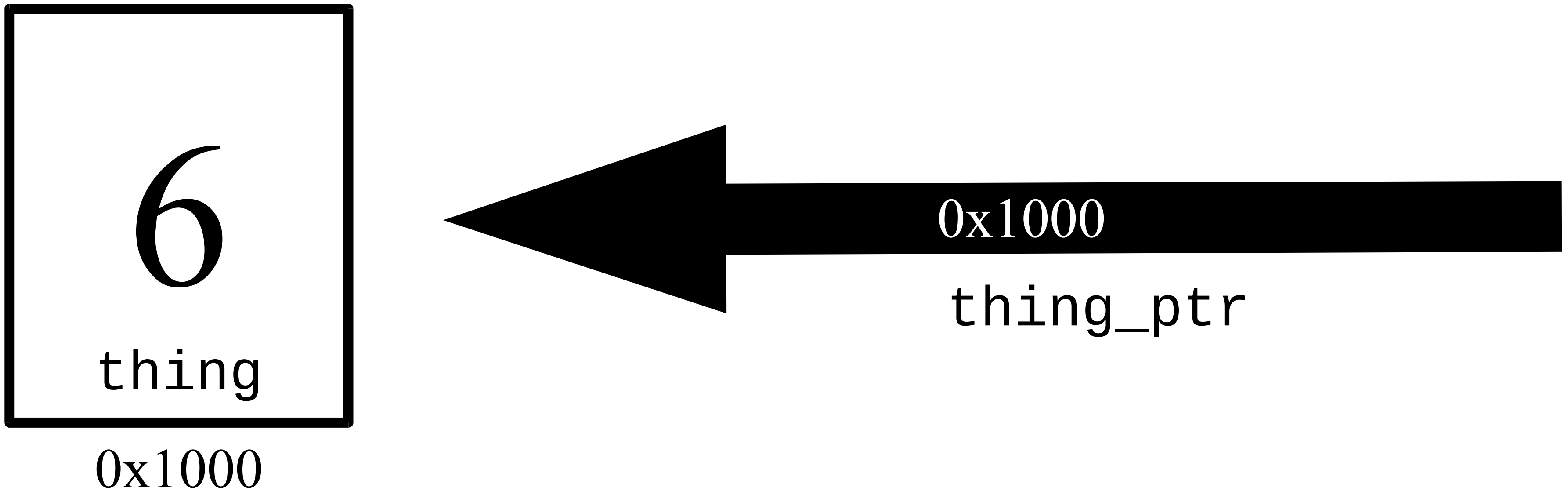
Chapter - 15
Simple Pointers

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
```

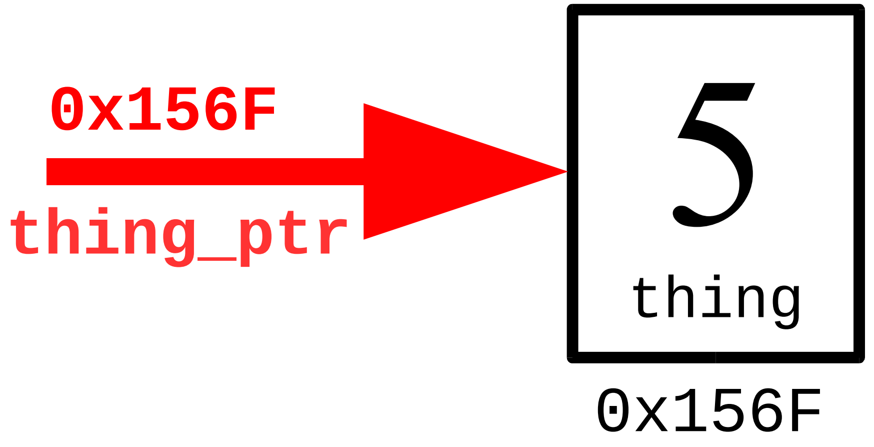
`thing_ptr` Thing pointer.

`*thing_ptr` A thing.

