Command Line Arguments

Prof. Jyotiprakash Mishra
mail@jyotiprakash.org

Basic argc and argv

Program 1:

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
  int i;
  printf("argc = %d\n", argc);
  for (i = 0; i < argc; i++) {
    printf("argv[%d] = %s\n", i, argv[i]);
  }
  return 0;
}
```

Run:

```
./prog hello world 123
```

Output:

```
argc = 4
argv[0] = ./prog
argv[1] = hello
argv[2] = world
argv[3] = 123
```

Note:

```
argc: argument count (includes
program name). argv: argument vector
(array of strings). argv[0] is
always program name.
```

Checking Argument Count

Program 2:

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
  if (argc != 3) {
    printf("Usage: %s <name> <age>\n",
      argv[0]);
    return 1;
  }
  printf("Name: %s\n", argv[1]);
  printf("Age: %s\n", argv[2]);
  return 0;
}
```

Run 1:

```
./prog
```

Output:

```
Usage: ./prog <name> <age>
```

Run 2:

```
./prog Alice 25
```

Output:

```
Name: Alice
Age: 25
```

Note:

```
Always validate argc. Print usage
message if wrong number of arguments.
Return non-zero on error.
```

Converting Arguments to Numbers

Program 3:

```c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
  if (argc != 3) {
    printf("Usage: %s <num1> <num2>\n",
      argv[0]);
    return 1;
  }
  int a = atoi(argv[1]);
  int b = atoi(argv[2]);
  printf("%d + %d = %d\n", a, b, a + b);
  printf("%d * %d = %d\n", a, b, a * b);
  return 0;
}
```

Run:

```
./prog 15 7
```

Output:

```
15 + 7 = 22
15 * 7 = 105
```

Note:

```
argv elements are strings. Use atoi
to convert to int, atof for float,
atol for long. Arguments passed as
text, not numbers.
```

Simple File Processor

Program 4:

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
  FILE *fp;
  char ch;
  if (argc != 2) {
    printf("Usage: %s <filename>\n",
      argv[0]);
    return 1;
  }
  fp = fopen(argv[1], "r");
  if (fp == NULL) {
    printf("Error opening %s\n",
      argv[1]);
    return 1;
  }
  while ((ch = fgetc(fp)) != EOF) {
    putchar(ch);
  }
  fclose(fp);
  return 0;
}
```

Run:

```
./prog test.txt
```

Output:

```
(contents of test.txt displayed)
```

Note:

```
Common pattern: filename as argument.
Open file, process, close. Like cat
command. Always check fopen return.
```

Optional Arguments with Flags

Program 5:

```c
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[]) {
  int verbose = 0;
  int i;
  for (i = 1; i < argc; i++) {
    if (strcmp(argv[i], "-v") == 0) {
      verbose = 1;
    } else if (strcmp(argv[i], "-h") == 0) {
      printf("Help: use -v for verbose\n");
      return 0;
    }
  }
  printf("Running program\n");
  if (verbose) {
    printf("Verbose mode enabled\n");
  }
  return 0;
}
```

Run 1:

```
./prog
```

Output:

```
Running program
```

Run 2:

```
./prog -v
```

Output:

```
Running program
Verbose mode enabled
```

Note:

Flags modify behavior. Check each argument for known flags. -v, -h are common conventions.

Option with Value

Program 6:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char *argv[]) {
  int count = 1;
  int i;
  for (i = 1; i < argc; i++) {
    if (strcmp(argv[i], "-n") == 0) {
      if (i + 1 < argc) {
        count = atoi(argv[++i]);
      }
    }
  }
  for (i = 0; i < count; i++) {
    printf("Hello %d\n", i + 1);
  }
  return 0;
}
```

Run 1:

```
./prog
```

Output:

```
Hello 1
```

Run 2:

```
./prog -n 3
```

Output:

```
Hello 1
Hello 2
Hello 3
```

Note:

```
Option followed by value. -n 3
means next argument is value for -n.
Increment i to skip value.
```

Processing Multiple Files

Program 7:

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
  FILE *fp;
  int i, lines;
  char ch;
  if (argc < 2) {
    printf("Usage: %s <files...>\n",
      argv[0]);
    return 1;
  }
  for (i = 1; i < argc; i++) {
    fp = fopen(argv[i], "r");
    if (fp == NULL) continue;
    lines = 0;
    while ((ch = fgetc(fp)) != EOF) {
      if (ch == '\n') lines++;
    }
    printf("%s: %d lines\n",
      argv[i], lines);
    fclose(fp);
  }
  return 0;
}
```

Run:

```
./prog file1.txt file2.txt file3.txt
```

Output:

```
file1.txt: 10 lines
file2.txt: 25 lines
file3.txt: 5 lines
```

Note:

```
Process all arguments as files.
Like wc -l command. Loop through
argv starting from index 1.
```

Environment Variables

Program 8:

```c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
  char *home = getenv("HOME");
  char *user = getenv("USER");
  char *path = getenv("PATH");
  if (home)
    printf("HOME: %s\n", home);
  if (user)
    printf("USER: %s\n", user);
  if (path)
    printf("PATH: %s\n", path);
  return 0;
}
```

Output:

```
HOME: /Users/username
USER: username
PATH: /usr/bin:/bin:/usr/sbin
```

Note:

