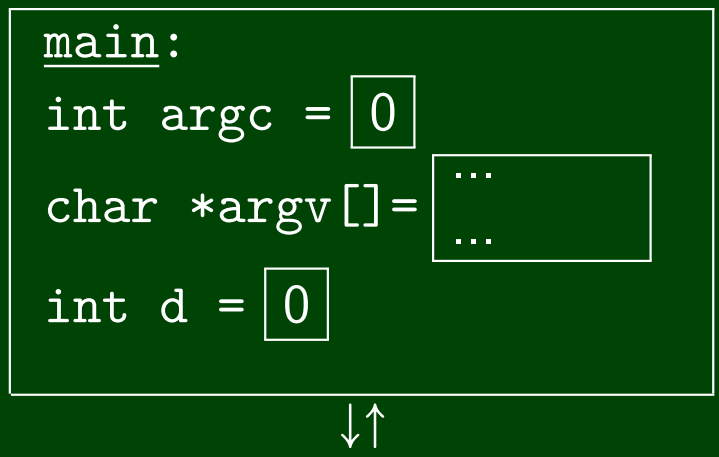
Investigating By Analogy to `int`

```
1  void f(int i) {
2     i = 5;
3  }
```

```
1  void g(int *pi) {
2     *pi = 5;
3  }
```

```
1  int main(int argc, char *argv[]) {
2     int d = 0;
3     f(d);
4     printf("%d\n", d);
5
6     int *p = malloc(sizeof(int));
7     *p = 0;
8     g(p);
9     printf("%d\n", *p);
10    free(p);
11 }
```



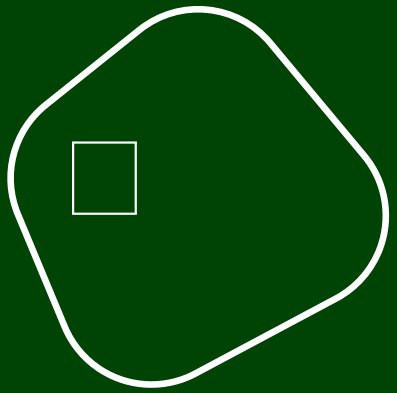


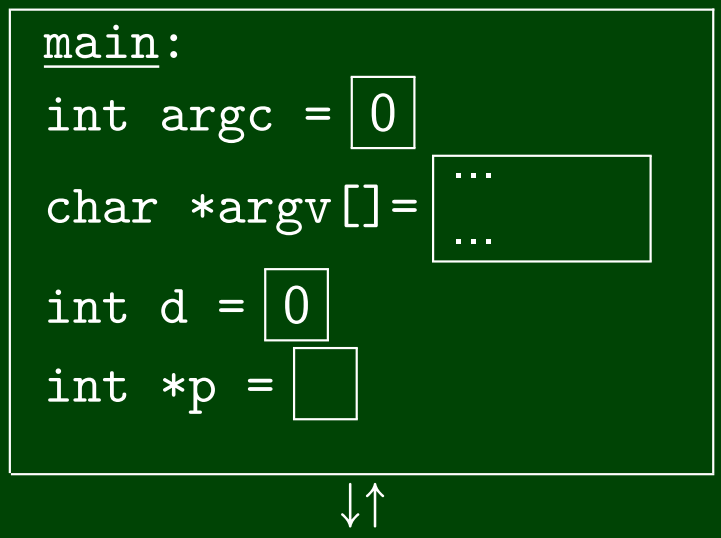
Investigating By Analogy to `int`

```
1  void f(int i) {
2     i = 5;
3  }
```

```
1  void g(int *pi) {
2     *pi = 5;
3  }
```

```
1  int main(int argc, char *argv[]) {
2     int d = 0;
3     f(d);
4     printf("%d\n", d);
5
6     int *p = malloc(sizeof(int));
7     *p = 0;
8     g(p);
9     printf("%d\n", *p);
10    free(p);
11 }
```



