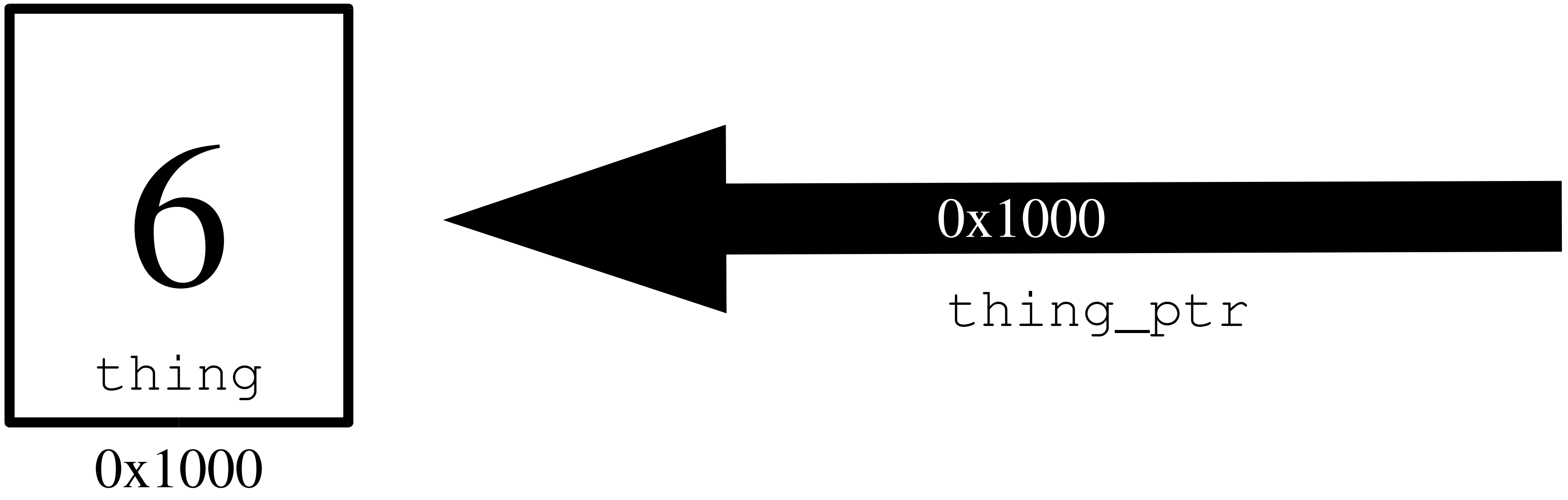
Chapter - 15
Simple Pointers

Things and Pointers to Things

There are things

and pointers to things



A Small Town

Service (Variable Name)	Address (Address value)	Building (Thing)
Fire Department	1 Main Street	City Hall
Police Station	1 Main Street	City Hall
Planning office	1 Main Street	City Hall
Gas Station	2 Main Street	Ed’s Gas Station

Pointer Operators

A pointer is declared by putting an asterisk (*) in front of the variable name in the declaration statement:

```
int thing;         // define "thing"
int *thing_ptr;  // define "pointer to a thing"
```

Pointer operations:

Operator	**Meaning**
*	*Dereference* (given a pointer, get the thing referenced)
&	*Address of* (given a thing, point to it).

Things and pointers to things

`Thing` A thing.

`thing = 4;`

`&thing` A pointer to thing. thing is an object. The & (address of) operator gets the address of an object (a pointers), so &thing is a pointer.

Example:

```
thing_ptr = &thing; // Point to the thing
*thing_ptr = 5;     // Set "thing" to 5
```

`thing_ptr`

Thing pointer.