```
thing_ptr = 5;    // Assign 5 to an integer

                  // We may or may not be
                  // pointing to the specific
                  // integer "thing"
```

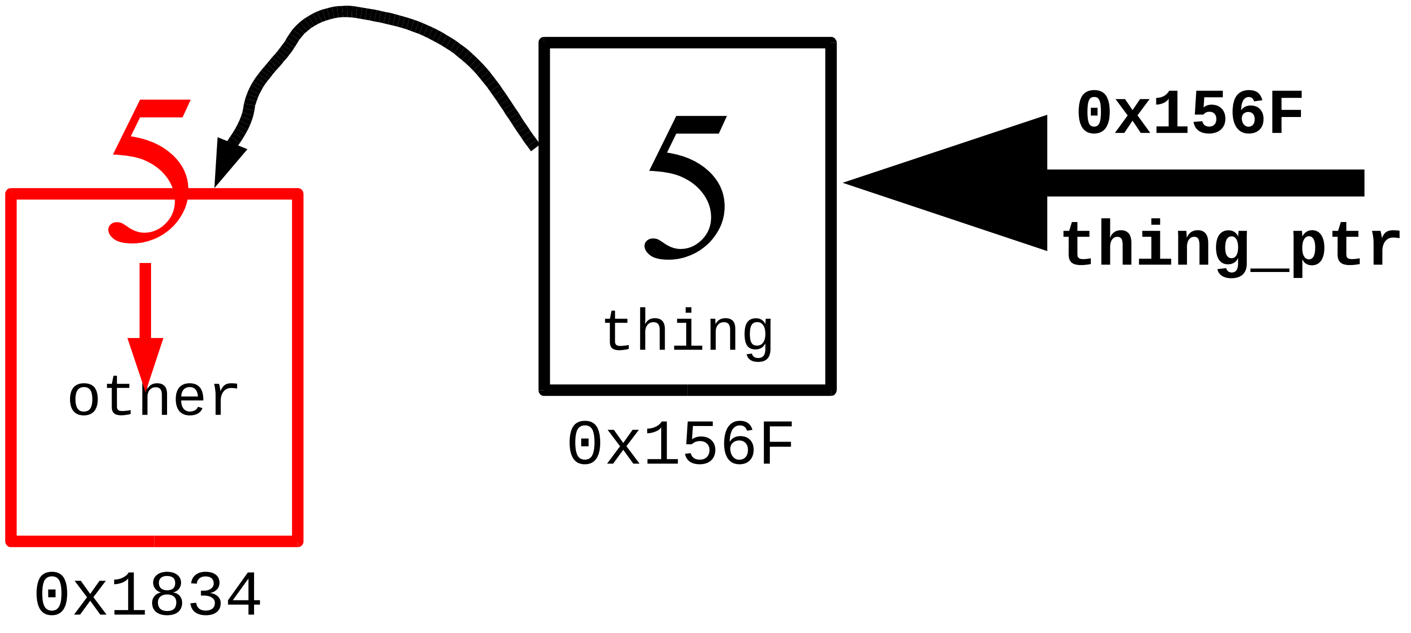
Make "thing_ptr" point to "thing"

```
thing_ptr = &thing;
```



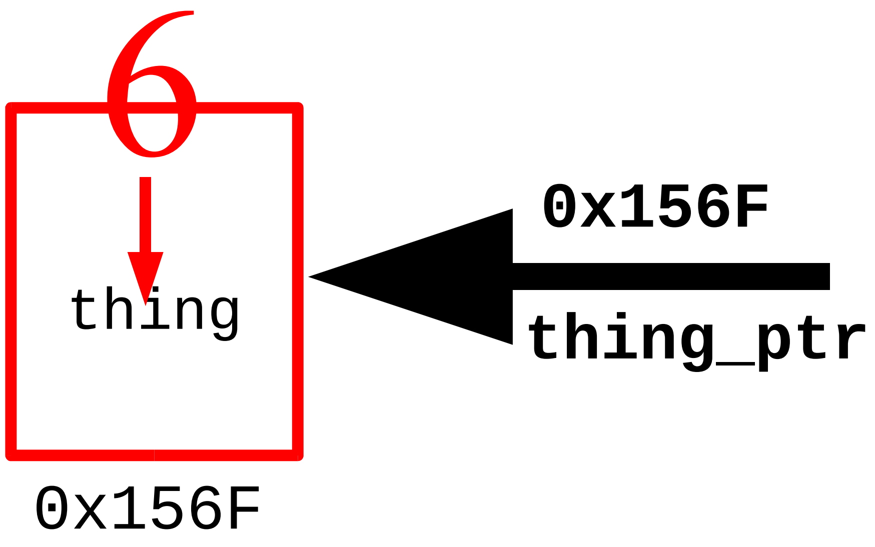
Copy data from thing pointed to by "thing_ptr" into "other"