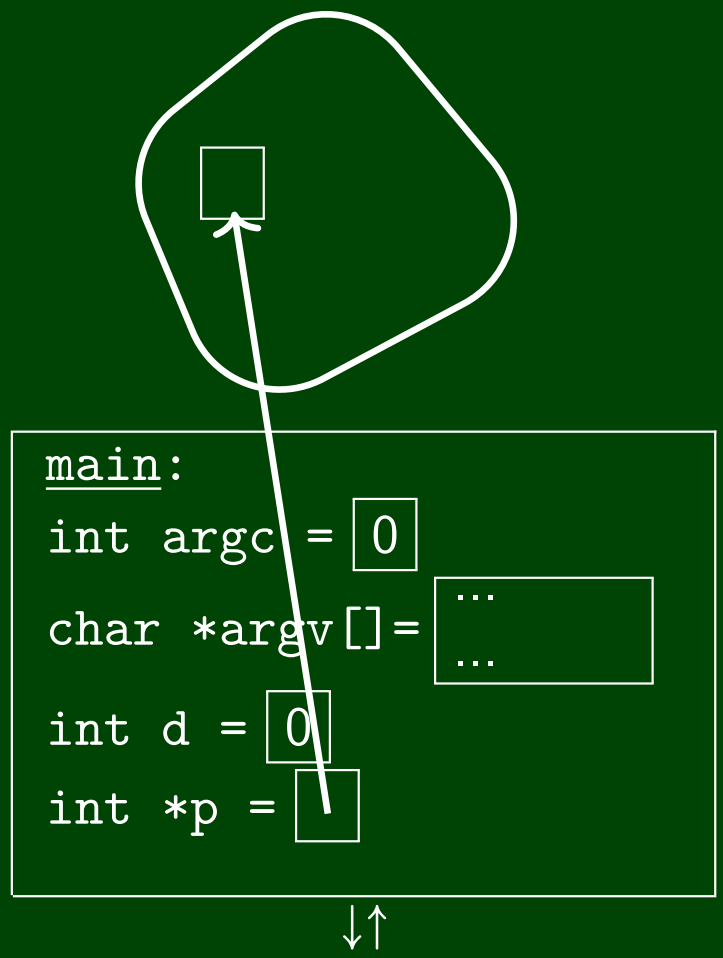


Investigating By Analogy to `int`


```
1  void f(int i) {
2     i = 5;
3  }
```

```
1  void g(int *pi) {
2     *pi = 5;
3  }
```

```
1  int main(int argc, char *argv[]) {
2     int d = 0;
3     f(d);
4     printf("%d\n", d);
5
6     int *p = malloc(sizeof(int));
7     *p = 0;
8     g(p);
9     printf("%d\n", *p);
10    free(p);
11 }
```

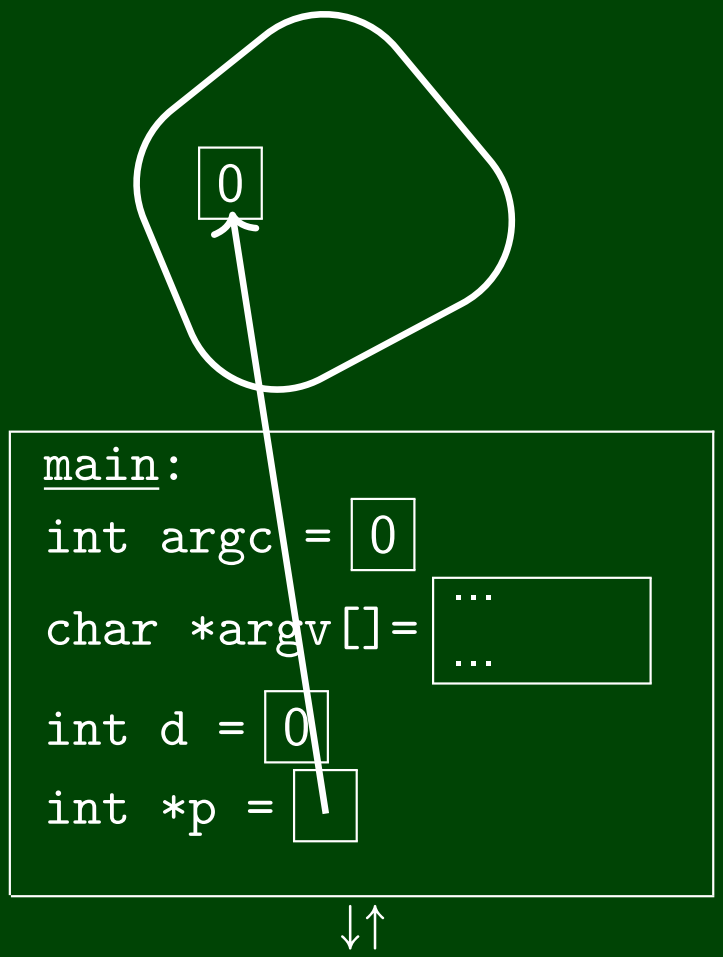


Investigating By Analogy to `int`


```
1  void f(int i) {
2     i = 5;
3  }
```

```
1  void g(int *pi) {
2     *pi = 5;
3  }
```

```
1  int main(int argc, char *argv[]) {
2     int d = 0;
3     f(d);
4     printf("%d\n", d);
5
6     int *p = malloc(sizeof(int));
7     *p = 0;
8     g(p);
9     printf("%d\n", *p);
10    free(p);
11 }
```

