The Command-Line Arguments

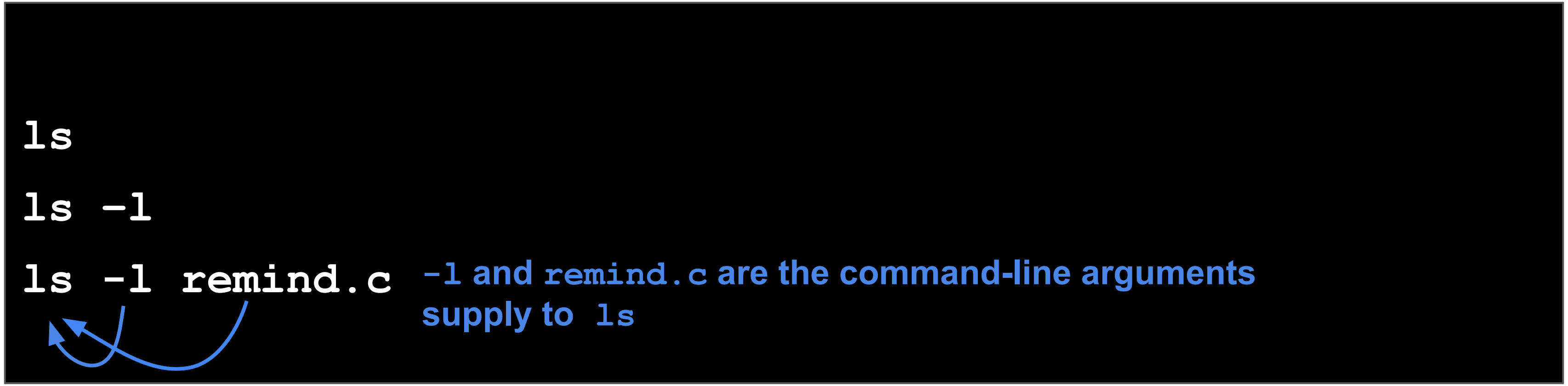
Program Design (II)

2022 Spring

Fu-Yin Cherng
Dept. CSIE, National Chung Cheng University

Introduction

- After we know how to use array of string and use terminal to run C program
- We can learn command-line aruguments now
- When we run a program, we’ll often need to supply it with information.
- Examples of the UNIX ls command:



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

`-l` and `remind.c` are the command-line arguments supply to `ls`

Command-Line Arguments

- Command-line information is available to all programs, not just operating system commands.
- We can also make our C program obtain command-line information!
- For example, write a C program to print the strings written in command-line

```
./a.out Hellow World
```

Command-Line Arguments

- To obtain access to ***command-line arguments*,** `main` must have two parameters:
- Command-line arguments are called ***program parameters*** in the C standard.

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

Command-Line Arguments

- `argc` (“argument count”) is the int number of command-line arguments.
- For example, If the user enters the command line, then argc will be 3

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

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

Command-Line Arguments

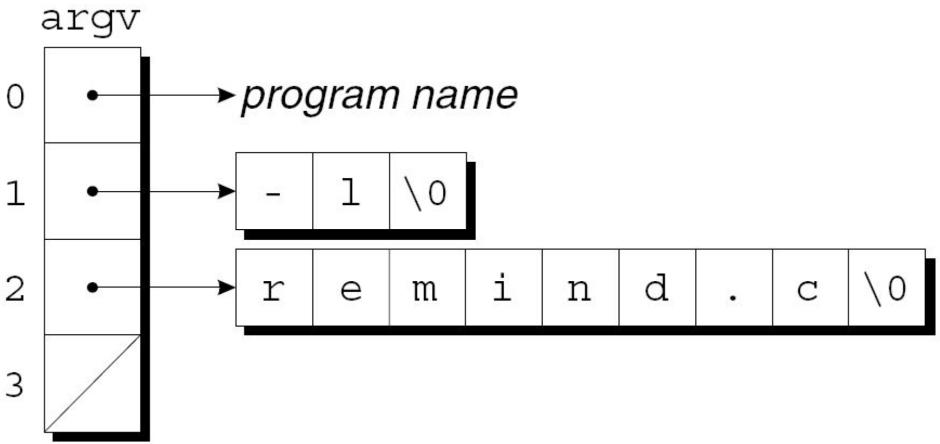
- `argv` ("argument vector") is an array of pointers to the command-line arguments (stored as strings).
  - `argv[0]` points to the **name of the program**, while `argv[1]` through `argv[argc-1]` point to the **remaining command-line arguments.**