```
other = *thing_ptr;
```



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`

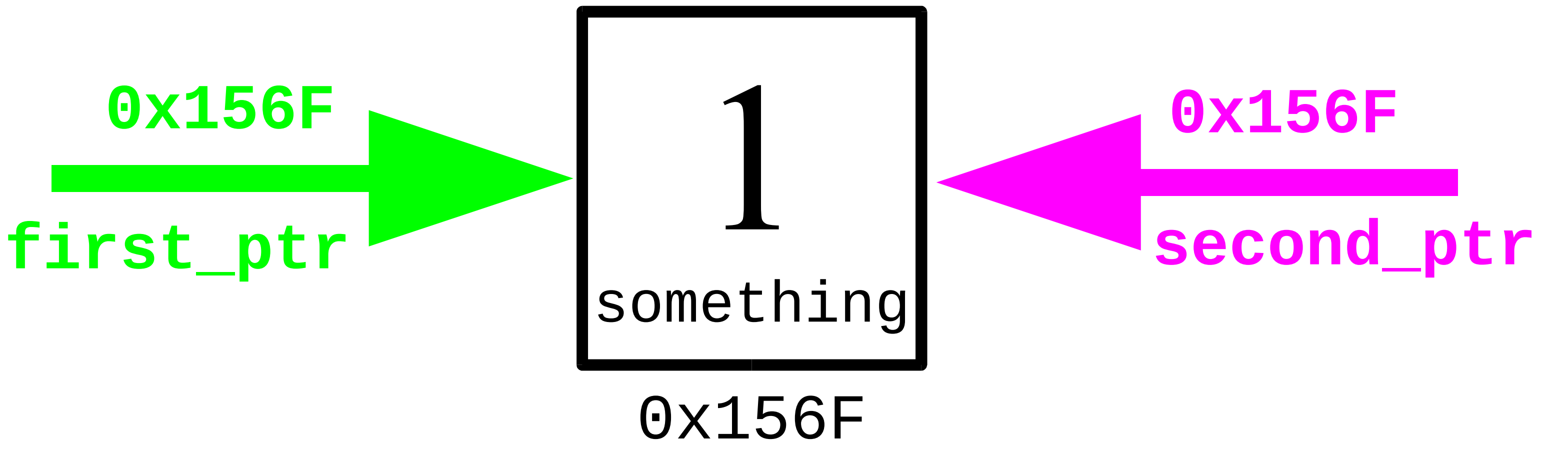
Legal, but strange. `thing_ptr` is a pointer. The & (address of) operator gets a pointer to the object (in this case `thing_ptr`). Result is pointer to a pointer. (Pointers to pointers do occur in more complex programs.)

Pointer Usage

```
#include <iostream>
main()
{
    int   thing_var;  // define a variable
    int  *thing_ptr;  // define a pointer

    thing_var = 2;      // assigning a value to thing
    std::cout <<"Thing " << thing_var << '\n';

    thing_ptr = &thing_var; // make the pointer point to thing
    *thing_ptr = 3;     // thing_ptr points to thing_var so
                        // thing_var changes to 3
    std::cout << "Thing " << thing_var << '\n';