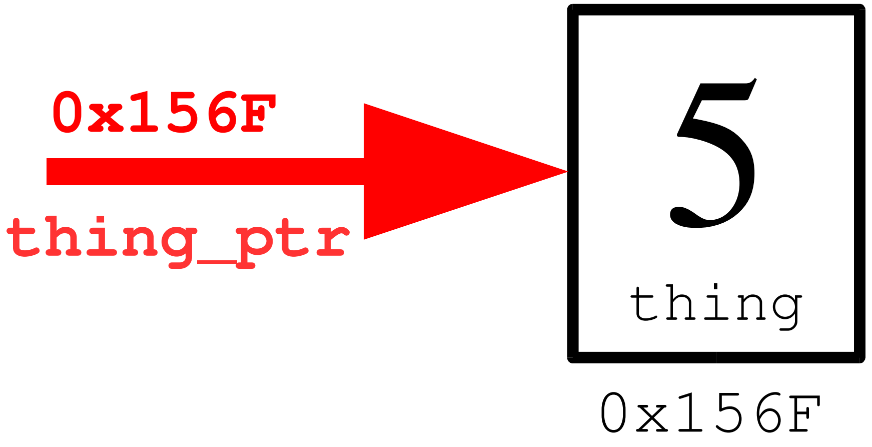
`*thing_ptr`

A thing.

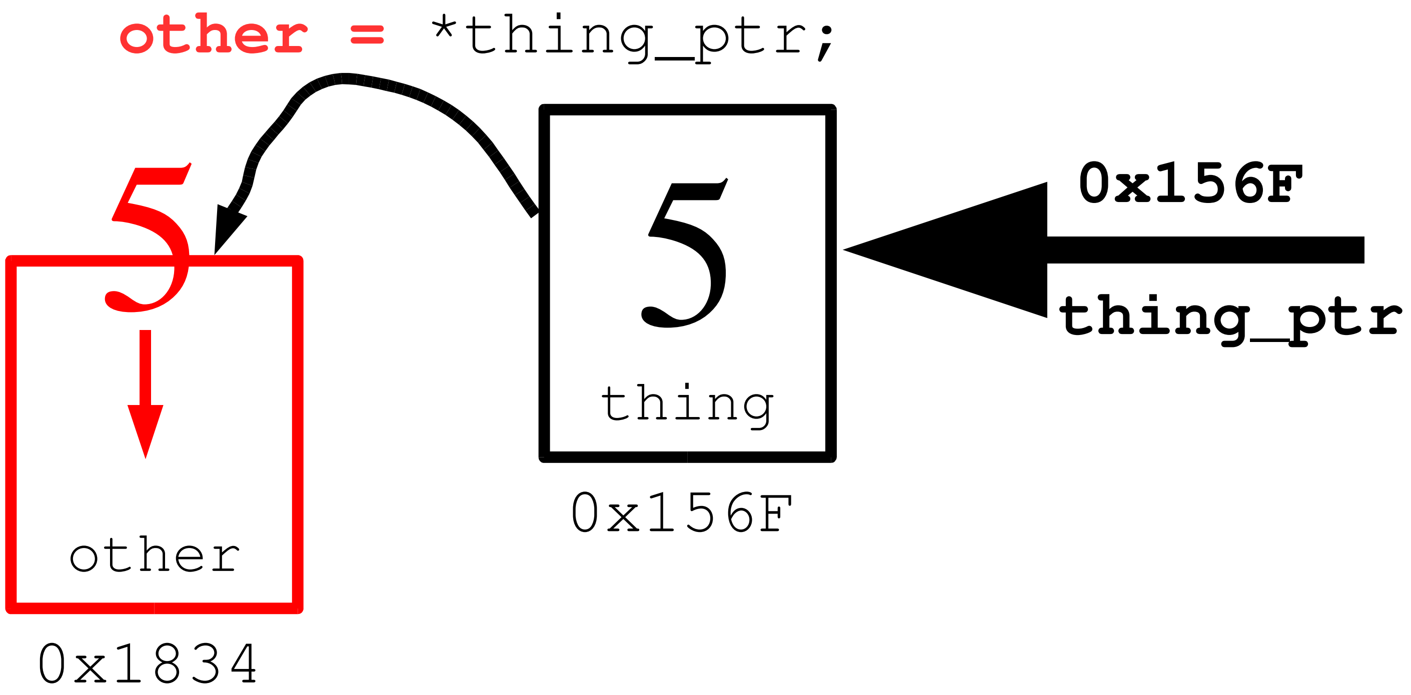
```
thing_ptr = 5;    // Assign 5 to an integer
                  // We may or may not be
                     // pointing to the specific
                     // integer "thing"
```

