APS105
Winter 2012

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

Lecture 23
March 14, 2012

Today

- Dynamic Memory Allocation

- No CodeLab this week

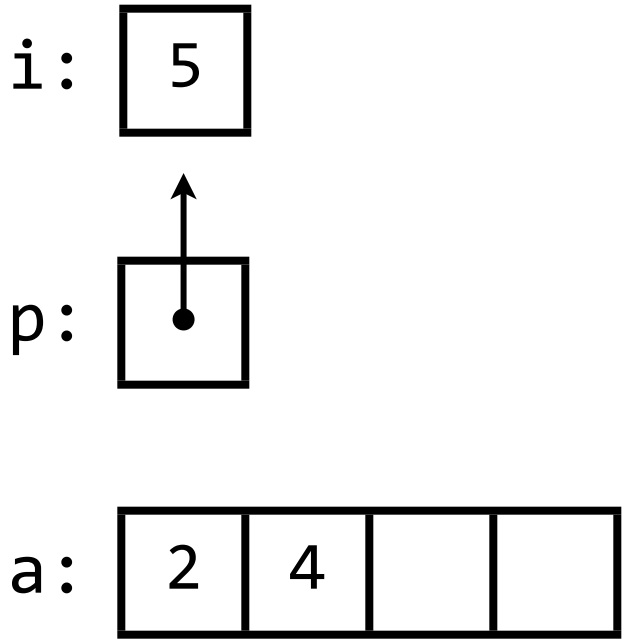
Dynamic Memory Allocation

`malloc()` and friends

Automatic Allocation

- We've been doing this since the first week
- Variables have a name and an address

```
int i;
i = 5;

int *p = &i;

int a[4];
a[0] = 2;
a[1] = 4;
```

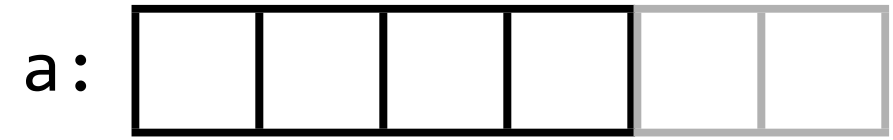


Fixed Sizes

- Automatically allocated variables can't change size
- Until C99, array sizes needed to be fixed at compile time

Pre-C99

```
int a[4];
```



C99

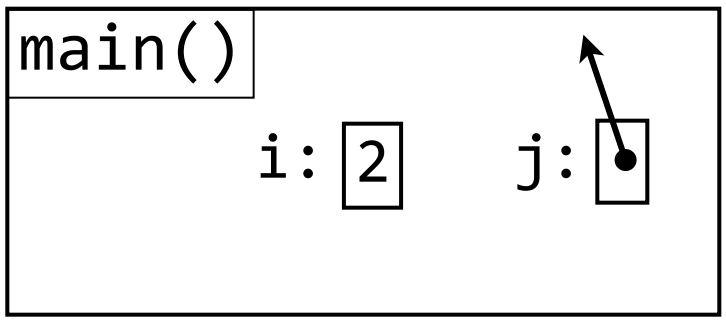
```
int n;
scanf("%d", &n);
int a[n];
```

```
int *square2(int *p)
{
  int result = *p * *p;
  return &result;
}

int main (void)
{
  int i = 2;
  int *j = square2(&i);
  printf("%d", *j);

  return 0;
}
```

```
int *square2(int *p)
{
  int result = *p * *p;
  return &result;
}

int main (void)
{
  int i = 2;
  int *j = square2(&i);
  printf("%d", *j);

  return 0;
}
```

Wrong

????



Dynamic Memory Allocation

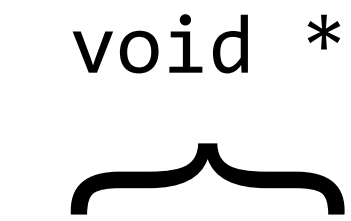
- Alternate way of allocating memory
- Obtained from a different pool of memory

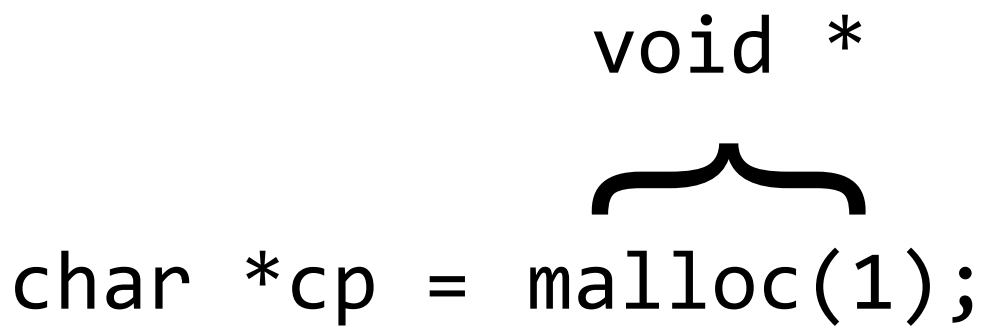
malloc()

```
void *malloc(size_t size);
```

for now think of it as int

- "Memory Allocator"
- `#include <stdlib.h>`
- You ask `malloc()` for some memory, it finds some, and then gives it to you
- `void *` pointer ("generic" pointer)
- Different pool of memory (called the heap)


void *
malloc(1);


void *
char *cp = malloc(1);

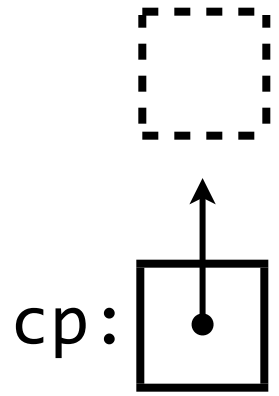
```
                  void *
                  ⏞
char *cp =  malloc(1);
```

```
  char c =  malloc(1);
```

Wrong

```
int i = 5;
int j = &i;
```

Dotted border means
"dynamically allocated"



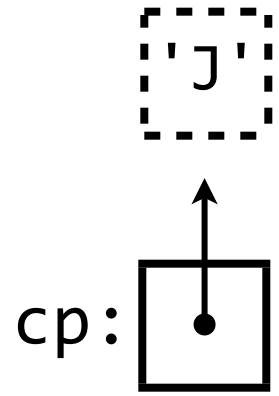

```
        void *
          ⏟
char *cp = malloc(1);
```

```
  char c = malloc(1);
```

Wrong

```
int i = 5;
int j = &i;
```

Dotted border means
"dynamically allocated"

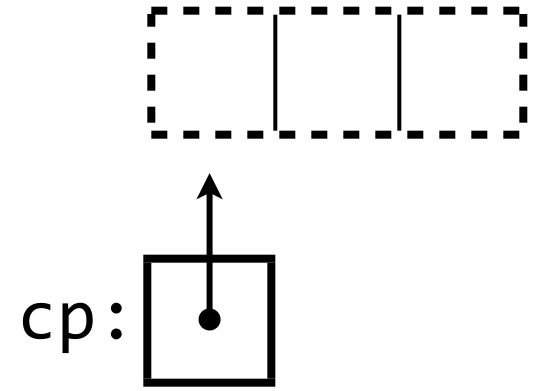



```
                void *
              ⏞
char *cp = malloc(1);

  char c = malloc(1);
```

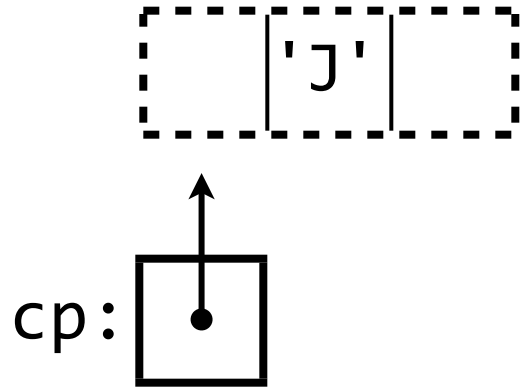
Wrong

```
   *cp = 'J';
```

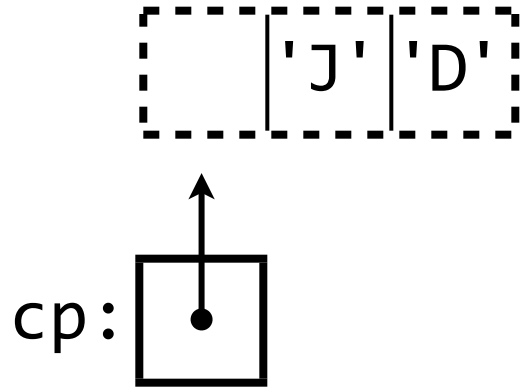
```
int i = 5;
int j = &i;
```



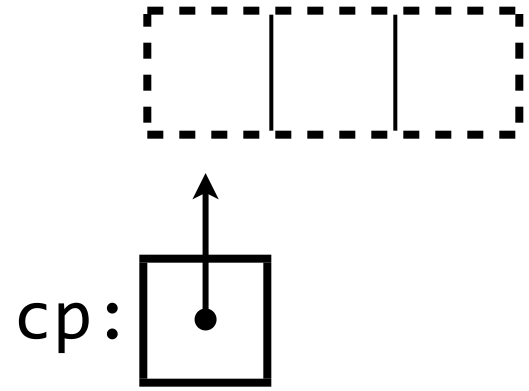

```
char *cp = malloc(3);
```



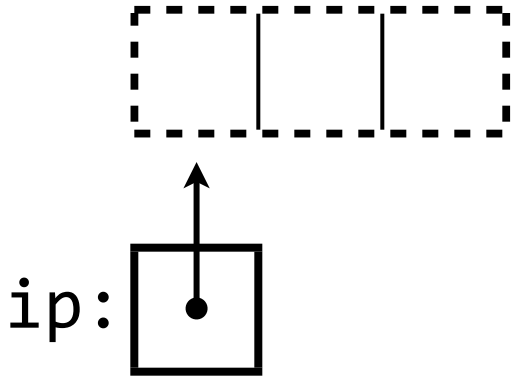

```
char *cp = malloc(3);

*(cp + 1) = 'J';
```