    // another way of printing the data
    std::cout << "Thing " << *thing_ptr << '\n';
    return (0);
}
```

Two pointers, one thing

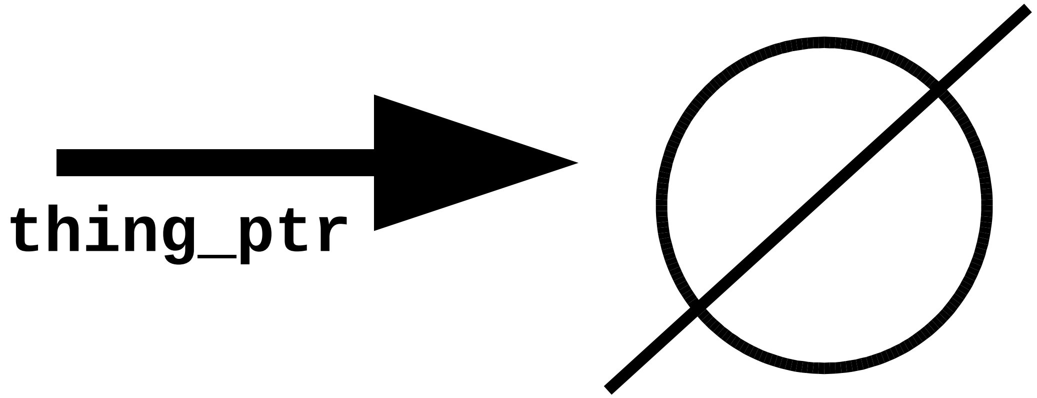
```
int         something;

int         *first_ptr;  // one pointer
int         *second_ptr; // another pointer

something = 1;         // give the thing a value

first_ptr = &something;
second_ptr = first_ptr;
```



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

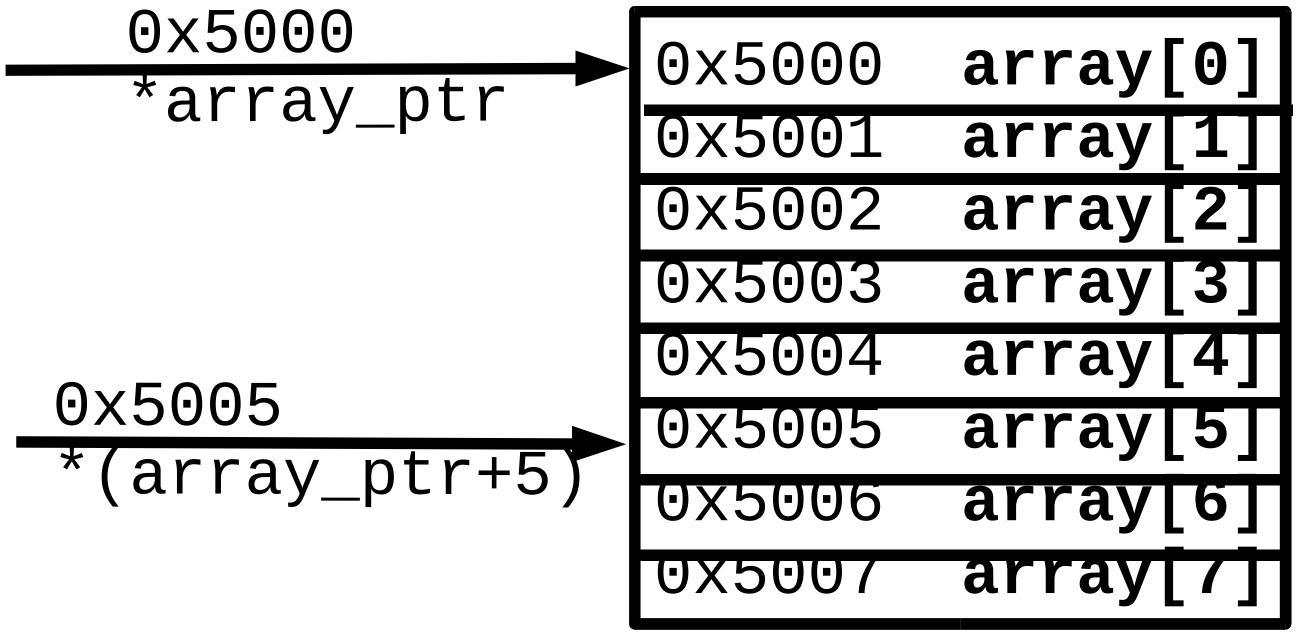
Example:
```
int an_integer = 5;         // A simple integer
int *int_ptr = &an_integer; // Ptr to an integer