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

Command-Line Arguments

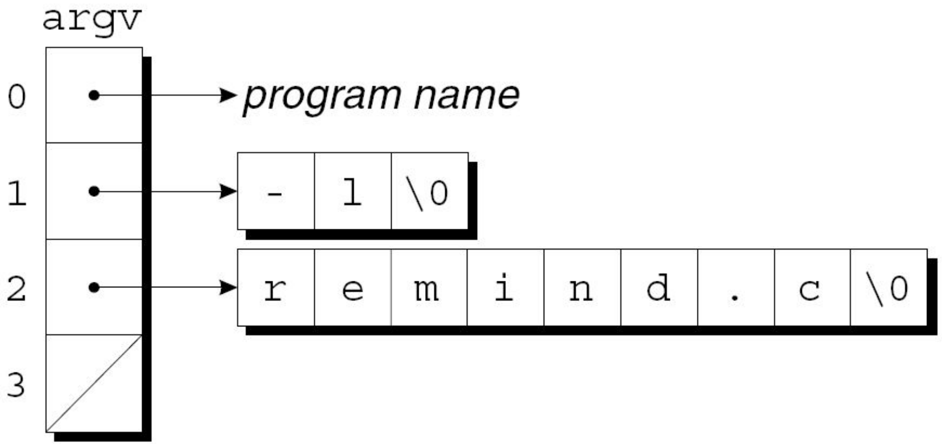
- If the user enters the command line, `argv` will have the following appearance
- `argv[argc]` (argc is 3) is always a ***null pointer***
  - a special pointer that points to nothing.
  - The macro `NULL` represents a null pointer.
  - `int *p = NULL;`

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



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.



Command-Line Arguments

- One way to write such a loop is to use an integer variable as an index into the `argv` array:

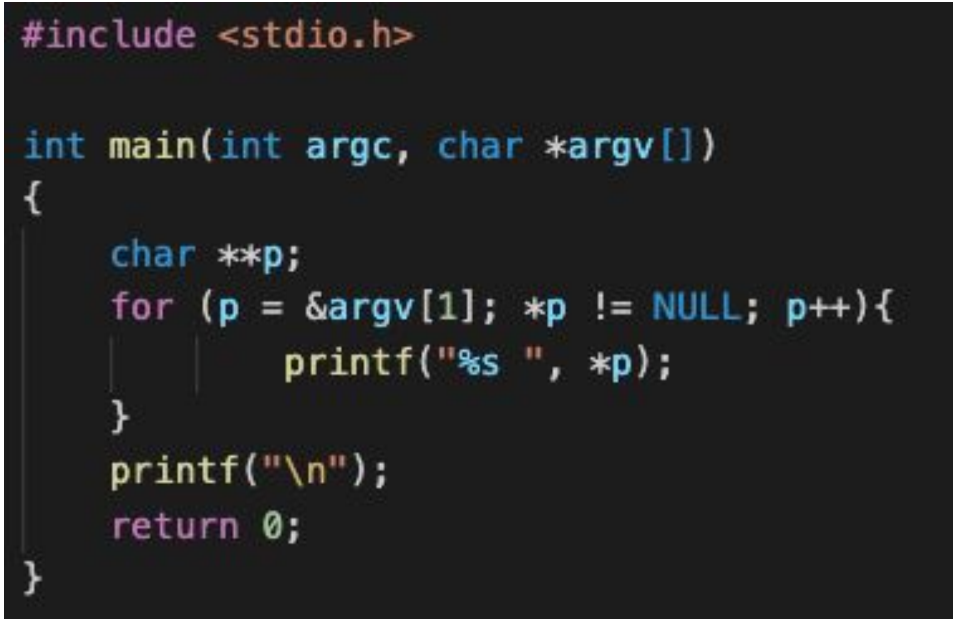
```
int main(int argc, char *argv[]){
    int i;
    for (i = 1; i < argc; i++){
        printf("%s\n", argv[i]);
    }
    …
}
```

Command-Line Arguments

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

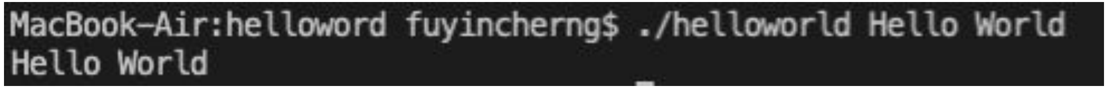
```
int main(int argc, char *argv[]){
    char **p;
    for (p = &argv[1]; *p != NULL; p++){
        printf("%s\n", *p);
    }
    …
}
```

Command-Line Arguments: Revised helloworld.c

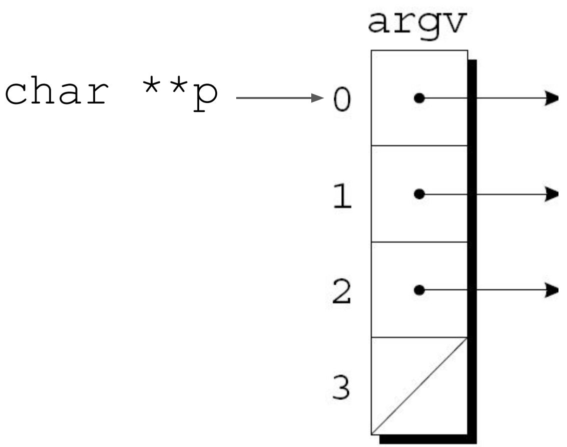


```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    char **p;
    for (p = &argv[1]; *p != NULL; p++){
            printf("%s ", *p);
    }
    printf("\n");
    return 0;
}
```



```
MacBook-Air:helloword fuyincherng$ ./helloworld Hello World
Hello World
```



argv
char **p
0
1
2
3