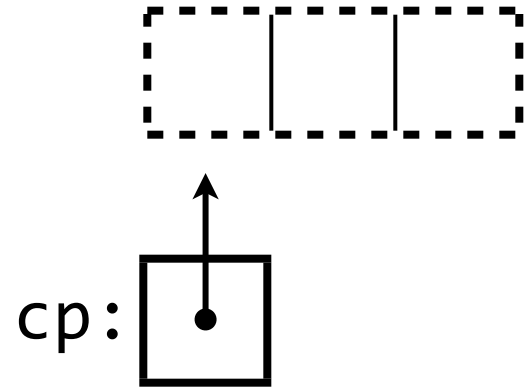



```
char *cp = malloc(3);

*(cp + 1) = 'J';
cp[2] = 'D';
```

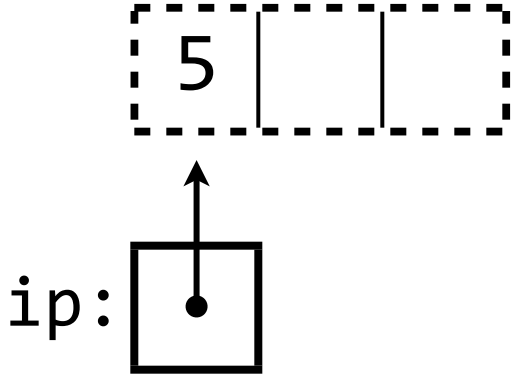


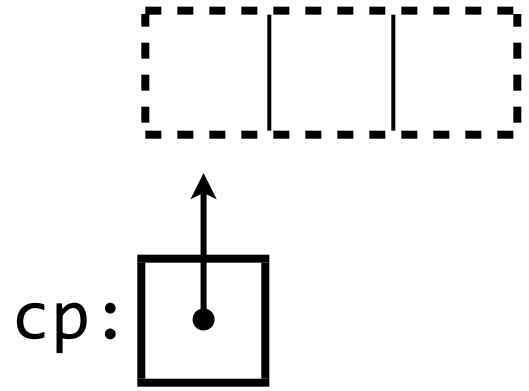

```
char *cp = malloc(3);

 int *ip = malloc(3);
```





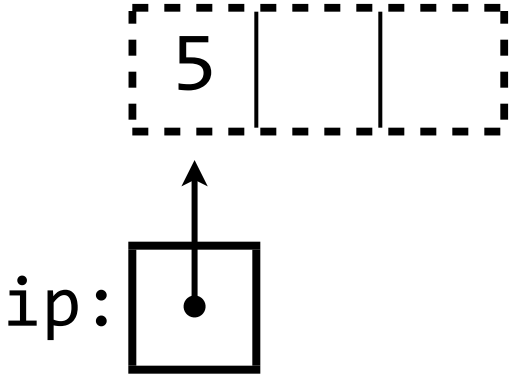

```
char *cp = malloc(3);

int *ip = malloc(3);

ip[0] = 5;
```






```
char *cp = malloc(3);

int *ip = malloc(3);

  ip[0] = 5;
```

Wrong

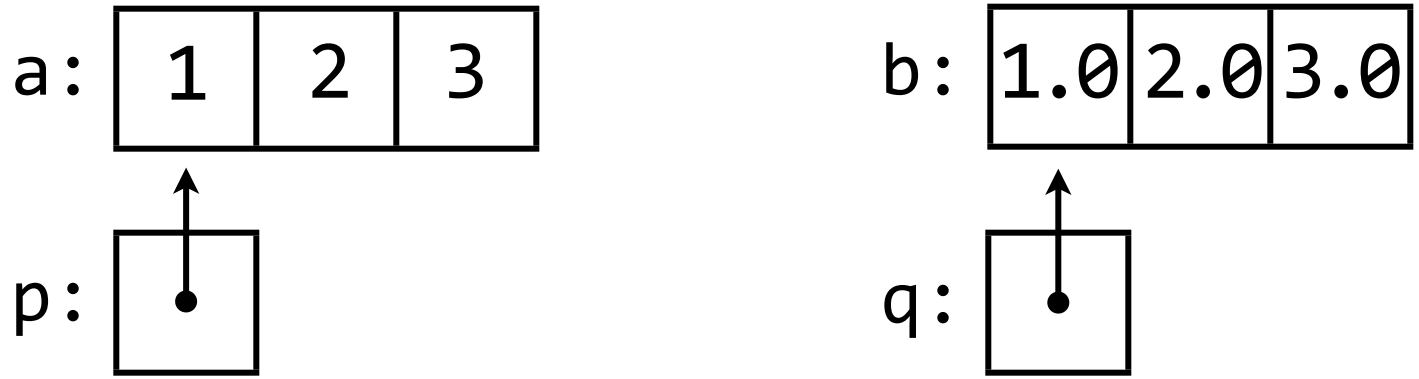


`char` is 1 byte (everywhere)

`int` is 4 bytes (on ECF)

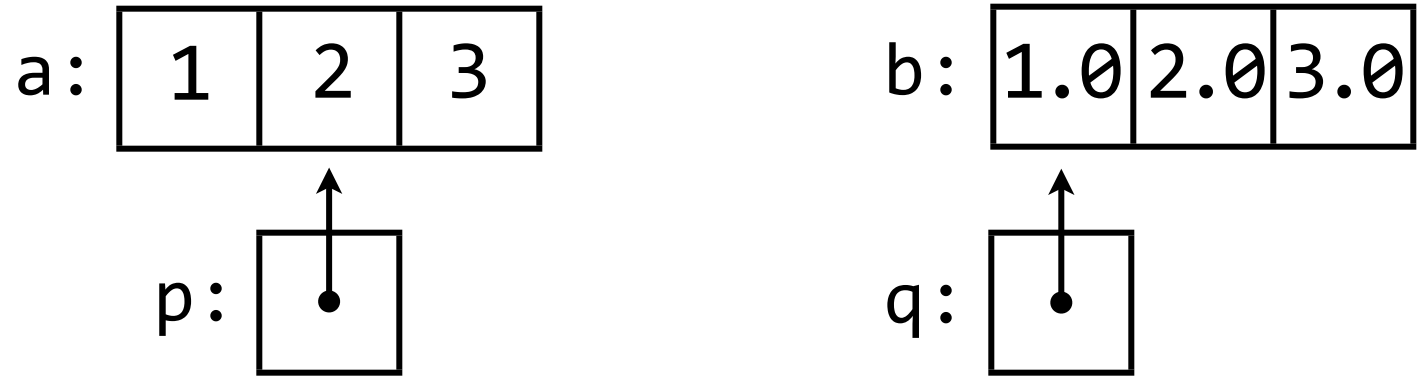
Scaling

```
int a[] = {1,2,3};
double b[] = {1.0, 2.0, 3.0}
int *p = &a[0];
double *q = &b[0];
```



Scaling

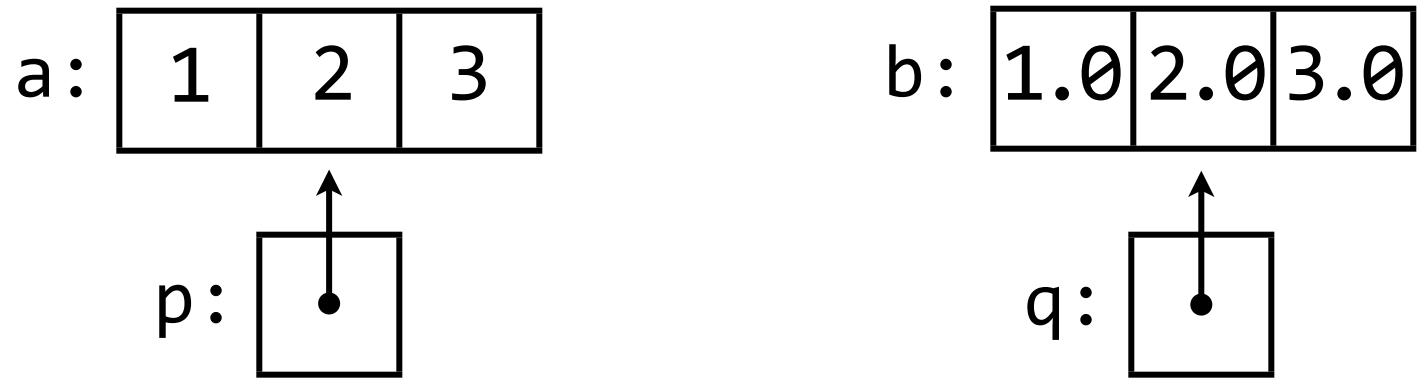
```
int a[] = {1,2,3};
double b[] = {1.0, 2.0, 3.0}
int *p = &a[0];
double *q = &b[0];
```




```
p++;
```

Scaling

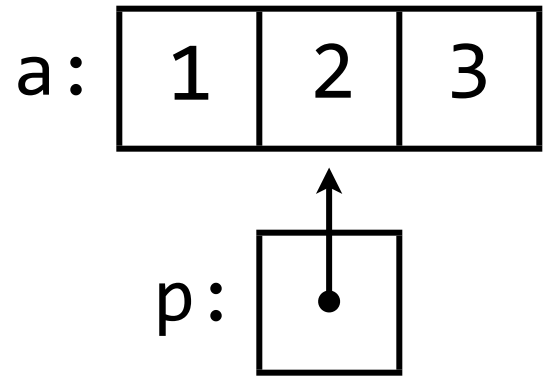
```
int a[] = {1,2,3};
double b[] = {1.0, 2.0, 3.0}
int *p = &a[0];
double *q = &b[0];
```