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
   char *char_ptr = &some_characters[0];

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

Pointers and Arrays

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



Example

```
#include <iostream>
#include <iomanip.h>


const int ARRAY_SIZE  = 10; // Number of characters in array
// Array to print
char array[ARRAY_SIZE] = "012345678";


int main()
{
    int index;  /* Index into the array */


    for (index = 0; index < ARRAY_SIZE; ++index) {
        std::cout << hex;    // Trick to print hex numbers
        std::cout <<
            "&array[index]=0x" <<  int(&array[index]) <<
            " (array+index)=0x" << int(array+index) <<
            " array[index]=0x" <<  int(array[index]) << '\n',
       std::cout << dec;
    }
    return (0);
}
```

Output

```
&array[index]=0x20090 (array+index)=0x20090 array[index]=0x30
&array[index]=0x20091 (array+index)=0x20091 array[index]=0x31
&array[index]=0x20092 (array+index)=0x20092 array[index]=0x32
&array[index]=0x20093 (array+index)=0x20093 array[index]=0x33
&array[index]=0x20094 (array+index)=0x20094 array[index]=0x34
&array[index]=0x20095 (array+index)=0x20095 array[index]=0x35
&array[index]=0x20096 (array+index)=0x20096 array[index]=0x36
&array[index]=0x20097 (array+index)=0x20097 array[index]=0x37
&array[index]=0x20098 (array+index)=0x20098 array[index]=0x38
&array[index]=0x20099 (array+index)=0x20099 array[index]=0x0
```

Array Shorthand

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

is the same as:

```
array_ptr = array;
```

Summing an Array (Index Version)

```
#include <iostream>

int array[10] = {4, 5, 8, 9, 8, 1, 0, 1, 9, 3};
int index;

int main()
{
    index = 0;
    while (array[index] != 0)
        ++index;

    std::cout << "Number of elements before zero "
              << index << '\n';
    return (0);
}
```

