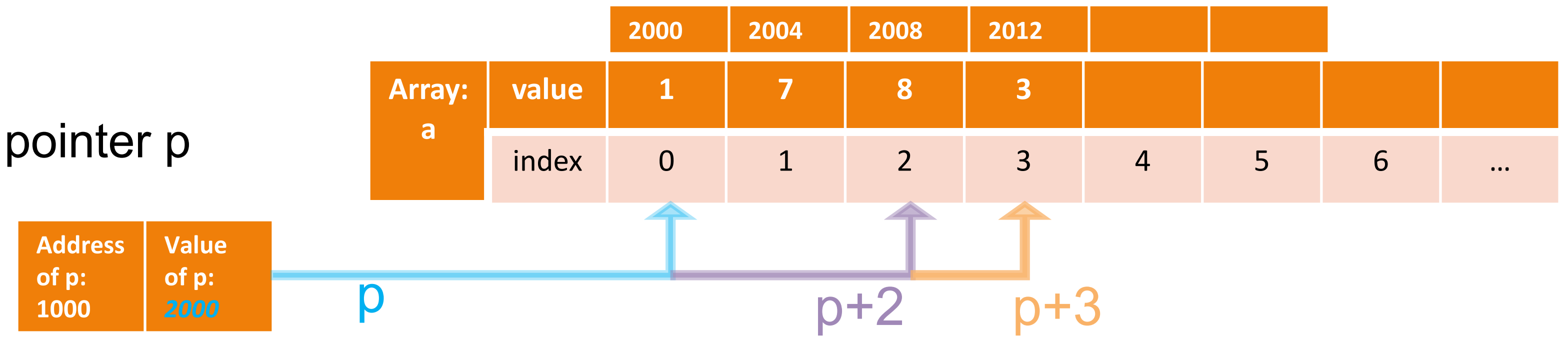
Pointers and Arrays

```
int a[20] = {1, 7, 8, 3};
int *p; // declare a pointer named p

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

cout << a[0] << endl; // 1
cout << a[1] << endl; // 7

cout << p[2] << endl; // 8; p[2] ⇔ a[2]
cout << p[3] << endl; // 3; p[3] ⇔ a[3]

cout << *p << endl; // 1
cout << *(p+2) << endl; //8; *(p+2) ⇔ a[2]
cout << *(p+3) << endl; // 3
```

```
int num = 78;
int *p;
p = &num;
```



		2000	2004	2008	2012				
Array: a	value	1	7	8	3				
	index	0	1	2	3	4	5	6	...

Variables, Memory and Pointers

- A variable is a named piece of memory
  - The name stands in for the *memory address*

```
int num;//allocate memory to it first
num = 10;
```