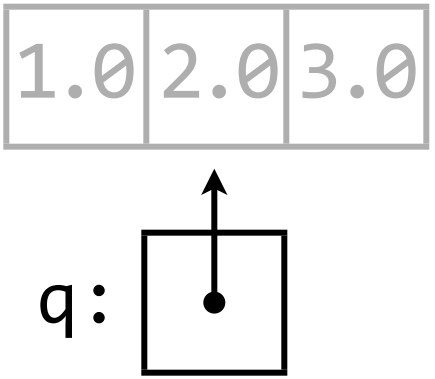



```
p++;
q++;
```

Scaling

```
int a[] = {1,2,3};
double b[] = {1.0, 2.0, 3.0}
int *p = &a[0];
double *q = &b[0];
```





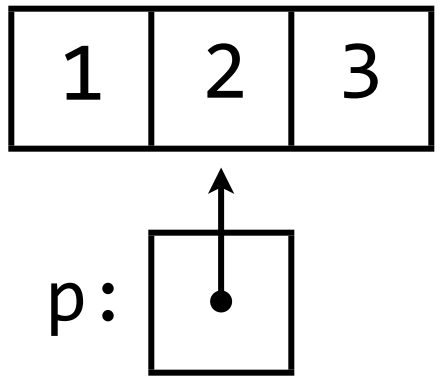

```
p++;
q++;
```

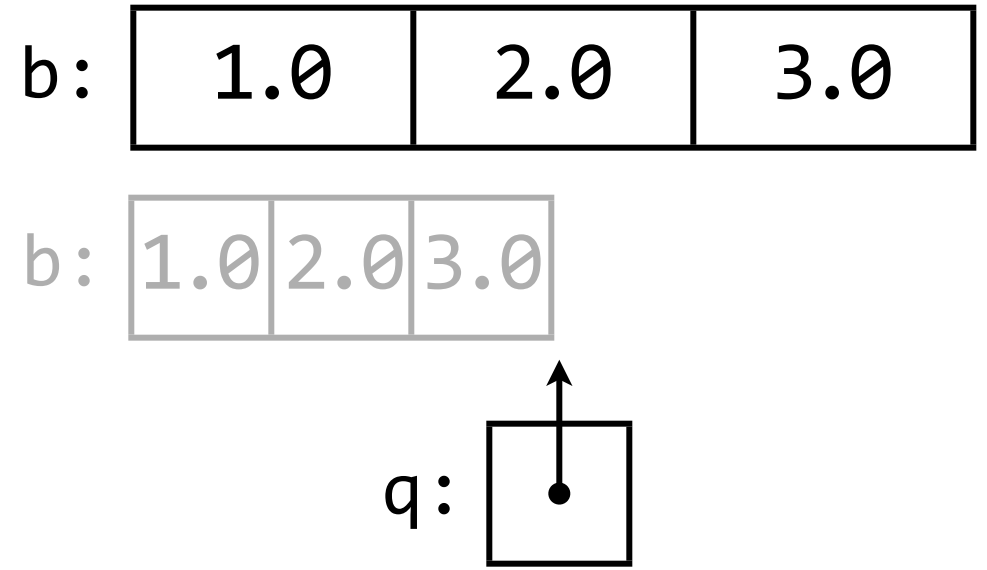
`int` is 4 bytes on ECF

`double` is 8 bytes on ECF

Scaling

```
int a[] = {1,2,3};
double b[] = {1.0, 2.0, 3.0}
int *p = &a[0];
double *q = &b[0];
```

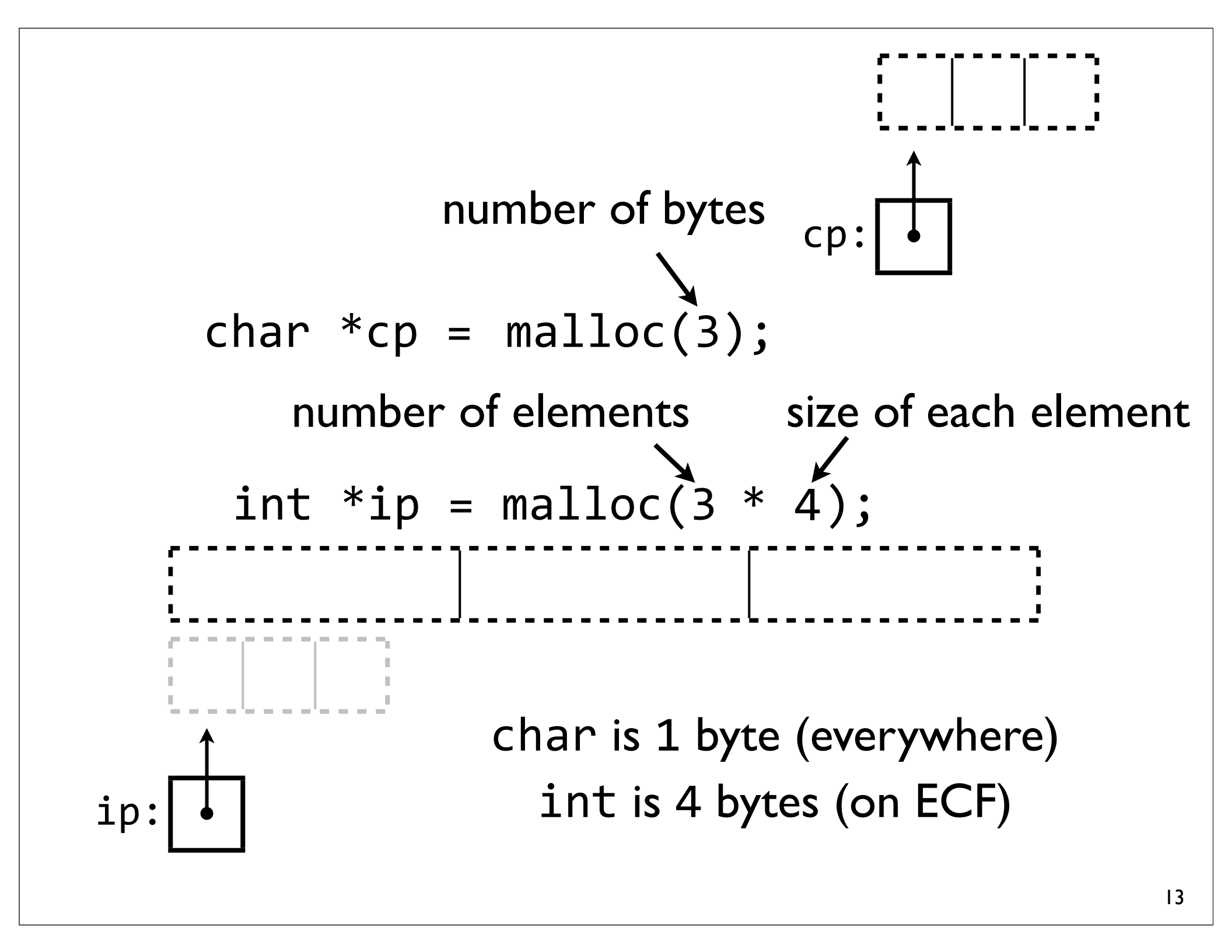





```
p++;
q++;
```

`int` is 4 bytes on ECF

`double` is 8 bytes on ECF


number of bytes
cp:
char *cp = malloc(3);
number of elements
size of each element
int *ip = malloc(3 * 4);
ip:
char is 1 byte (everywhere)
int is 4 bytes (on ECF)

sizeof()

- C operator that tells us how big something is

```
sizeof(char);
```
1

```
sizeof(int);
```
4 (on ECF)

```
int i = 42;
sizeof(i);
```
4 (on ECF)

```
int size = sizeof(i);
```

(not 100% correct)

sizeof() and malloc()

```
int *ip = malloc(3 * 4);
```

```
int *ip = malloc(3 * sizeof(int));
```

You should always use `sizeof()`

```
char *cp = malloc(3);
```

```
char *cp = malloc(3 * sizeof(char));
```

`sizeof()` and Arrays

- You can also use `sizeof()` on arrays

```
int a[] = {2, 4, 8, 16};

sizeof(a);
```

16 (on ECF)

`4 * sizeof(int)`

```
int length = sizeof(a) / sizeof(int);
```

4

`4 * sizeof(int) / sizeof(int)`

But I've said arrays don't know how big they are!

```
int a[] = {2, 4, 8, 16};
int *p = a;
```

`sizeof(a);` **16** **4 * sizeof(int)** (on ECF)

`sizeof(p);` **4** **sizeof(int *)** (on ECF)

```
int b[] = {2, 4, 8, 16, 32, 64, 128};
int *q = b;
```

`sizeof(b);` **28** **7 * sizeof(int)** (on ECF)

`sizeof(q);` **4** **sizeof(int *)** (on ECF)

`sizeof()` and Arrays

- If we have the array itself, we can figure out the length
- If we have a pointer to the array, we can't
  - And all arrays get passed to functions as pointers...

```
int *square2(int *p)
{
  int result = *p * *p;
  return &result;
}


int main (void)
{
  int i = 2;
  int *j = square2(&i);
  printf("%d", *j);

  return 0;
}
```