Make "thing_ptr" point to "thing"

```
thing_ptr = &thing;
```

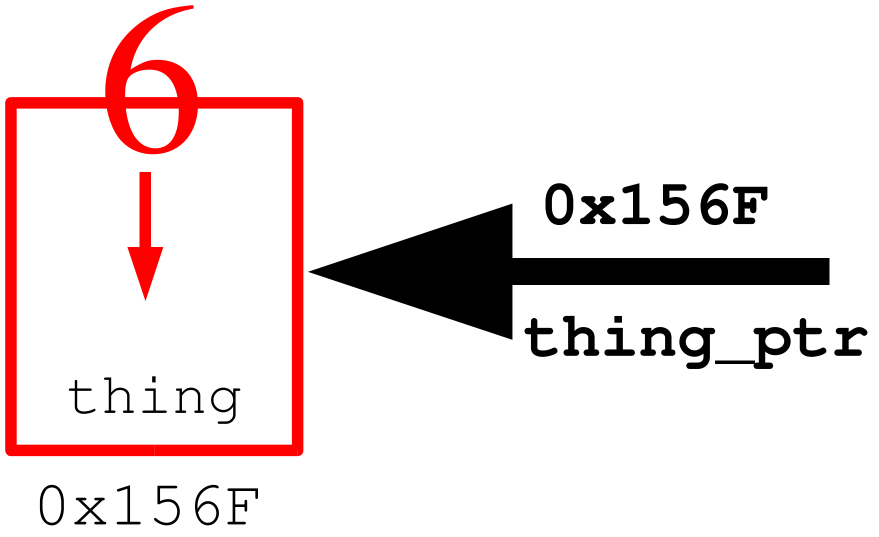


Copy data from thing pointed to by "thing_ptr" into "other"



Setting the item pointed to by "thing_ptr" to the value 6.

```
*thing_ptr = 6;
```



How not to use pointer operators

`*thing`
Illegal. Asks C++ to get the object pointed to by the variable `thing`. Since `thing` is not a pointer, this is an invalid operation.

`&thing_ptr`
Legal, but strange. `thing_ptr` is a pointer. The & (address of) operator gets a pointer to the object (in this case `thing_ptr`). Result is pointer to a pointer. (Pointers to pointers do occur in more complex programs.)