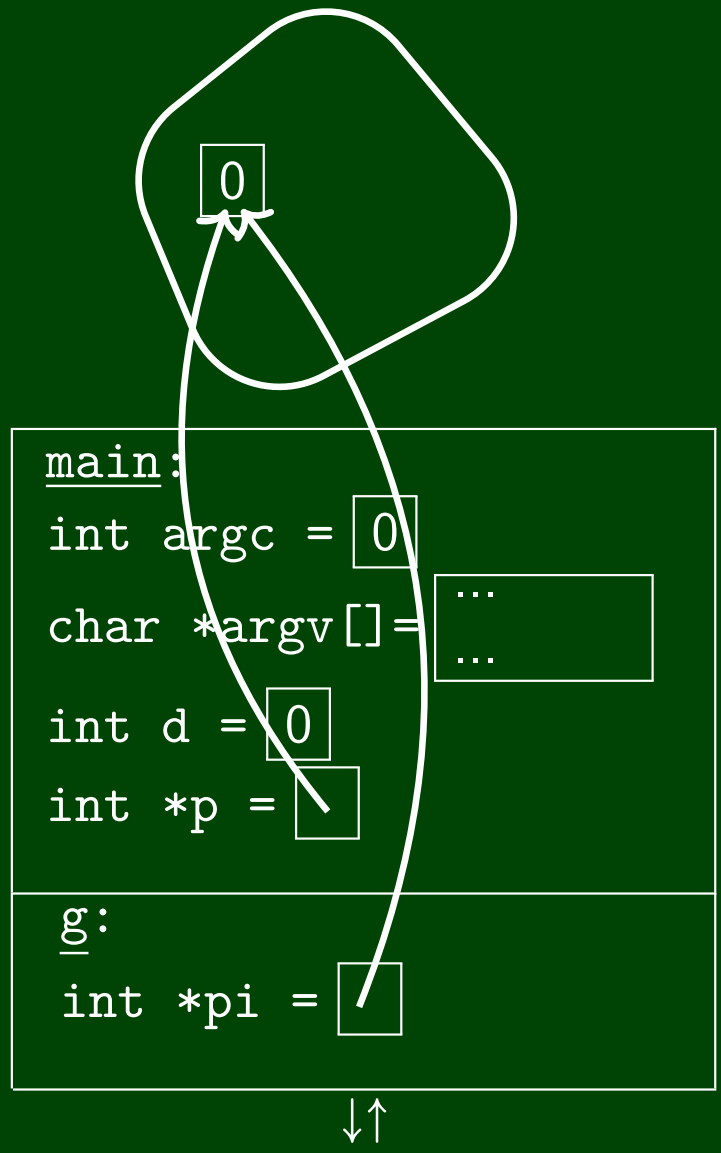


Investigating By Analogy to `int`

```
1  void f(int i) {
2     i = 5;
3  }
```

```
1  void g(int *pi) {
2     *pi = 5;
3  }
```

```
1  int main(int argc, char *argv[]) {
2     int d = 0;
3     f(d);
4     printf("%d\n", d);
5
6     int *p = malloc(sizeof(int));
7     *p = 0;
8     g(p);
9     printf("%d\n", *p);
10    free(p);
11 }
```