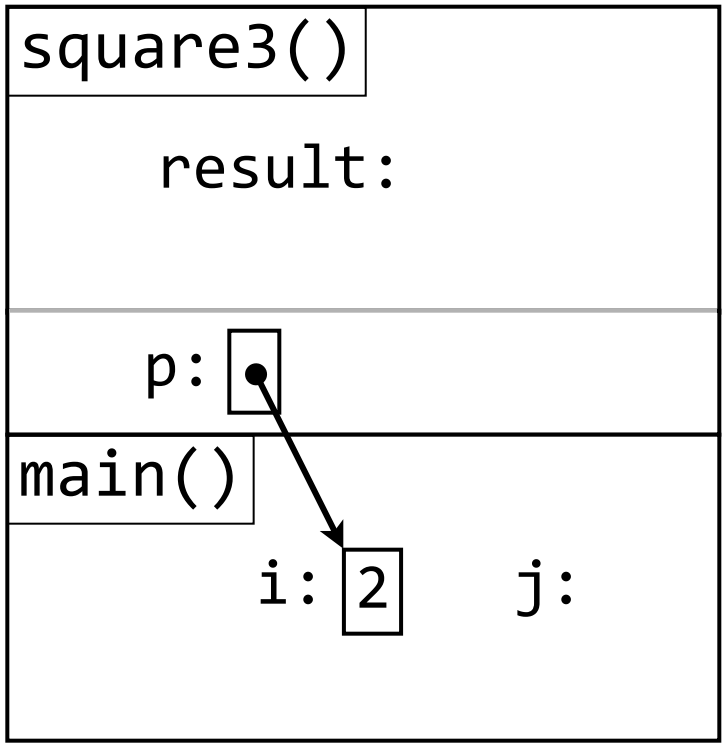
```
int *square3(int *p)
{
  int *result = malloc(sizeof(int));
  int result = *p * *p;
  return &result;
}


int main (void)
{
  int i = 2;
  int *j = square3(&i);
  printf("%d", *j);

  return 0;
}
```

```
int *square3(int *p)
{
  int *result = malloc(sizeof(int));
  result = *p * *p;
  return &result;
}


int main (void)
{
  int i = 2;
  int *j = square3(&i);
  printf("%d", *j);

  return 0;
}
```

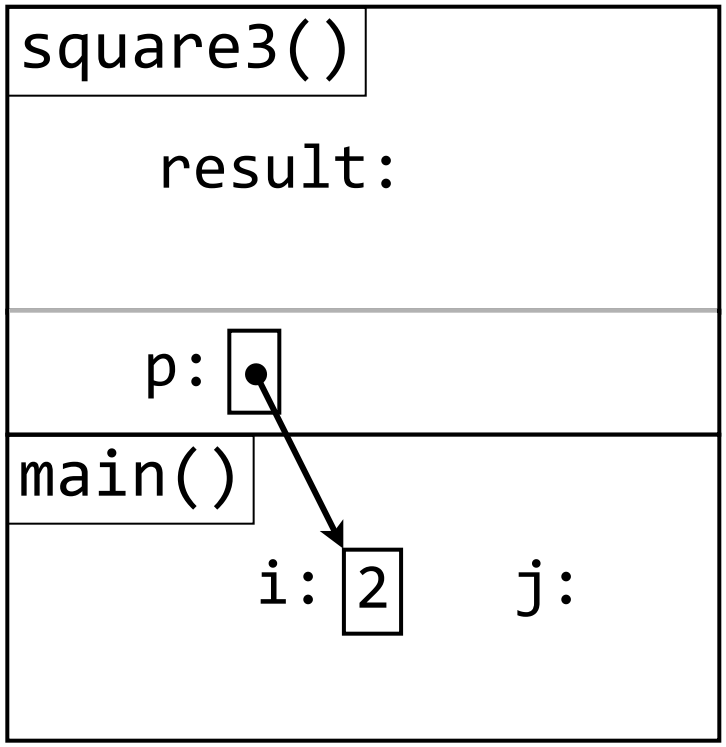
```
int *square3(int *p)
{
  int *result = malloc(sizeof(int));
  *result = *p * *p;
  return &result;
}


int main (void)
{
  int i = 2;
  int *j = square3(&i);
  printf("%d", *j);

  return 0;
}
```

```
int *square3(int *p)
{
  int *result = malloc(sizeof(int));
  *result = *p * *p;
  return result;
}


int main (void)
{
  int i = 2;
  int *j = square3(&i);
  printf("%d", *j);

  return 0;
}
```

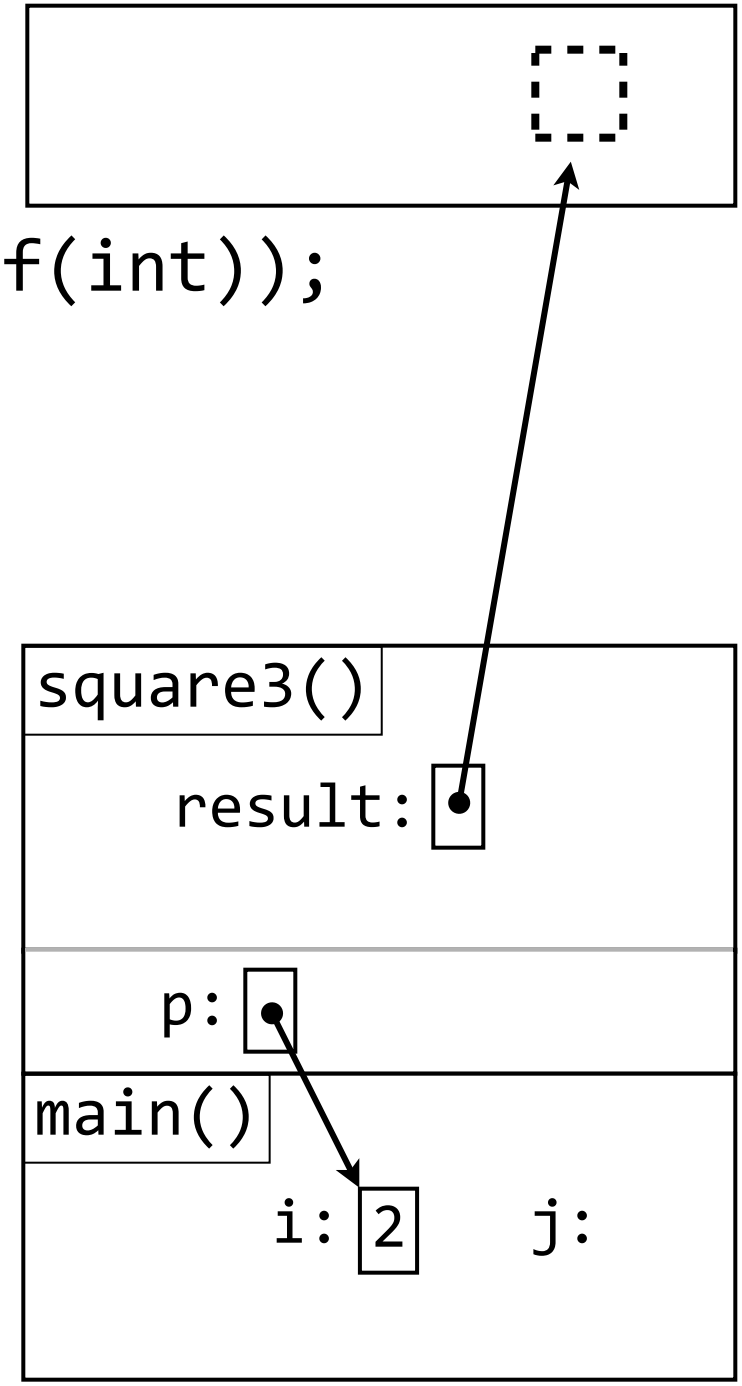
```
int *square3(int *p)
{
  int *result = malloc(sizeof(int));
  *result = *p * *p;
  return result;
}


int main (void)
{
  int i = 2;
  int *j = square3(&i);
  printf("%d", *j);

  return 0;
}
```



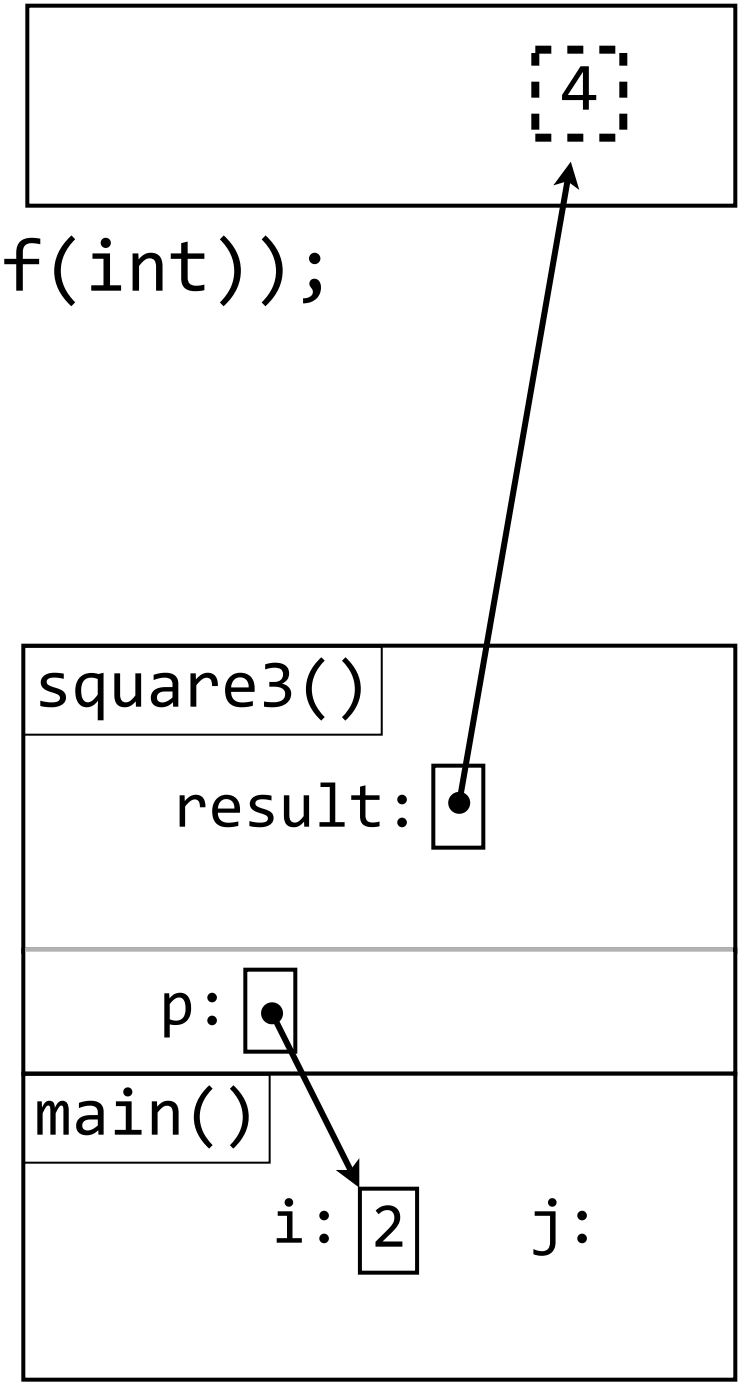
```
int *square3(int *p)
{
  int *result = malloc(sizeof(int));
  *result = *p * *p;
  return result;
}


int main (void)
{
  int i = 2;
  int *j = square3(&i);
  printf("%d", *j);

  return 0;
}
```