Pointer Usage

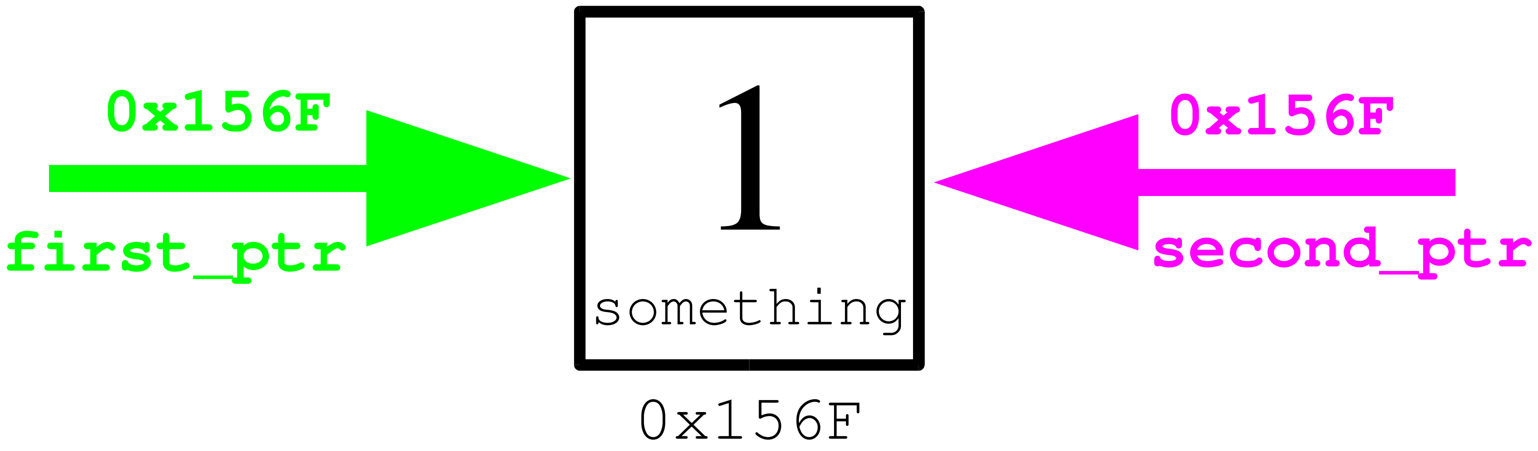
```
main()
{






}
```

Two pointers, one thing

2:

5:

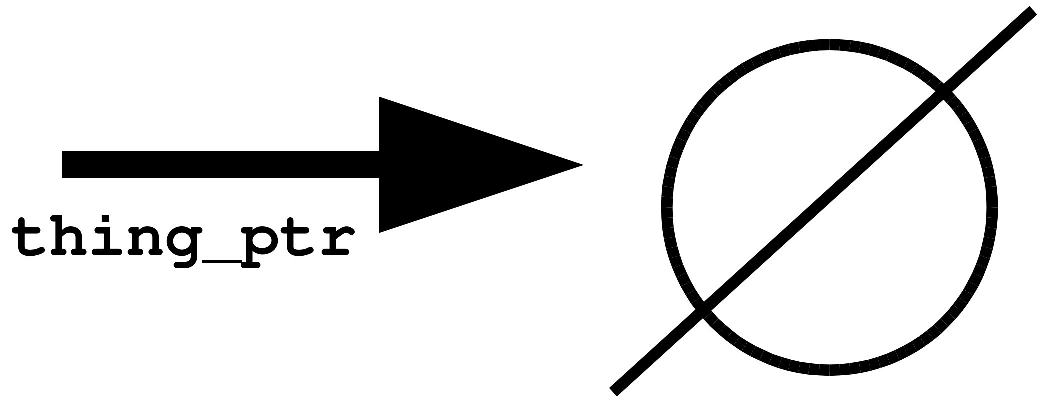
7:



Null Pointer

The null pointer points to nothing.

```
thing_ptr = NULL;
```



const Pointers

There are several flavors of constant pointers. It’s important to know what the *const* apples to.

```
const char* first_ptr = "Forty-Two";
first_ptr = "Fifty six";                    // Legal or Illegal
*first_ptr = 'X';                           // Legal or Illegal

char* const second_ptr = "Forty-Two";
second_ptr = "Fifty six";                   // Legal or Illegal
*second_ptr = 'X';                          // Legal or Illegal

const char* const third_ptr = "Forty-Two";
third_ptr = "Fifty six";                    // Legal or Illegal
*third_ptr = 'X';                           // Legal or Illegal
```

Pointers and Printing

Example:

```
std::cout << "Integer pointer " << int_ptr << '\n';
```

outputs:

```
          Integer pointer 0x58239A
```

Example:

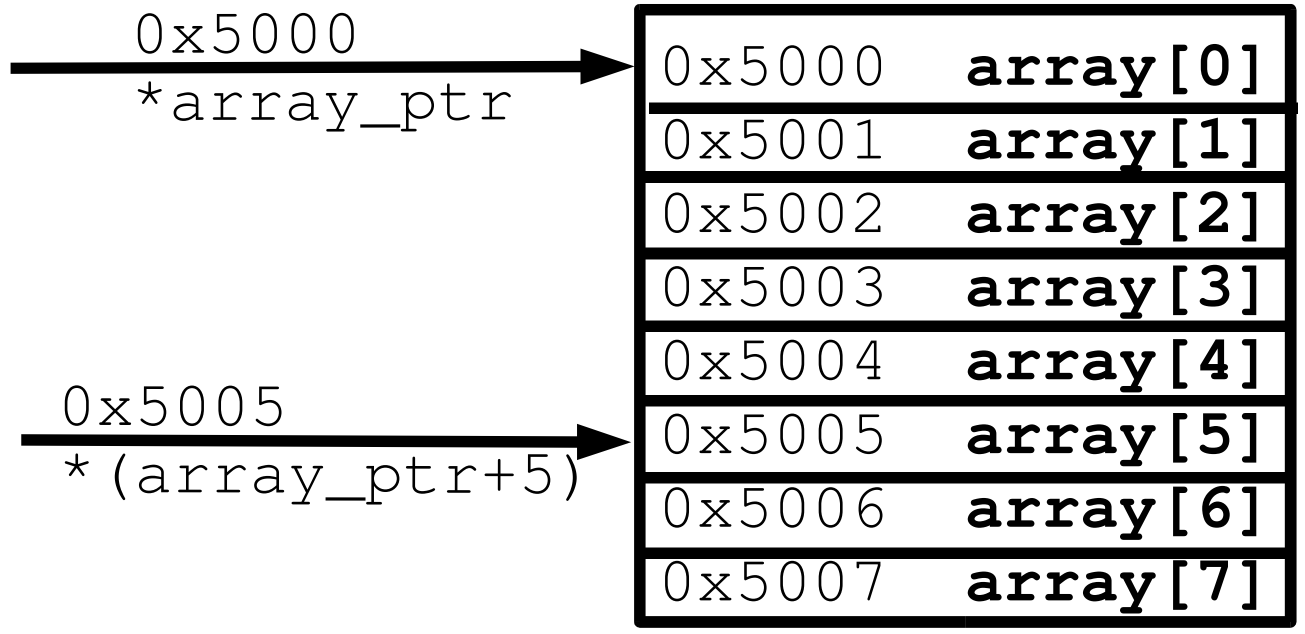
```
    // A Simple set of characters
    char some_characters[10] = "Hello";
    // Pointer to a character


std::cout << "String pointer " << char_ptr << '\n';
```

outputs

```
          String pointer Hello
```

Pointers and Arrays

```
char array[10];
char *array_ptr = &array[0];
```



[illegible]