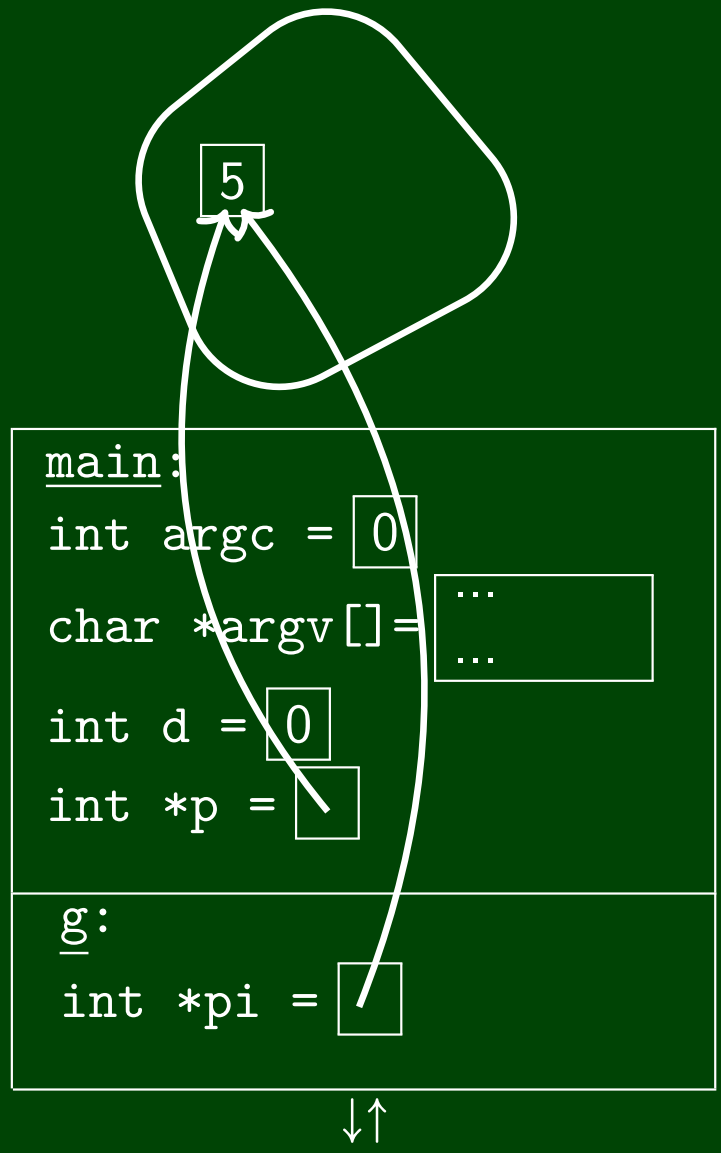


Investigating By Analogy to `int`

```
1  void f(int i) {
2     i = 5;
3  }
```

```
1  void g(int *pi) {
2     *pi = 5;
3  }
```

```
1  int main(int argc, char *argv[]) {
2     int d = 0;
3     f(d);
4     printf("%d\n", d);
5
6     int *p = malloc(sizeof(int));
7     *p = 0;
8     g(p);
9     printf("%d\n", *p);
10    free(p);
11 }
```