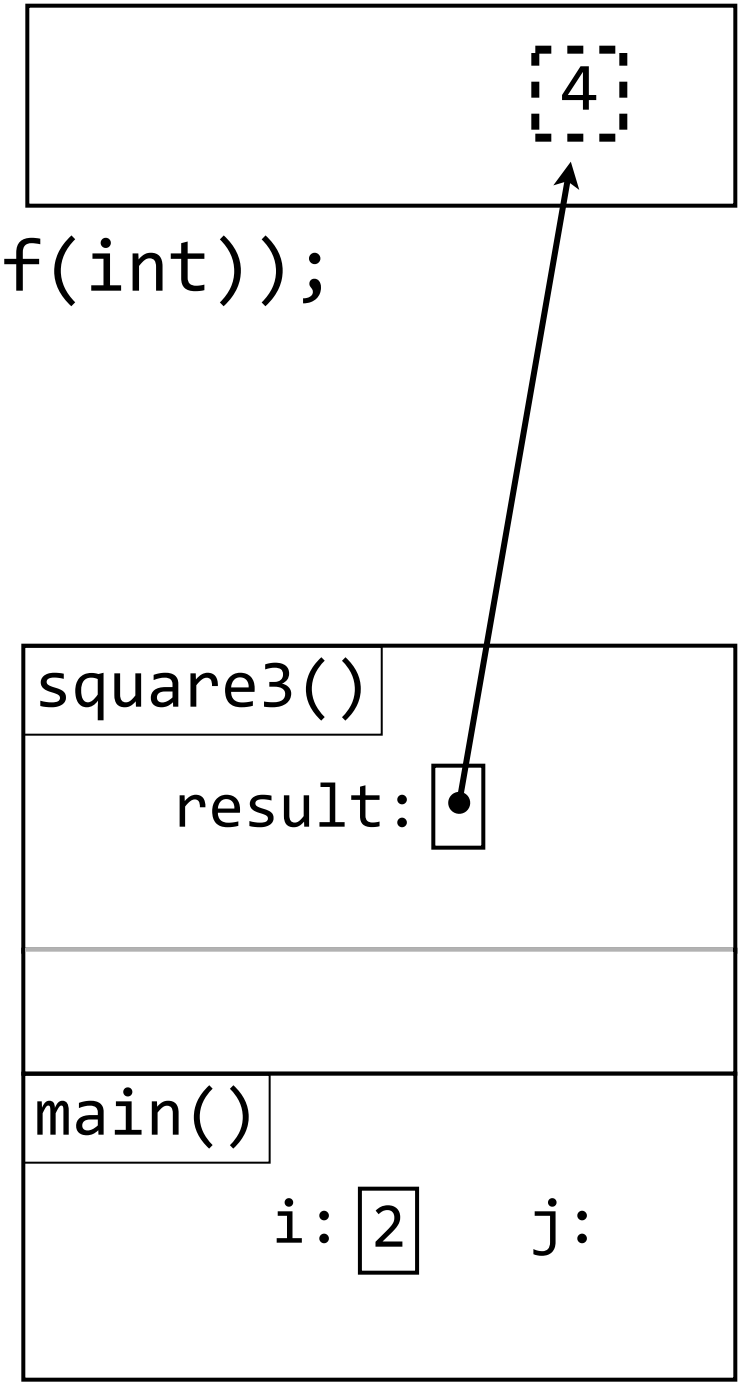
```
int *square3(int *p)
{
  int *result = malloc(sizeof(int));
  *result = *p * *p;
  return result;
}


int main (void)
{
  int i = 2;
  int *j = square3(&i);
  printf("%d", *j);

  return 0;
}
```



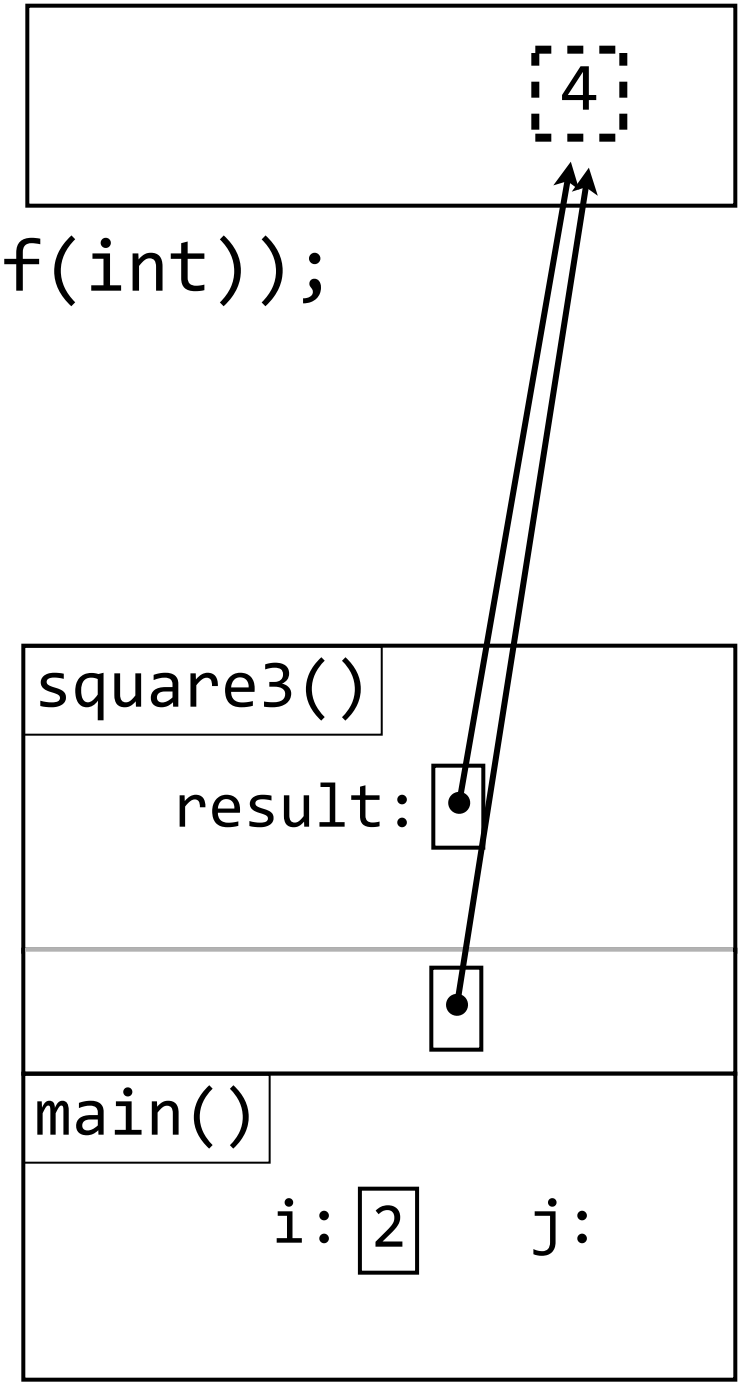
```
int *square3(int *p)
{
  int *result = malloc(sizeof(int));
  *result = *p * *p;
  return result;
}


int main (void)
{
  int i = 2;
  int *j = square3(&i);
  printf("%d", *j);

  return 0;
}
```



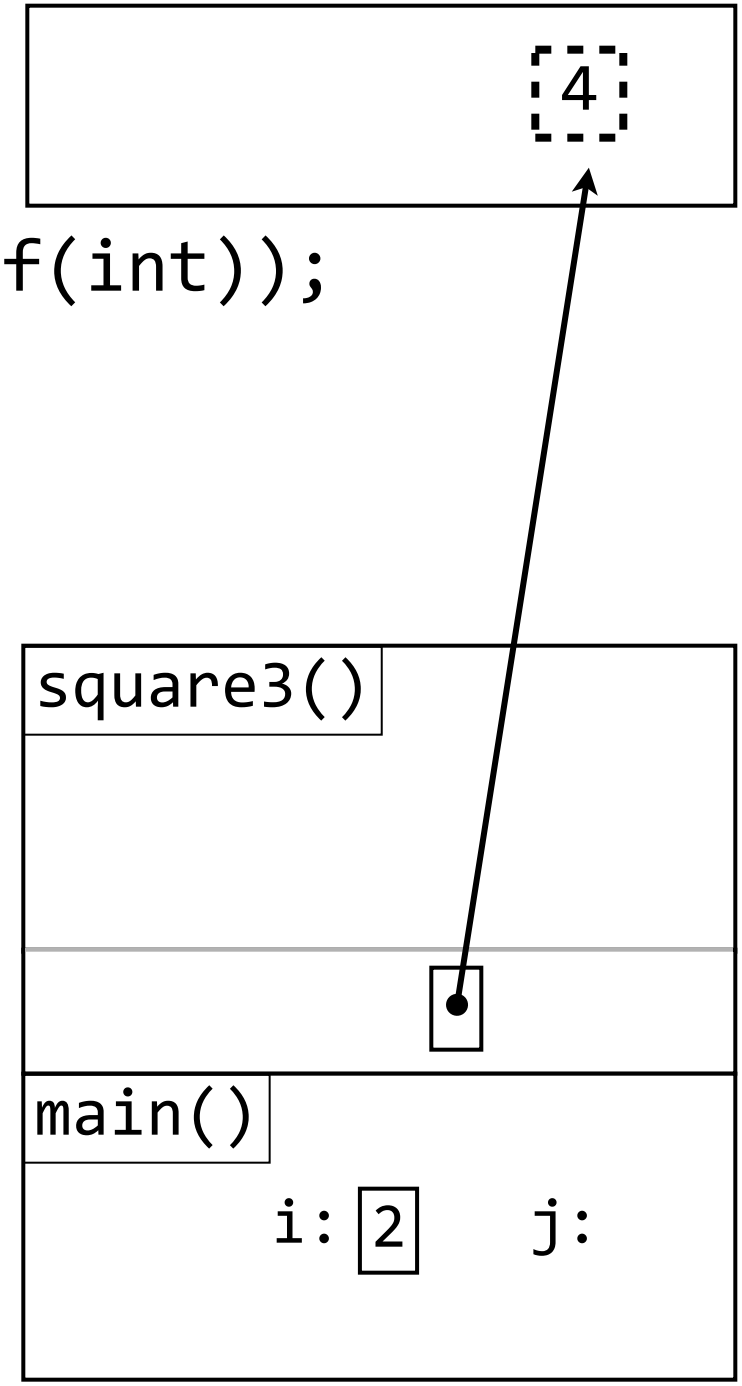
```
int *square3(int *p)
{
  int *result = malloc(sizeof(int));
  *result = *p * *p;
  return result;
}


int main (void)
{
  int i = 2;
  int *j = square3(&i);
  printf("%d", *j);

  return 0;
}
```