Example

```
int main()
{









          }

}
```

Output

Array Shorthand

```
array_ptr = &array[0];
```

is the same as:

```
array_ptr = array;
```

Summing an Array (Index Version)

```
int main()
{

        ++index;



}
```

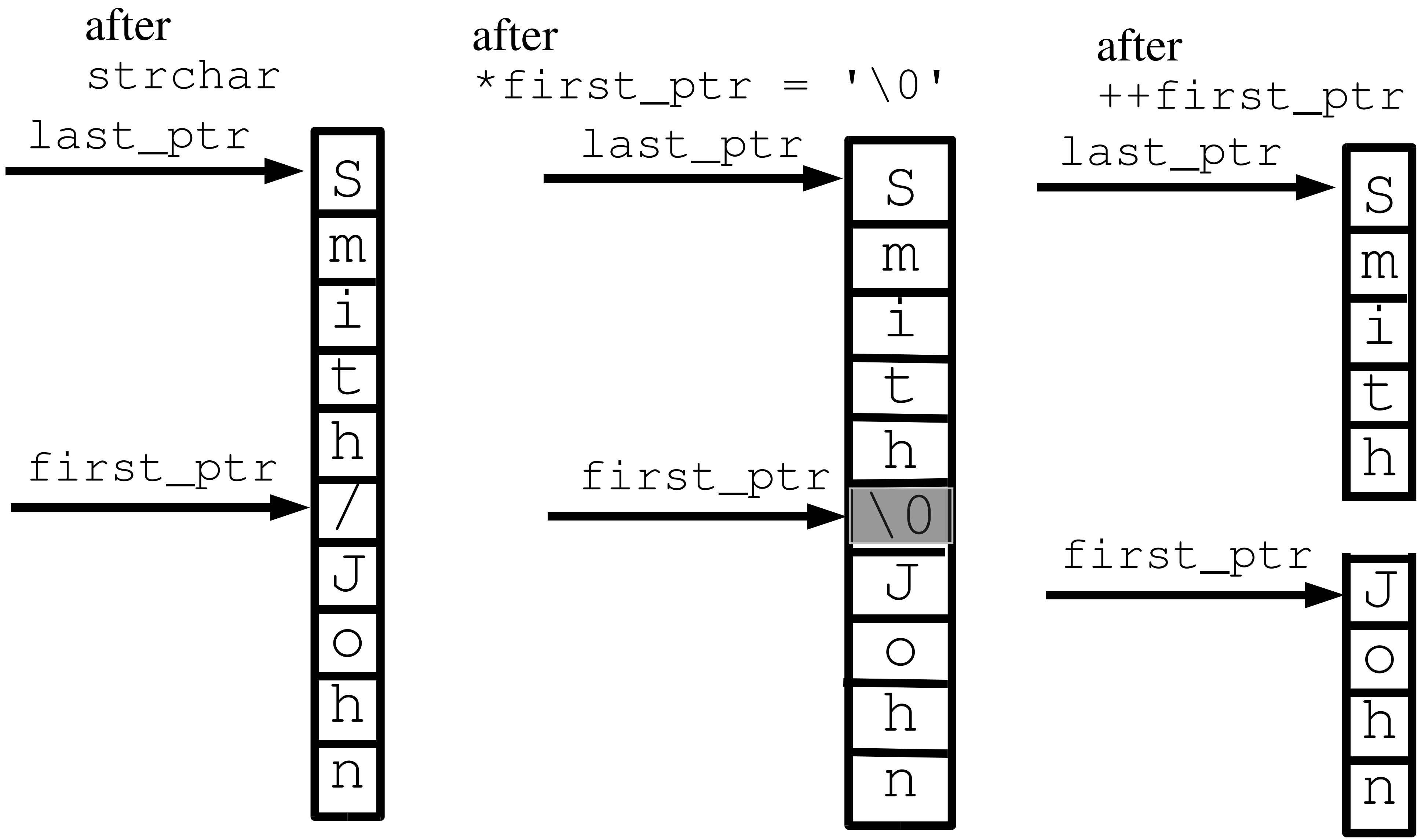
Same Program Using Pointers

```
main()
{
          ++array_ptr;
}
```

Zeroing an array

```
{

}

{

}
int main()
{

    init_array_1(array);

    init_array_1(&array[0]);

    init_array_2(array);

}
```

Splitting a C style string



[illegible]

Splitting a string

```
main() {









                   '\n';

    }






}








        ++string_ptr;
    }

}
```

Question: Why does this program print garbage?

```
/*********************************************************





 *********************************************************/
{








    return(name);
}

{

    return(0);
}
```

Pointers and Structures

```
// ....
```

Command Line Arguments

```
int main(int argc, char *argv[])
{
```

`argc` The number of arguments (program counts as one, so this number is always >= 1).
`argv` The arguments (program name is `argv[0]`).

Example:

```
args this is a test
```

turns into:

```
argc       = 5
argv[0]    = "args"
argv[1]    = "this"
argv[2]    = "is"
argv[3]    = "a"
argv[4]    = "test"
```

Example

Our mission is to make the following program:

```
print_file [-v] [-l<length>]
         [-o<name>] [file1] [file2] ...
```

`-v` Verbose option. Turns on a lot of progress information messages.

`-l<`*`length`*`>`

Set the page size to <*length*> lines. (Default = 66).

`-o<`*`name`*`>`

Set the output file to <*name*>. (Default = print.out)

[illegible]

print_file

```
/********************************************************
 ********************************************************/





/********************************************************



 ********************************************************/




}
```

[illegible]

print_file (cont)

```
/********************************************************

 ********************************************************/
{




}
```

print_file (cont)

```
{





     */

        /*

         */

            /*

             */

                break;
```

print_file (cont)

```
/*





      */


         break;
     /*

      */



         break;
     default:

         usage();
 }
```

print_file (cont)

```
        /*


         */
        ++argv;
        --argc;
    }

    /*




     */

        do_file("print.in");


          do_file(argv[1]);
          ++argv;
          --argc;
        }
    }

}
```