命令列參數 (Command-Line Arguments)

Command-Line Arguments

- When we run a program, we'll often need to supply it with information.
- This may include a file name or a switch that modifies the program's behavior.
- Examples of the UNIX `ls` command:

```
ls
ls -l
ls -l remind.c
```

Command-Line Arguments

- Command-line information is available to all programs, not just operating system commands.
- To obtain access to ***command-line arguments***, `main` must have two parameters:

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

- Command-line arguments are called ***program parameters*** in the C standard.

Command-Line Arguments

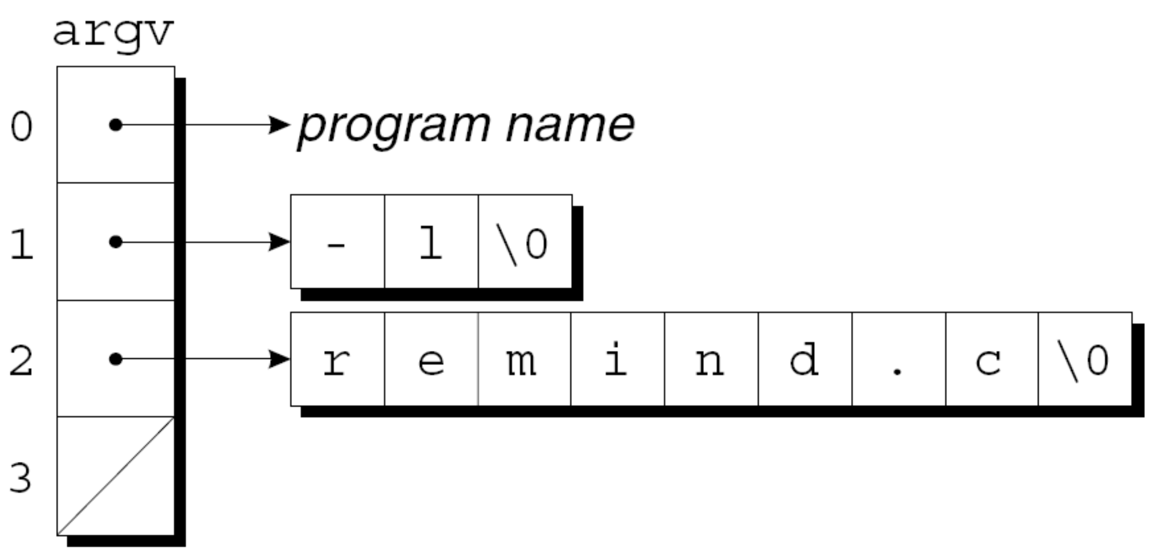
- `argc` ("argument count") is the number of command-line arguments.
- `argv` ("argument vector") is an array of pointers to the command-line arguments (stored as strings).
- `argv[0]` points to the name of the program, while `argv[1]` through `argv[argc-1]` point to the remaining command-line arguments.
- `argv[argc]` is always a ***null pointer***—a special pointer that points to nothing.
  - The macro `NULL` represents a null pointer.

Command-Line Arguments

- If the user enters the command line

  ```
  ls -l remind.c
  ```

  then `argc` will be 3, and `argv` will have the following appearance:



Command-Line Arguments

- Since `argv` is an array of pointers, accessing command-line arguments is easy.
- Typically, a program that expects command-line arguments will set up a loop that examines each argument in turn.
- One way to write such a loop is to use an integer variable as an index into the `argv` array:

```
int i;

for (i = 1; i < argc; i++)
  printf("%s\n", argv[i]);
```

Command-Line Arguments

- Another technique is to set up a pointer to `argv[1]`, then increment the pointer repeatedly:

```
char **p;

for (p = &argv[1]; *p != NULL; p++)
  printf("%s\n", *p);
```