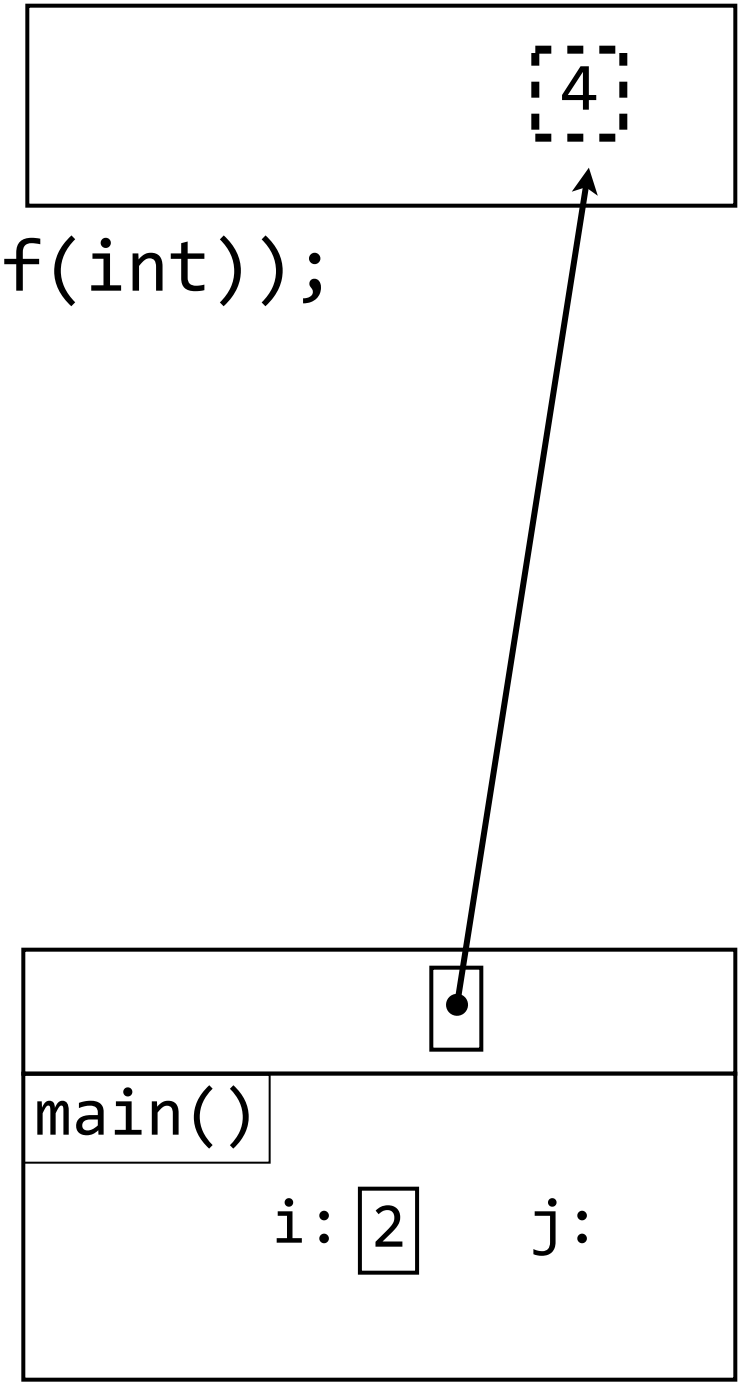
```
int *square3(int *p)
{
  int *result = malloc(sizeof(int));
  *result = *p * *p;
  return result;
}


int main (void)
{
  int i = 2;
  int *j = square3(&i);
  printf("%d", *j);

  return 0;
}
```



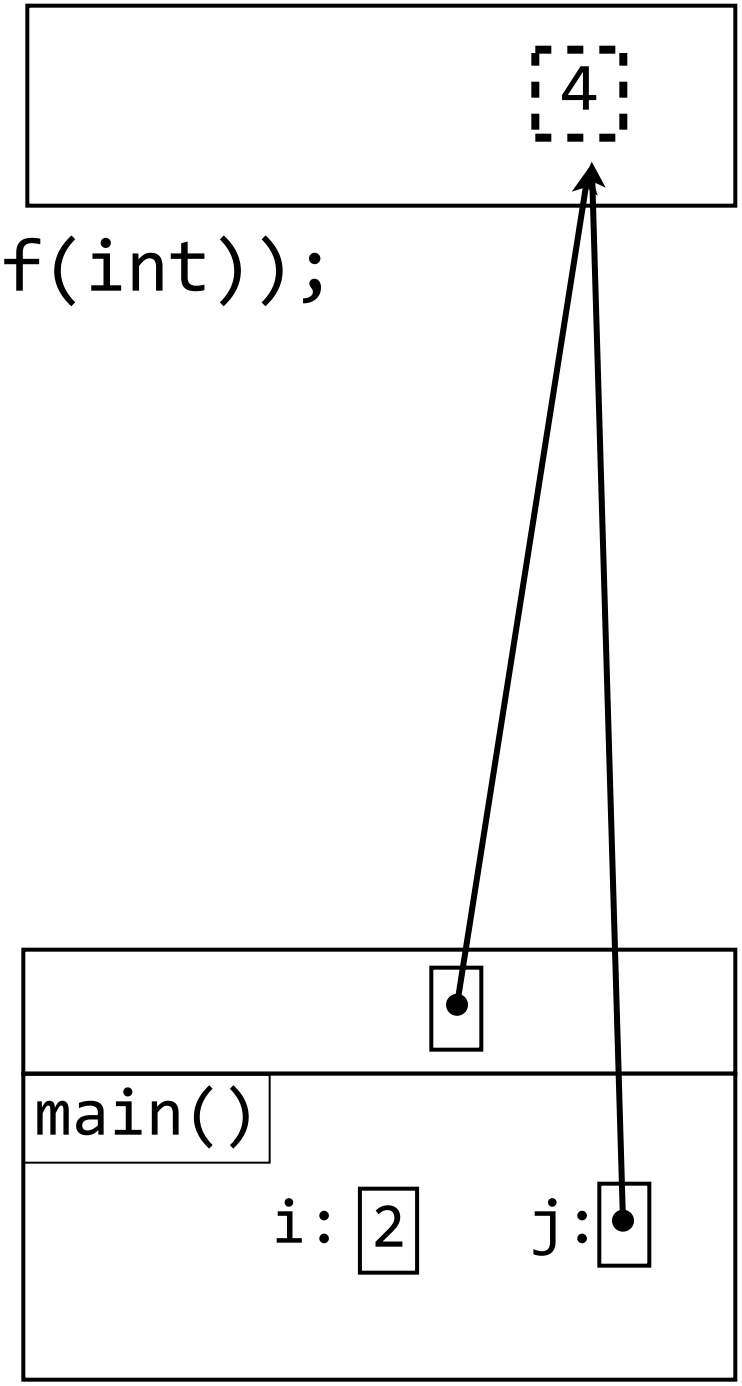
```
int *square3(int *p)
{
  int *result = malloc(sizeof(int));
  *result = *p * *p;
  return result;
}


int main (void)
{
  int i = 2;
  int *j = square3(&i);
  printf("%d", *j);

  return 0;
}
```