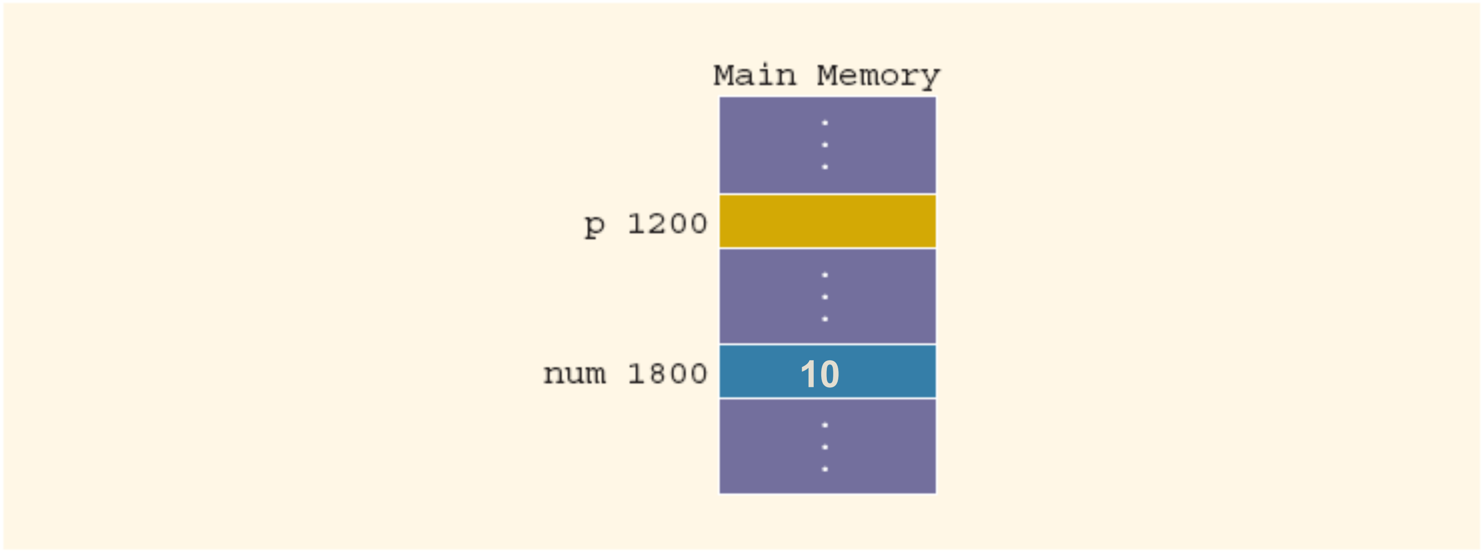


FIGURE 13-1 Main memory, p, and num

Variables, Memory and Pointers

- When a value is assigned to a variable, it is stored at that address in memory

```
num = 78;
```

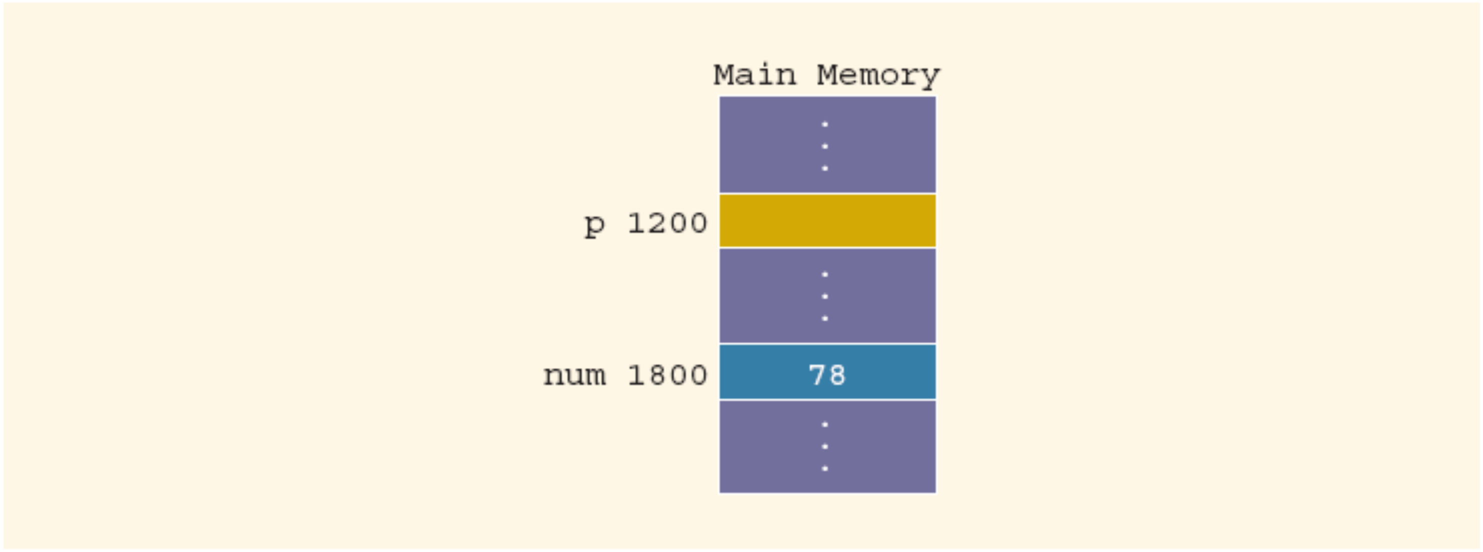


FIGURE 13-2 num after the statement num = 78; executes

Variables, Memory and Pointers

- A *pointer* is a variable that holds the address of another variable
  - It is declared in terms of the type of variable it points at:

```
int *p; // given a * in front of a variable, it means
  that this variable is a pointer.
```