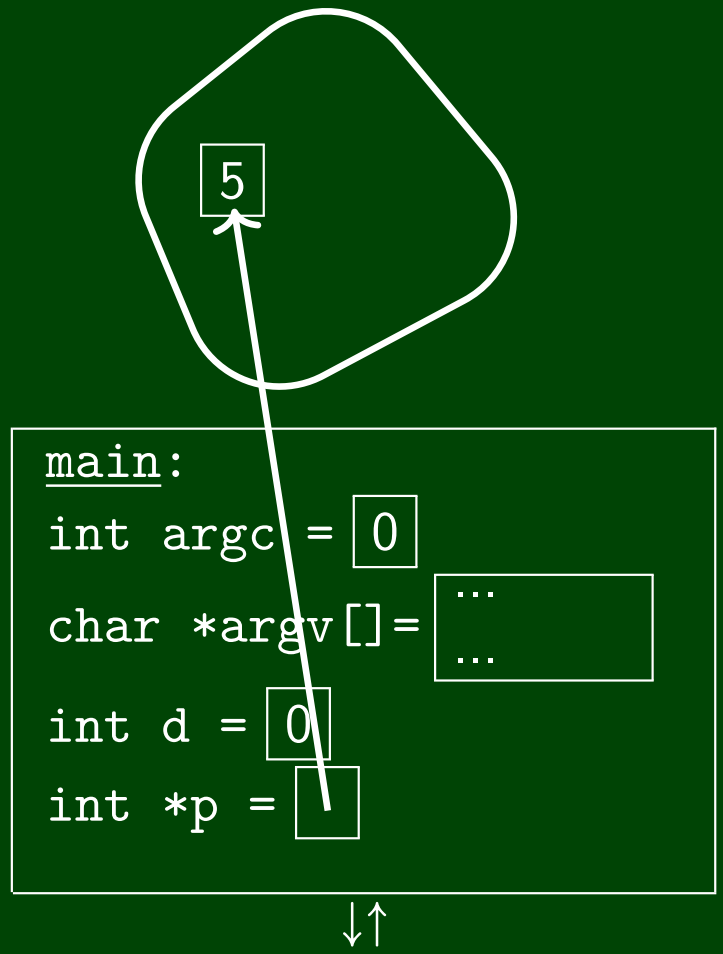


Investigating By Analogy to `int`

```
1  void f(int i) {
2     i = 5;
3  }
```

```
1  void g(int *pi) {
2     *pi = 5;
3  }
```

```
1  int main(int argc, char *argv[]) {
2     int d = 0;
3     f(d);
4     printf("%d\n", d);
5
6     int *p = malloc(sizeof(int));
7     *p = 0;
8     g(p);
9     printf("%d\n", *p);
10    free(p);
11 }
```

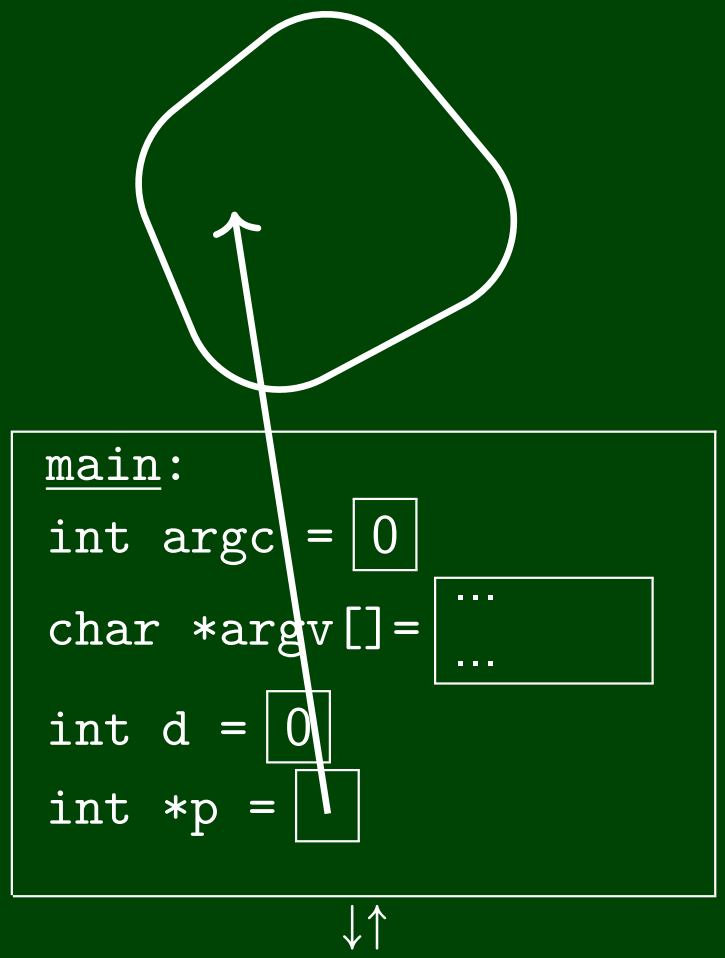


Investigating By Analogy to `int`


```
1  void f(int i) {
2     i = 5;
3  }
```

```
1  void g(int *pi) {
2     *pi = 5;
3  }
```

```
1  int main(int argc, char *argv[]) {
2     int d = 0;
3     f(d);
4     printf("%d\n", d);
5
6     int *p = malloc(sizeof(int));
7     *p = 0;
8     g(p);
9     printf("%d\n", *p);
10    free(p);
11 }
```



pixel_t in C

```
1  typedef struct pixel {
2     uint8_t red;
3     uint8_t green;
4     uint8_t blue;
5  } pixel_t;
6
7
8  void pixel_zero_red(pixel_t *p) {
9     p->red = 0;
10 }
```

So, what does this do?

```
1  void pixel_zero_without_star(pixel_t p) {
2     p.red = 0;
3  }
```

Allocating on the Stack


```
1  // Initializes pixel on the stack
2  pixel_t p;
3  p.red = 0;
4  p.green = 255;
5  p.blue = 0;
6
7  // Shorthand for initializing pixel on the stack
8  pixel_t p2 = {
9      .red = 0,
10     .green = 255,
11     .blue = 0
12 };
```

```
1  pixel_t **foo = malloc(n * sizeof(pixel_t *));
```