10.2. Arguments to main

For those writing programs which will run in a hosted environment, arguments to main provide a useful opportunity to give parameters to programs. Typically, this facility is used to direct the way the program goes about its task. It's particularly common to provide file names to a program through its arguments.

The declaration of main looks like this:

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

This indicates that `main` is a function returning an integer. In hosted environments such as DOS or UNIX, this value or *exit status* is passed back to the command line interpreter. Under UNIX, for example, the exit status is used to indicate that a program completed successfully (a zero value) or some error occurred (a non-zero value). The Standard has adopted this convention; `exit(0)` is used to return 'success' to its host environment, any other value is used to indicate failure. If the host environment itself uses a different numbering convention, `exit` will do the necessary translation. Since the translation is implementation-defined, it is now considered better practice to use the values defined in `<stdlib.h>`: `EXIT_SUCCESS` and `EXIT_FAILURE`.

There are at least two arguments to `main`: `argc` and `argv`. The first of these is a count of the arguments supplied to the program and the second is an array of pointers to the strings which are those arguments—its type is (almost) ‘array of pointer to `char`’. These arguments are passed to the program by the host system's command line interpreter or job control language.

The declaration of the `argv` argument is often a novice programmer's first encounter with pointers to arrays of pointers and can prove intimidating. However, it is really quite simple to understand. Since `argv` is used to refer to an array of strings, its declaration will look like this:

```
char *argv[]
```

Remember too that when it is passed to a function, the name of an array is converted to the address of its first element. This means that we can also declare `argv` as `char **argv`; the two declarations are equivalent in this context.

Indeed, you will often see the declaration of `main` expressed in these terms. This declaration is exactly equivalent to that shown above:

```
int main(int argc, char **argv);
```

When a program starts, the arguments to main will have been initialized to meet the following conditions:

- `argc` is greater than zero.
- `argv[argc]` is a null pointer.
- `argv[0]` through to `argv[argc-1]` are pointers to strings whose meaning will be determined by the program.
- `argv[0]` will be a string containing the program's name or a null string if that is not available. Remaining elements of `argv` represent the arguments supplied to the program. In cases where there is only support for single-case characters, the

contents of these strings will be supplied to the program in lower-case.

To illustrate these points, here is a simple program which writes the arguments supplied to `main` on the program's standard output.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
        while(argc--)
                printf("%s\n", *argv++);
        exit(EXIT_SUCCESS);
}
```

Example 10.1

If the program name is `show_args` and it has arguments `abcde`, `text`, and `hello` when it is run, the state of the arguments and the value of `argv` can be illustrated like this:

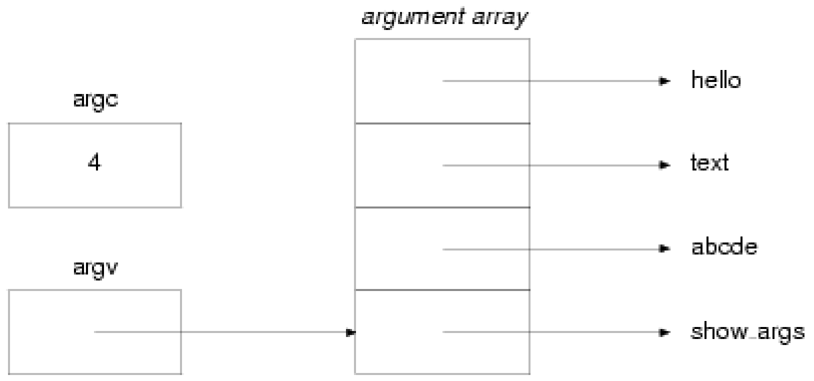

Figure 10.1. Arguments to a program

Each time that `argv` is incremented, it is stepped one item further along the array of arguments. Thus after the first iteration of the loop, `argv` will point to the pointer which in turn points to the `abcde` argument. This is shown in Figure 10.2.

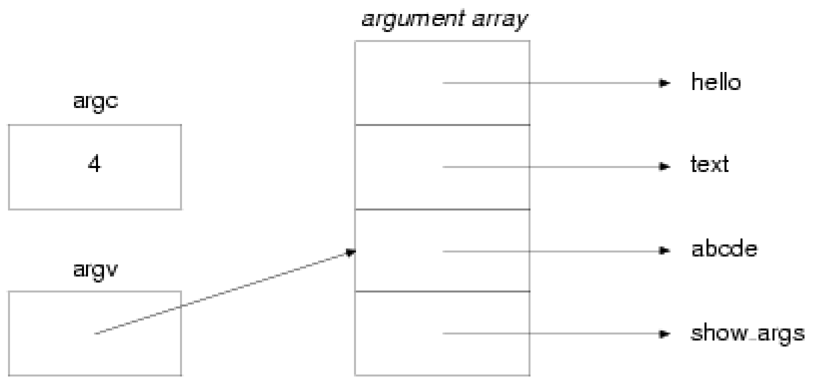

Figure 10.2. Arguments to a program after incrementing `argv`

On the system where this program was tested, a program is run by typing its name and then the arguments, separated by spaces. This is what happened (the `$` is a prompt):

```
$ show_args abcde text hello
show_args
abcde
text
hello
$
```

Previous section | Chapter contents | Next section