Address of Operator (&)

- The ampersand, &, is called the ***address of operator***
- The address of operator is a unary operator that returns the ***address of its operand***

- ***Binary operator***

  LHS = RHS, cout << "hello", cin >> a, +, -, *, /, %

- ***Unary operator***

  Only need one side

  &RHS; &a

Dereferencing Operator (*)

- When used as a unary operator, * is the dereferencing operator or indirection operator
  - Refers to object to which its operand points
- Example:

declare a pointer

value

```
int x = 25;
int *p;
p = &x;      //store the addr
```

*p ✓ 153

- To print the value of x, using p:

```
cout << *p << endl;
```

25

- To store a value in x, using p:

```
*p = 55;
```

✓

(1). what this pointer p points to?

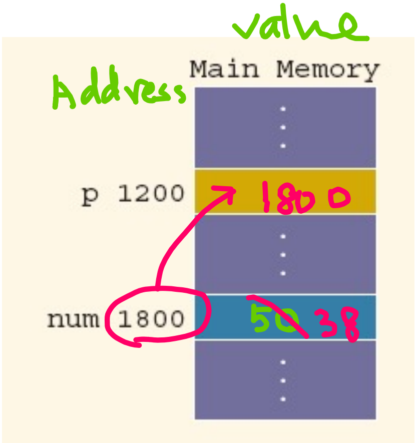
(2) what's val of x ? 25

(3) *p is 25

✓ x=55;

Variable name	address	value
x	153	25 55
p (pointer)	1008	53 >

Exercise



- Assuming the memory layout provided, after this code executes:

```
int num; // declare an integer variable
int *p; // declare a pointer named: p
num = 50; // assign 50 to variable num
p = &num;
*p = 38;
num = 38;

cout << p; // 1800
cout << *p;
// variable num; *p the value of this variable: 38
```

value of p

1800

1) what variable p points to? num

(2) *p = value of this value num

- What are the values of these expressions?

```
&num // 1800;
num // num: value of this one ⇔ 38
&p    // 1200
p // 1800
*p // 38
```

Assigning Pointers

- Pointers can be assigned to pointers of the same type

```
int x, *p, *q; // variable x, two pointers: p, q
x = 50;
p = &x;
q = p; // q is a pointer; assign the value of p to the value
    of q ⇔ both pointers p and q are assigned to variable x.
```

value of p

value of q

- The value of *q is?

(1) variable : x

(2) *q : value of x = 50

*q: the value that the pointer q pointed to

So *q is 50

count << *q;

// 50

Variable name	address	value
x	153	50
p (pointer)	1008	153
q (pointer)	17	153

Assigning Pointers

- Pointers can be assigned to pointers of the same type

```
int x, *p, *q;
x = 50;
p = &x;
q = p;
```

- The value of *q is 50

The Null Pointer

- In addition to variable addresses and other pointers, a pointer can be assigned to the *null pointer*
  - Either the number `0` or the constant `NULL`
  - Used to indicate an invalid pointer (pointing to nothing)
  - Dereferencing a null pointer causes ***a hard error***

```
int *p = 0;
p = NULL;
*p //dereferencing
```

```
int num, *q;
q = &num;
```

Comparing Pointers

- Be careful of the difference between comparing two pointers and comparing their values:

```
int x = 50, y = 50, *p, *q;
p = &x; // assign the pointer p to the variable x
q = &y;
```

50

50

(1) variable : x

(2) *p : value of x = 50

- `*q == *p` evaluates to?

True

- `q == p` evaluates to?

False

value of p ⇔ address of x

value of q ⇔ &y

Comparing Pointers

- Be careful of the difference between comparing two pointers and comparing their values:

```
int x = 50, y = 50, *p, *q;
p = &x;
q = &y;
```

- `*q == *p` evaluates to `true`
- `q == p` evaluates to `false`

Pointers and Arrays

```
int a[20] = {1,2,3,4};
int *p; // declare a pointer named p

p = a; // the reference/address of the index 0.
```

```
int num = 78;
int *p;
p = &num;
// variable pass by values;
```

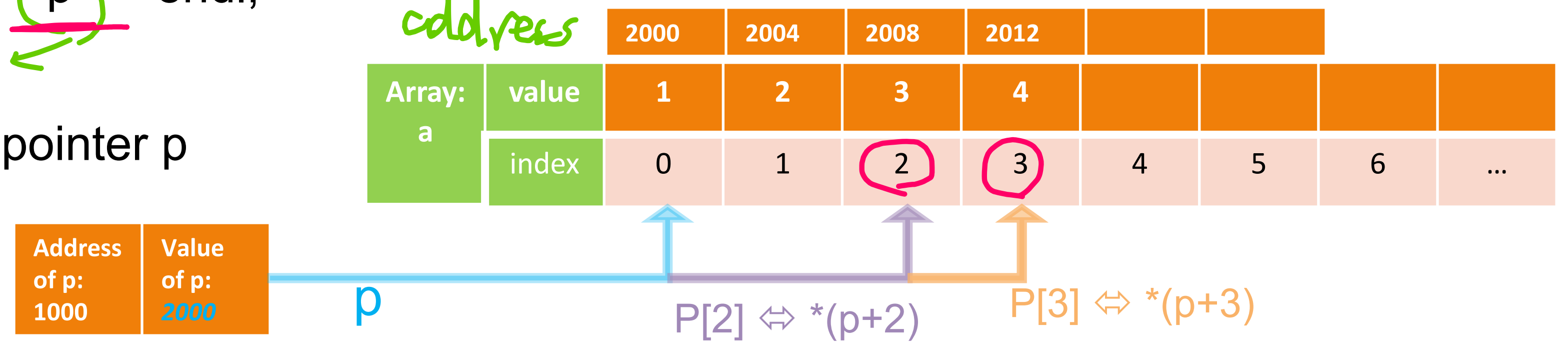
```
cout << a[0] << endl; // a[0]: the value of the 1st element: 1
cout << a[1] << endl; // a[1]: the value of the 2nd element: 2

cout << p[2] << endl; // 2  p points to element with index 0: p[2] ⇔ *(p + 2)
cout << p[3] << endl; // 4
```

vari/element : index: *p : value of this element = 1

```
cout << *p << endl;  //  1
cout << *(p+2) << endl; // 3
cout << *(p+3) << endl; // 4
cout << p << endl;
```

value of p



pointer p

		address							
		2000	2004	2008	2012				
Array: a	value	1	2	3	4				
	index	0	1	2	3	4	5	6	…

Address of p: 1000	Value of p: 2000