“Arrays and Pointers”

Taesoo Kwon

Hanyang University

Pointers

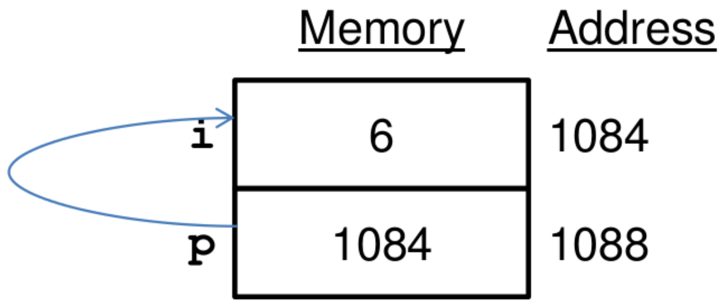
- **Pointers are variables whose value is an address**
- **Summary**
  - int x = 5;
  - int *ptr; /* pointer variable */
  - ptr = &x; /* getting address of x */
  - *ptr = 10; /* equivalent to x = 10 */
- **Let’s draw a diagram for the above**
- **Every variable is stored at an address in memory**
- **We use pointers to perform manipulation of memory, by accessing items at the address stored in the pointer**

Pointer operators

- **Obtaining the address of an object (&)**
  - Placed before a variable (or an object in memory)
- **Accessing the value at an address (*)**
  - Placed before an expression which is either a pointer or otherwise evaluates to an address

- Example:

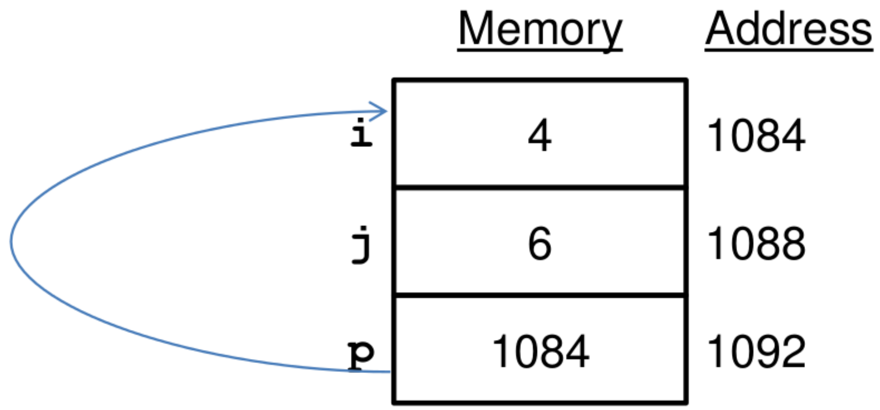
```
int i = 6;
int *p;
p = &i;
printf("%d %d\n", *p, *(&i));
```



Using a dereferenced pointer

- The `*` operator can be used on both the left and right sides of an assignment

```
int i = 6;
int j;
int *p;
p = &i;
j = *p;
printf("%d %d\n", i, j);
*p = 4;
printf("%d %d\n", i, j);
```



Multiple uses for *

- as multiplication operator
- * to declare a variable as a pointer variable

```
int *ptr;   /* we are not dereferencing here */
```

- to dereference

```
int x = 5;
int *ptr = &x;
*ptr = 10;
```

Pointers to Pointers

- You can also obtain the address of a pointer variable:

```
int i = 4;
int j = 6;
int *p = &i;
int *q = &j;
int **r = &p;
printf("%d\n", **r);
*r = &j;
printf("%d\n", *p);
```

	Memory	Address
`i`	4	1084
`j`	6	1088
`p`	1088	1092
`q`	1088	1096
`r`	1092	1100

- Let’s add some arrows to the memory map
- This technique will be useful when working with pointers as parameters

Pointers as parameters

- You can also pass addresses into a function:

```
void swap(int *a, int *b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}
...
int x = 2;
int y = 3;
swap(&x, &y);
printf("%d %d\n", x, y);
```

- Why do we need to use pointers here?
- Let's draw a memory map for the above
- What would happen if after *b = tmp we set a and b to null?