- int num; num = 78;

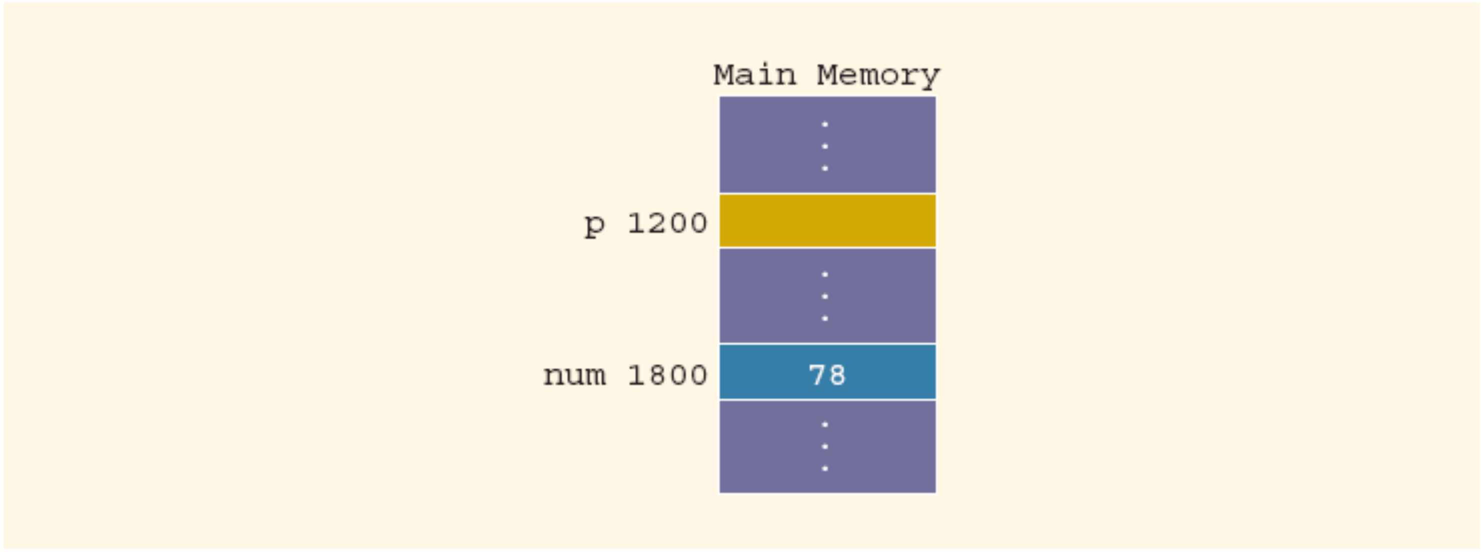


FIGURE 13-2 num after the statement num = 78; executes

Variables, Memory and Pointers

- The operator `&` returns the address of a variable
  - It can then be assigned to a pointer

```
p = &num;
// &num => the address of the variable num ⇔ 1800
// assign the address of num to the value of p.
```

