It's also possible to initialize a pointer variable at the time we declare it:

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

Q&A

We can even combine the declaration of `i` with the declaration of `p`, provided that `i` is declared first:

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

The Indirection Operator

Once a pointer variable points to an object, we can use the * (indirection) operator to access what's stored in the object. If `p` points to `i`, for example, we can print the value of `i` as follows:

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

Q&A

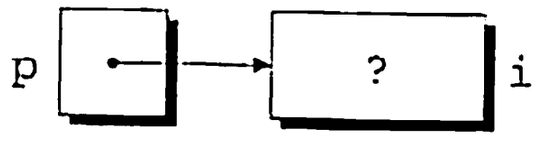
`printf` will display the *value* of `i`, not the *address* of `i`.

The mathematically inclined reader may wish to think of * as the inverse of &. Applying & to a variable produces a pointer to the variable; applying * to the pointer takes us back to the original variable:

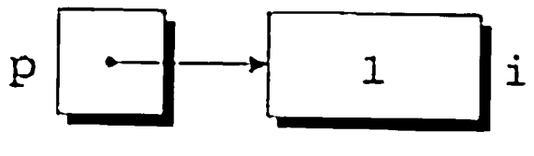
```
j = *&i;    /* same as j = i; */
```

As long as `p` points to `i`, `*p` is an ***alias*** for `i`. Not only does `*p` have the same value as `i`, but changing the value of `*p` also changes the value of `i`. (`*p` is an lvalue, so assignment to it is legal.) The following example illustrates the equivalence of `*p` and `i`; diagrams show the values of `p` and `i` at various points in the computation.

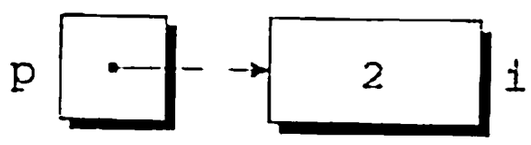
```
p = &i;
```




```
i = 1;
```




```
printf("%d\n", i);     /* prints 1 */
printf("%d\n", *p);    /* prints 1 */
*p = 2;
```




```
printf("%d\n", i);     /* prints 2 */
printf("%d\n", *p);    /* prints 2 */
```