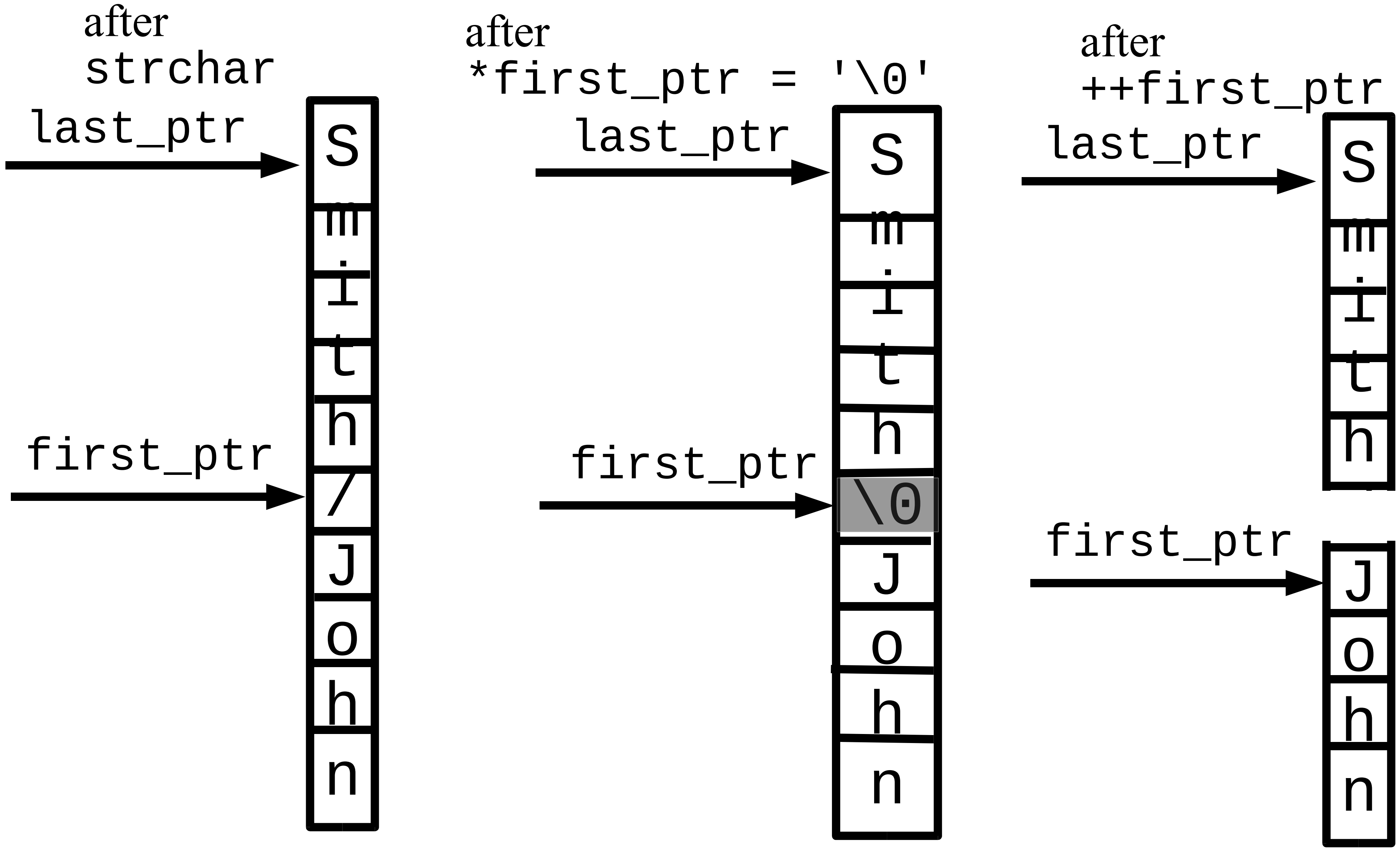
Same Program Using Pointers

```
#include <iostream>

int array[10] = {4, 5, 8, 9, 8, 1, 0, 1, 9, 3};
int *array_ptr;

main()
{
    array_ptr = array;

    while ((*array_ptr) != 0)
        ++array_ptr;

    std::cout << "Number of elements before zero " <<
        (array_ptr - array) << '\n';
    return (0);
}
```

Zeroing an array

```
const int MAX = 10;

void init_array_1(int data[])
{
    int  index;

    for (index = 0; index < MAX; ++index)
        data[index] = 0;
}

void init_array_2(int *data_ptr)
{
    int index;

    for (index = 0; index < MAX; ++index)
        *(data_ptr + index) = 0;
}

int main()
{
    int  array[MAX];

    // one way of initializing the array
    init_array_1(array);

    // another way of initializing the array
    init_array_1(&array[0]);

    // Similar to the first method but
    //    function is different
    init_array_2(array);

    return (0);
}
```

Splitting a C style string



Splitting a string

```
#include <iostream>
#include <string.h>
#include <stdlib.h>
main() {
    char line[80];    // The input line
    char *first_ptr;  // ptr we set to point to the first name
    char *last_ptr;   // ptr we set to point to the last name

    std::cin.getline(line, sizeof(line));

    last_ptr = line;    // last name is at beginning of line
    first_ptr = strchr(line, '/');      // Find slash

    // Check for an error
    if (first_ptr == NULL) {
        cerr << "Error: Unable to find slash in " << line << '\n';
        exit (8);
    }

    *first_ptr = '\0';  // Zero out the slash
    ++first_ptr;        // Move to first character of name

    std::cout << "First:" << first_ptr << " Last:" << last_ptr << '\n';
    return (0);
}
char *strchr(char * string_ptr, char find) {
    while (*string_ptr != find) {
       // Check for end
       if (*string_ptr == '\0')
           return (NULL);       // not found
        ++string_ptr;
    }
    return (string_ptr);        // Found
}
```