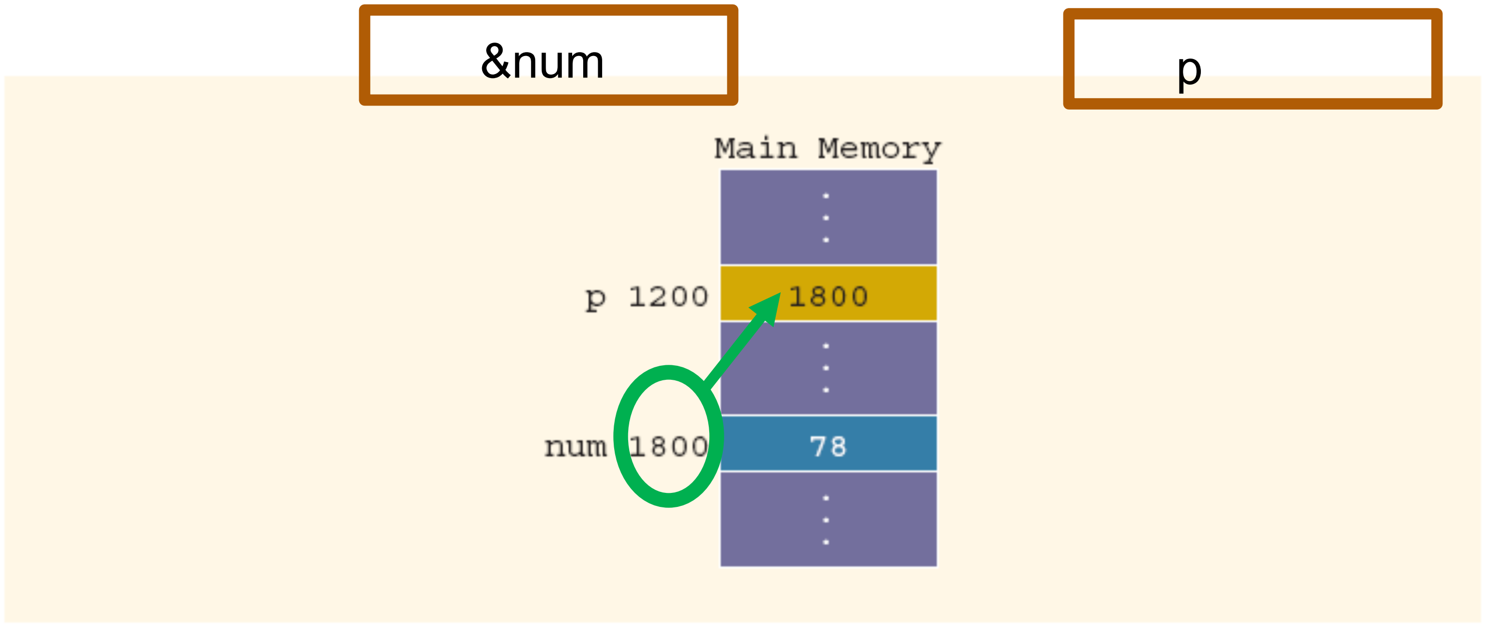


FIGURE 13-3 p after the statement p = &num; executes

Variables, Memory and Pointers

- The operator * takes an address (a pointer) and returns the location in memory being pointed to
  - Can only be applied to a pointer

```
*p = 24;
int *q; // define a pointer;
*q = 30; // assign 30 to the variable that the pointer
  pointed to.
```