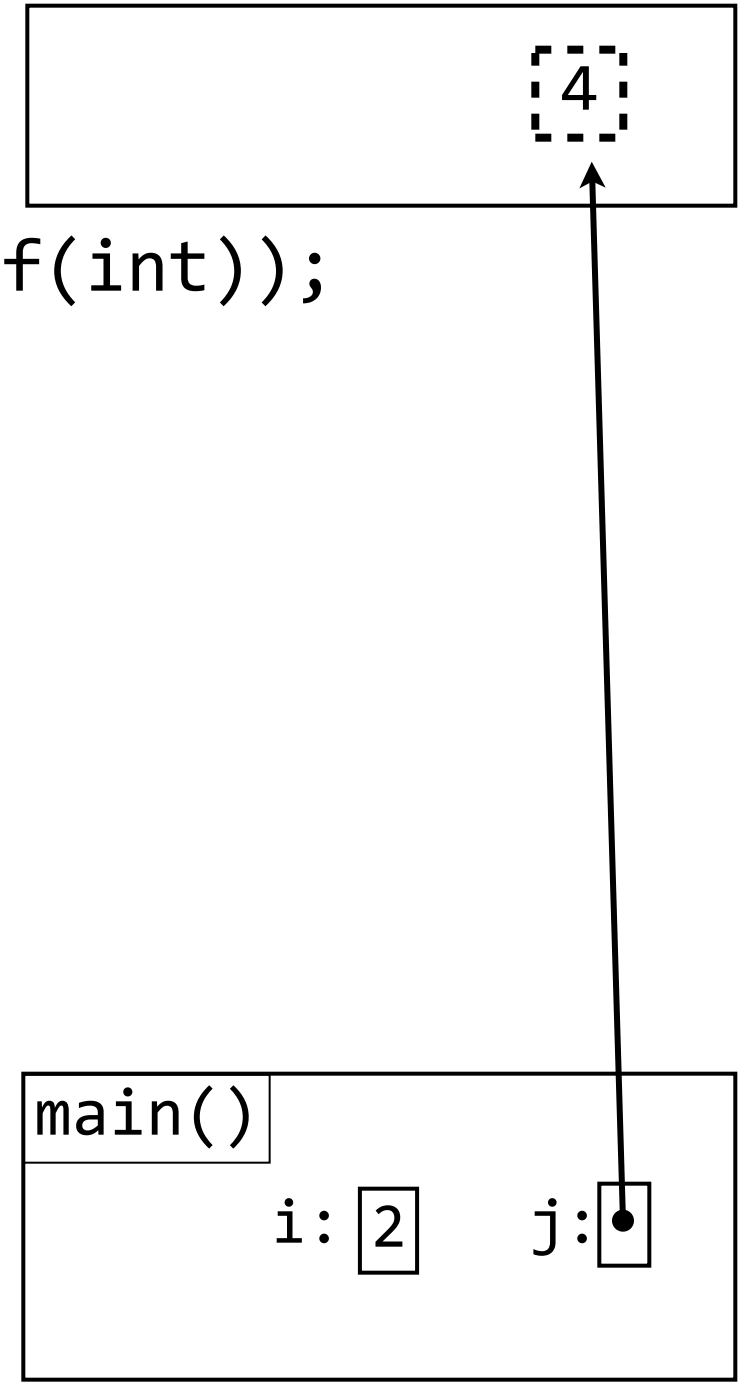
```
int *square3(int *p)
{
  int *result = malloc(sizeof(int));
  *result = *p * *p;
  return result;
}


int main (void)
{
  int i = 2;
  int *j = square3(&i);
  printf("%d", *j);

  return 0;
}
```



```
int *square3(int *p)
{
  int *result = malloc(sizeof(int));
  *result = *p * *p;
  return result;
}


int main (void)
{
  int i = 2;
  int *j = square3(&i);
  printf("%d", *j);

  return 0;
}
```

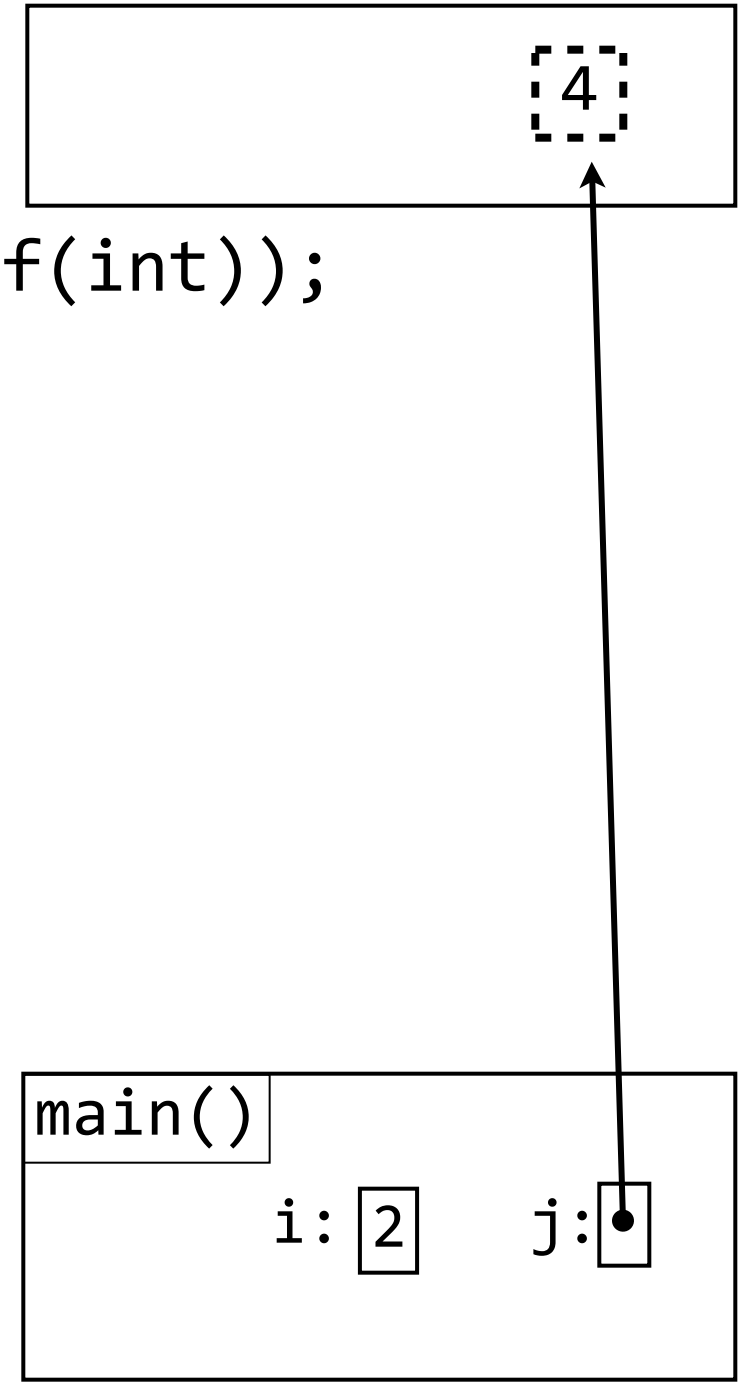


```
int *square3(int *p)
{
  int *result = malloc(sizeof(int));
  *result = *p * *p;
  return result;
}


int main (void)
{
  int i = 2;
  int *j = square3(&i);
  printf("%d", *j);

  return 0;
}
```



```
int *square3(int *p)
{
  int *result = malloc(sizeof(int));
  *result = *p * *p;
  return result;
}


int main (void)
{
  int i = 2;
  int *j = square3(&i);
  printf("%d", *j);

  return 0;
}
```

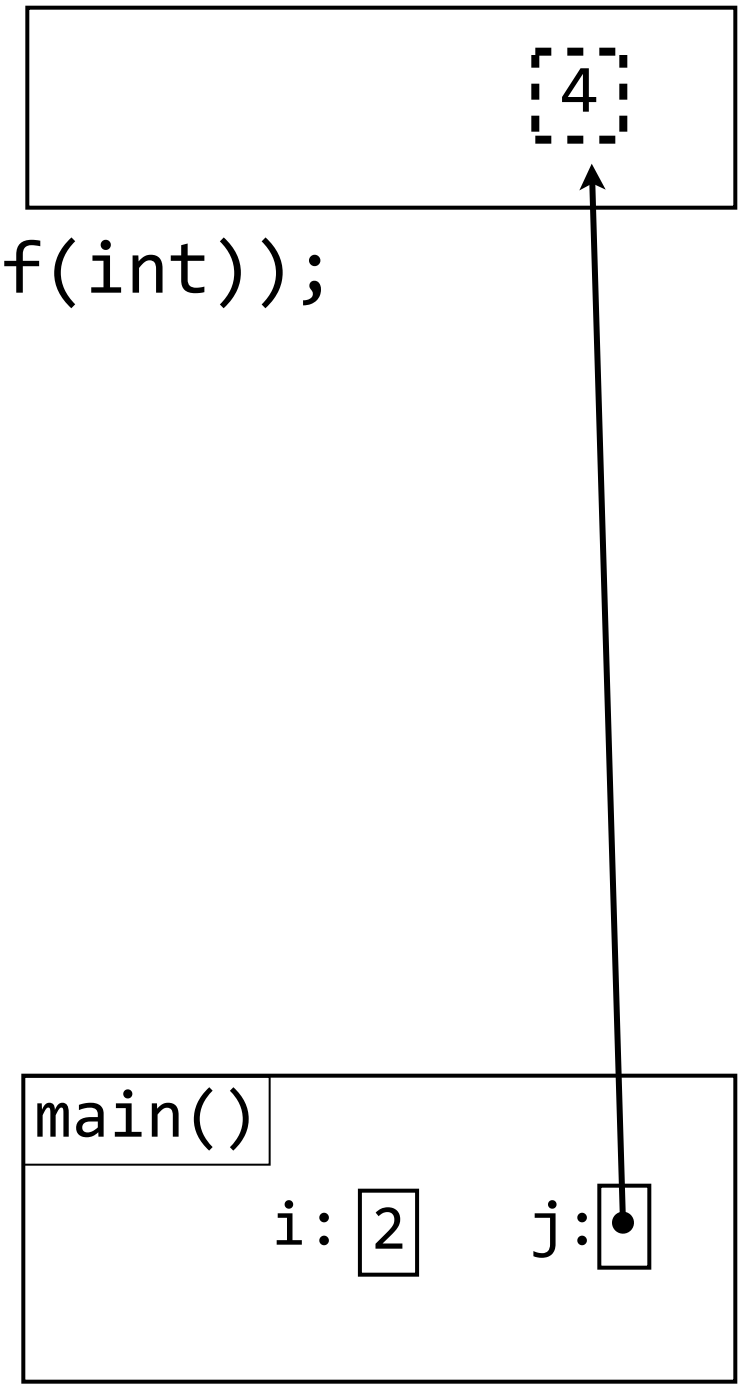
4



```
int *square3(int *p)
{
  int *result = malloc(sizeof(int));
  *result = *p * *p;
  return result;
}


int main (void)
{
  int i = 2;
  int *j = square3(&i);
  printf("%d", *j);

  return 0;
}
```