Question: Why does this program print garbage?

```
#include <iostream>
#include <string.h>

/********************************************************
 * tmp_name -- return a temporary file name             *
 *                                                      *
 * Each time this function is called, a new name will   *
 * be returned.                                         *
 *                                                      *
 * Returns                                              *
 *      Pointer to the new file name.                   *
 ********************************************************/
char *tmp_name(void)
{
    char name[30];              // The name we are generating
    static int sequence = 0;    // Sequence number for last digit

    ++sequence; // Move to the next file name

    strcpy(name, "tmp");

    // Put in the sequence digit
    name[3] = sequence + '0';

    // End the string
    name[4] = '\0';

    return(name);
}

int main()
{
    std::cout << "Name: " << tmp_name() << '\n';
    return(0);
}
```

Pointers and Structures

```
struct mailing {
    char name[60];    // last name, first name
    char address1[60];// Two lines of street address
    char address2[60];
    char city[40];
    char state[2];    // Two character abbreviation
    long int zip;     // numeric zip code
} list[MAX_ENTRIES];


// Pointer to the data
struct mailing *list_ptrs[MAX_ENTRIES];

int current;    // current mailing list entry

    // ....

    for (current = 0; current = number_of_entries; ++current)
        list_ptrs = &list[current];

    // Sort list_ptrs by zip code
```

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)

print_file

```
/********************************************************
 * print -- format files for printing                  *
 ********************************************************/
#include <iostream>
#include <stdlib.h>


int verbose = 0;            // verbose mode (default = false)
char *out_file = "print.out";   // output file name
char *program_name;       // name of the program (for errors)
int line_max = 66;              // number of lines per page


/********************************************************
 * do_file -- dummy routine to handle a file           *
 *                                                      *
 * Parameter                                            *
 *      name -- name of the file to print              *
 ********************************************************/
void do_file(char *name) {
    std::cout << "Verbose " << verbose <<
            " Lines " << line_max <<
            " Input " << name <<
            " Output " << out_file << '\n';
}
```

print_file (cont)

```
/********************************************************
 * usage -- tell the user how to use this program and   *
 *              exit                                    *
 ********************************************************/
void usage(void)
{
    cerr << "Usage is " << program_name <<
            " [options] [file-list]\n";
    cerr << "Options\n";
    cerr << "  -v          verbose\n";
    cerr << "  -l<number>  Number of lines\n";
    cerr << "  -o<name>    Set output file name\n";
    exit (8);
}
```

print_file (cont)

```
main(int argc, char *argv[])
{
    // save the program name for future use
    program_name = argv[0];


    /*
     * loop for each option.
     *   Stop if we run out of arguments
     *   or we get an argument without a dash.
     */
    while ((argc > 1) && (argv[1][0] == '-')) {
        /*
         * argv[1][1] is the actual option character.
         */
        switch (argv[1][1]) {
            /*
             * -v verbose
             */
            case 'v':
                verbose = 1;
                break;
```

print_file (cont)

```
/*
     * -o<name>  output file
     *    [0] is the dash
     *    [1] is the "o"
     *    [2] starts the name
     */
    case 'o':
        out_file = &argv[1][2];
        break;
    /*
     * -l<number> set max number of lines
     */
    case 'l':
        line_max = atoi(&argv[1][2]);
        break;
    default:
        cerr << "Bad option " << argv[1] <<'\n';
        usage();
}
```

print_file (cont)

```
            /*
             * move the argument list up one
             * move the count down one
             */
            ++argv;
            --argc;
        }


        /*
         * At this point all the options have been processed.
         * Check to see if we have no files in the list
         * and if so, we need to list just standard in.
         */
        if (argc == 1) {
            do_file("print.in");
        } else {
            while (argc > 1) {
              do_file(argv[1]);
              ++argv;
              --argc;
            }
        }
        return (0);
    }
```