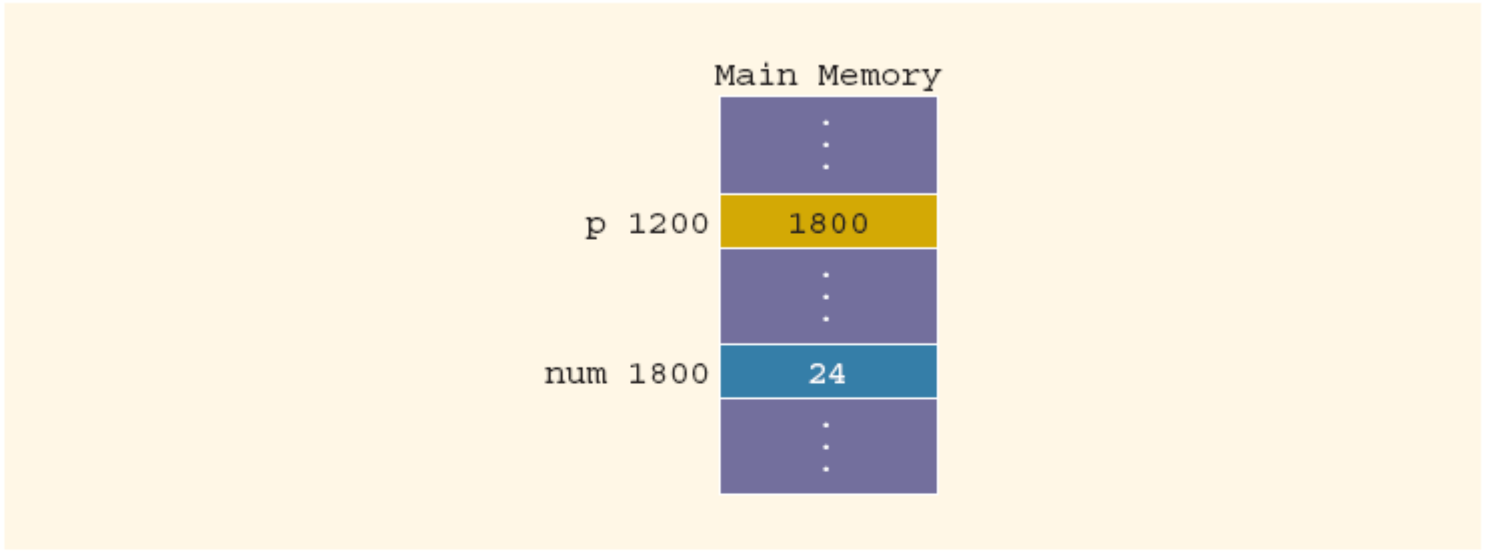


FIGURE 13-4 *p and num after the statement *p = 24; executes

Declaring Pointer Variables

- Syntax:

```
dataType *identifier;
```

- Examples:

```
int *p;
char *ch;
```

- These statements are equivalent:

```
int  *p;
int*  p;
int * p;
```

Declaring Pointer Variables (continued)

- In the statement:

  ```
  int* p, q; // p is a pointer; q is variable
  int num1, num2; ⇔ int num1; int num2;
  ```

  only `p` is the pointer variable, not `q`; here `q` is an `int` variable

- To avoid confusion, attach the character `*` to the variable name:

  ```
  int    *p, q;
  int    *p, *q;
  int array1[100], array2[20];
  ```

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***

Dereferencing Operator (*)

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

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

  - To print the value of `x`, using `p`:

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

  - To store a value in `x`, using `p`:

```
*p = 55;
```

Variable name	address	value
x	153	55
p (pointer)	1008	***153***

Exercise

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

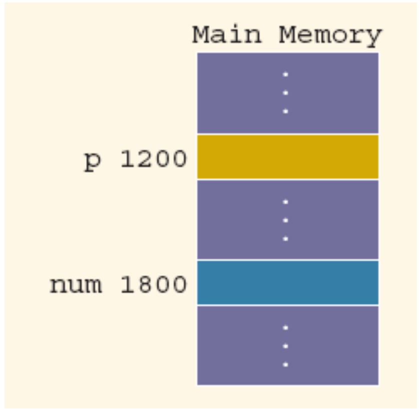
```
int num; // declare an integer variable
int *p; // declare a pointer named: p
num = 50; // assign 50 to variable num
```

p = &num;

//1. assign a pointer p to the variable num;

//2. assign the address of num (1800) to the value of the pointer p

```
*p = 38;
//assign 38 to *p (the value of the pointer pointed to)
// the value of num = 38
```



- What are the values of these expressions?

```
&num = 1800; // &num: the address of the variable num
num = ? // value of num = 38
&p = ? // &: address of something; &p: the address of the pointer p 1200
p = ? // the value of p ⇔ the address that the pointer pointed to, 1800.
*p = ? // *p: the value of the pointer pointed to, 38.
```

Assigning Pointers

- Pointers can be assigned to pointers of the same type

```
int x, *p, *q; //declare one variable x, and 2 pointers p, q
x = 50; // the value of x = 50
p = &x;
//1. a pointer p points to the variable x
//2. assign the address of x to the value of the pointer p
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.
```

- The value of *q is?

*q: the value that the pointer q pointed to

So *q is 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
```

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?
- `q == p` evaluates to?

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 Class

```
class A
{
public:
        char a, b, c;
        int r[7];
};
```

Class	A		
	Public variables:		
		char	a = ‘7’
		char	b = ‘a’
		char	c = ‘a’
		int	r[7]

value						5
index	0	1	..	...	..	6

Pointers and Class

***A** x;* // declare an object named x with the type of the class A.

int num;

```
x.a = '7';
x.b = 'b';
x.c = 'a';
```

num = 78;

```
A *p; // declare a pointer named p with the type A.
p = &x;
```

p = &num;

```
(*p).a = '8';
(*p).b = 'b';
p->b = 'a'; // 1. p is a pointer;
               // 2. p points to an object;
               // 3. one element of this object is b;
               // 4. we are updated the value of the element b for
this object ( the object that the pointer p points to).
```

y.a = '9';
y.b = 'b';

```
p->r[6] = 5;
cout << x.r[6] << endl;
```