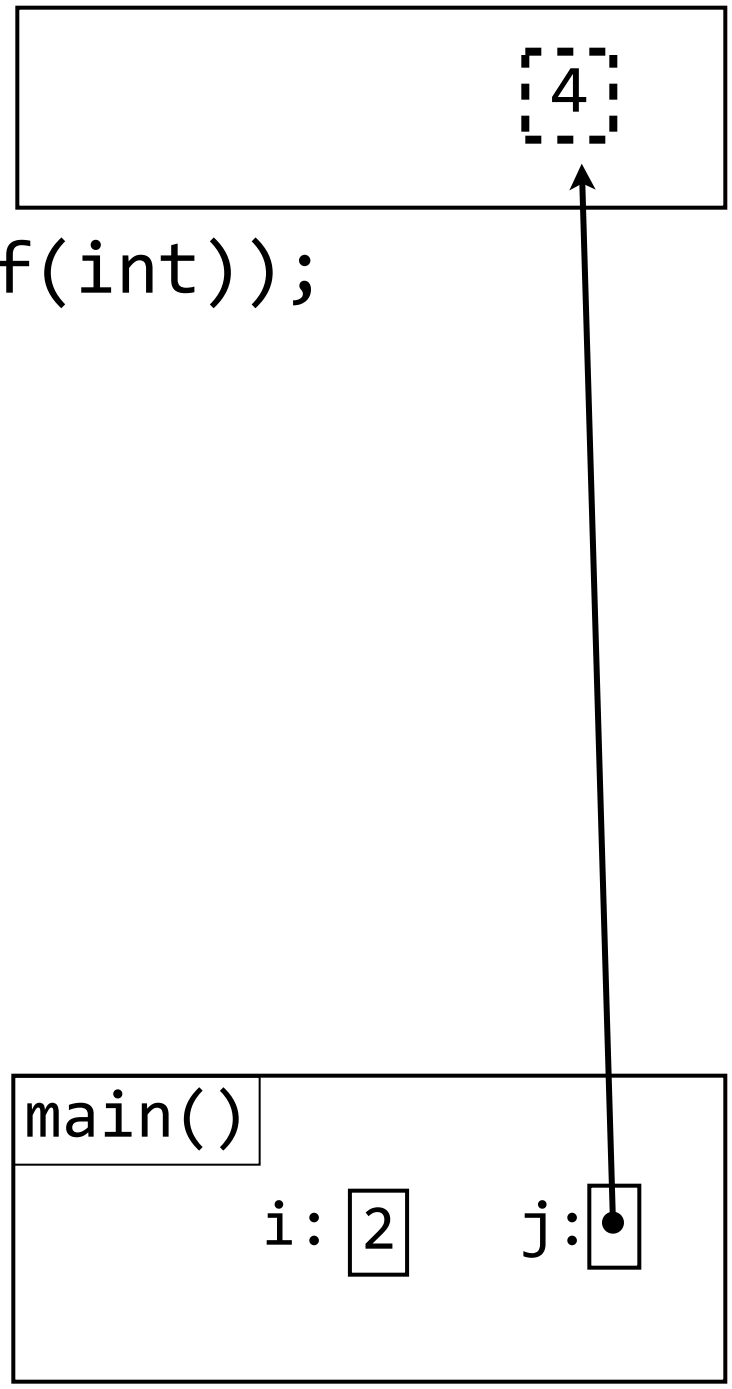
4

Stack



```
int *square3(int *p)
{
  int *result = malloc(sizeof(int));
  *result = *p * *p;
  return result;
}


int main (void)
{
  int i = 2;
  int *j = square3(&i);
  printf("%d", *j);

  return 0;
}
```

4



```
char *concat(const char *s1, const char *s2)
{
  int length = strlen(s1) + strlen(s2);

  char result[length + 1];

  strncpy(result, s1, length + 1);
  strncat(result, s2, length - strlen(result));

  return result;
}
```

```
char *s = concat("Hello", "World");

printf("<<%s>>", s);
```

```
char *concat(const char *s1, const char *s2)
{
  int length = strlen(s1) + strlen(s2);

  char result[length + 1];

  strncpy(result, s1, length + 1);
  strncat(result, s2, length - strlen(result));

  return result;
}
```

Wrong

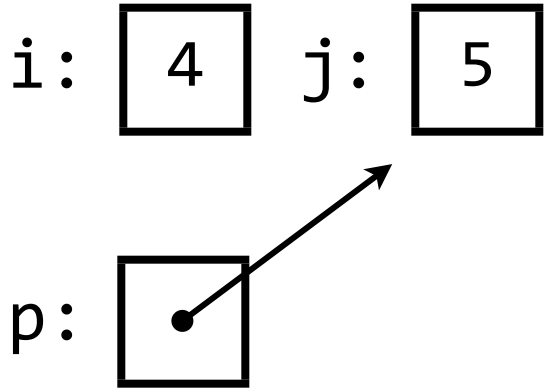
```
char *s = concat("Hello", "World");

printf("<<%s>>", s);
```

```
char *concat(const char *s1, const char *s2)
{
  int length = strlen(s1) + strlen(s2);

  char *result = malloc((length + 1) * sizeof(char));

  strncpy(result, s1, length + 1);
  strncat(result, s2, length - strlen(result));

  return result;
}
```

```
char *s = concat("Hello", "World");

printf("<<%s>>", s);
```

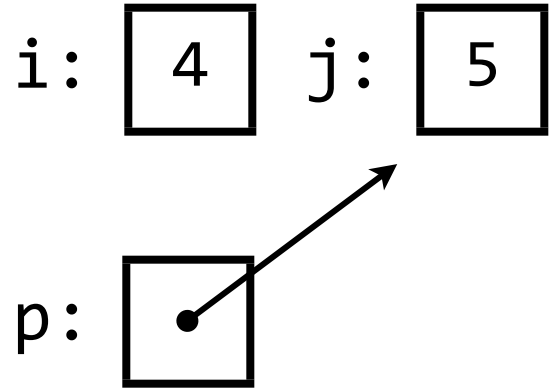
Cleaning Up

```
int i = 4;
int j = 5;
int *p = &i;

p = &j;
```

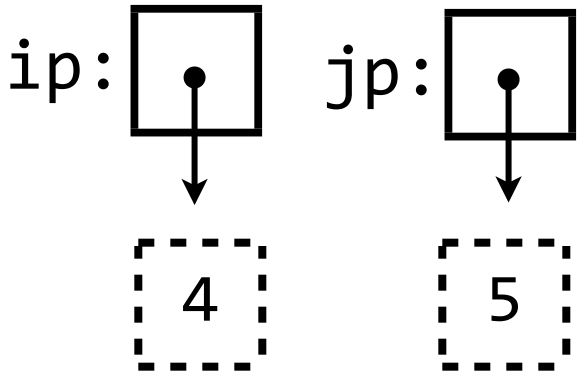


Cleaning Up

```
int i = 4;
int j = 5;
int *p = &i;

p = &j;
```

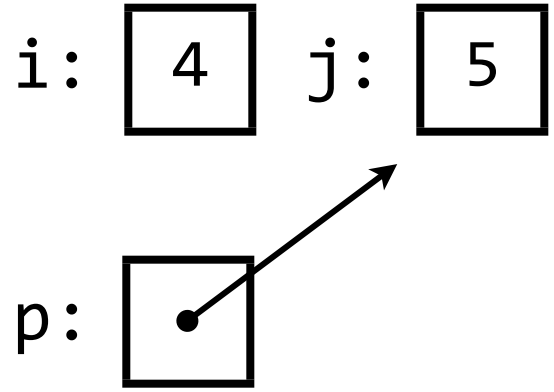



```
int *ip = malloc(sizeof(int));
int *jp = malloc(sizeof(int));
*ip = 4;
*jp = 5;
```

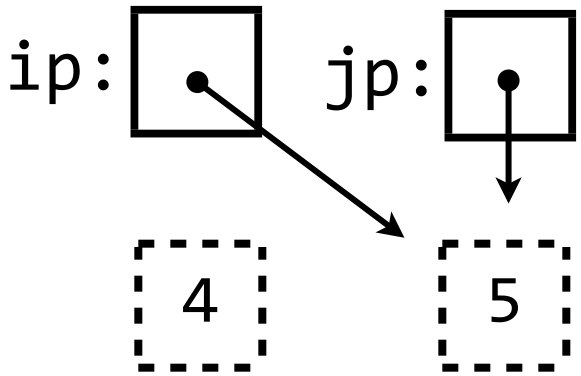


Cleaning Up

```
int i = 4;
int j = 5;
int *p = &i;

p = &j;
```

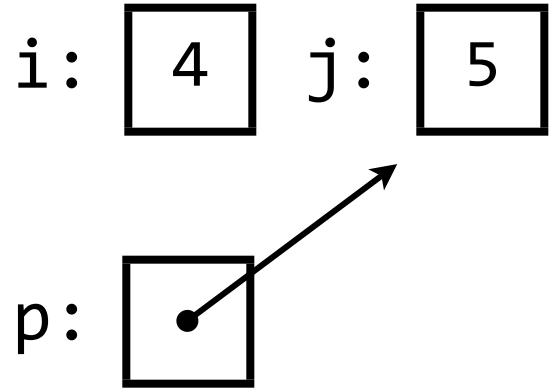



```
int *ip = malloc(sizeof(int));
int *jp = malloc(sizeof(int));
*ip = 4;
*jp = 5;

 ip = jp;
```



Cleaning Up

```
int i = 4;
int j = 5;
int *p = &i;

p = &j;
```




```
int *ip = malloc(sizeof(int));
int *jp = malloc(sizeof(int));
*ip = 4;
*jp = 5;

 ip = jp;
```

This value is lost

This bit of memory is lost