```
getenv retrieves environment
variables. Returns NULL if not set.
Common variables: HOME, USER, PATH.
Complement to command line args.
```

Calculator with Arguments

Program 9:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char *argv[]) {
  int a, b;
  if (argc != 4) {
    printf("Usage: %s <num1> <op> <num2>\n",
      argv[0]);
    return 1;
  }
  a = atoi(argv[1]);
  b = atoi(argv[3]);
  if (strcmp(argv[2], "+") == 0)
    printf("%d\n", a + b);
  else if (strcmp(argv[2], "-") == 0)
    printf("%d\n", a - b);
  else if (strcmp(argv[2], "x") == 0)
    printf("%d\n", a * b);
  else
    printf("Unknown op\n");
  return 0;
}
```

Run 1:

```
./calc 10 + 5
```

Output:

```
15
```

Run 2:

```
./calc 10 x 5
```

Output:

```
50
```

Note:

```
Command line calculator. Parse
operator as string. Use x instead
of * (shell expands *).
```

grep-like Pattern Search

Program 10:

```c
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[]) {
  FILE *fp;
  char line[256];
  if (argc != 3) {
    printf("Usage: %s <pattern> <file>\n",
      argv[0]);
    return 1;
  }
  fp = fopen(argv[2], "r");
  if (fp == NULL) {
    printf("Cannot open %s\n", argv[2]);
    return 1;
  }
  while (fgets(line, sizeof(line), fp)) {
    if (strstr(line, argv[1])) {
      printf("%s", line);
    }
  }
  fclose(fp);
  return 0;
}
```

Run:

```
./search error logfile.txt
```

Output:

```
Line 5: error in module A
Line 12: critical error detected
```

Note:

```
Simple grep implementation. Search
pattern in file. strstr finds
substring. Print matching lines.
```

Converting Case

Program 11:

```c
#include <stdio.h>
#include <ctype.h>
#include <string.h>
int main(int argc, char *argv[]) {
  FILE *fp;
  int ch, upper = 0;
  if (argc < 2) return 1;
  if (strcmp(argv[1], "-u") == 0)
    upper = 1;
  int file_idx = upper ? 2 : 1;
  if (argc <= file_idx) return 1;
  fp = fopen(argv[file_idx], "r");
  if (fp == NULL) return 1;
  while ((ch = fgetc(fp)) != EOF) {
    putchar(upper ? toupper(ch) :
      tolower(ch));
  }
  fclose(fp);
  return 0;
}
```

Run 1:

```
./case file.txt
```

Output:

```
(file in lowercase)
```

Run 2:

```
./case -u file.txt
```

Output:

```
(FILE IN UPPERCASE)
```

Note:

```
Optional flag changes behavior.
Default lowercase, -u for uppercase.
Adjust file index based on flag.
```

Word Count Tool

Program 12:

```c
#include <stdio.h>
#include <ctype.h>
int main(int argc, char *argv[]) {
  FILE *fp;
  int ch, words = 0, in_word = 0;
  if (argc != 2) {
    printf("Usage: %s <file>\n", argv[0]);
    return 1;
  }
  fp = fopen(argv[1], "r");
  if (fp == NULL) return 1;
  while ((ch = fgetc(fp)) != EOF) {
    if (isspace(ch)) {
      in_word = 0;
    } else if (!in_word) {
      in_word = 1;
      words++;
    }
  }
  printf("Words: %d\n", words);
  fclose(fp);
  return 0;
}
```

Run:

```
./wc document.txt
```

Output:

```
Words: 245
```

Note:

```
Count words in file. Track whether
in word or between words. Transition
from space to non-space starts word.
```

File Copy with Arguments

Program 13:

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
  FILE *src, *dst;
  int ch;
  if (argc != 3) {
    printf("Usage: %s <src> <dst>\n",
      argv[0]);
    return 1;
  }
  src = fopen(argv[1], "r");
  if (src == NULL) {
    printf("Cannot open %s\n", argv[1]);
    return 1;
  }
  dst = fopen(argv[2], "w");
  if (dst == NULL) {
    fclose(src);
    return 1;
  }
  while ((ch = fgetc(src)) != EOF) {
    fputc(ch, dst);
  }
  printf("Copied %s to %s\n",
    argv[1], argv[2]);
  fclose(src);
  fclose(dst);
  return 0;
}
```

Run:

```
./cp source.txt dest.txt
```

Output:

```
Copied source.txt to dest.txt
```

Note:

```
cp command implementation. Two
filenames as arguments. Open source
for reading, dest for writing.
Copy byte by byte.
```

Hex Dump Utility

Program 14:

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
  FILE *fp;
  unsigned char ch;
  int count = 0;
  if (argc != 2) {
    printf("Usage: %s <file>\n", argv[0]);
    return 1;
  }
  fp = fopen(argv[1], "rb");
  if (fp == NULL) return 1;
  while (fread(&ch, 1, 1, fp) == 1) {
    printf("%02X ", ch);
    count++;
    if (count % 16 == 0) printf("\n");
  }
  if (count % 16 != 0) printf("\n");
  fclose(fp);
  return 0;
}
```

Run:

```
./hexdump data.bin
```

Output:

```
48 65 6C 6C 6F 20 57 6F 72 6C 64 0A
```

Note:

```
Display file in hexadecimal. Open
in binary mode. Print each byte as
2-digit hex. 16 bytes per line.
```

Reversing File Contents

Program 15:

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
  FILE *fp;
  long size, i;
  char ch;
  if (argc != 2) return 1;
  fp = fopen(argv[1], "r");
  if (fp == NULL) return 1;
  fseek(fp, 0, SEEK_END);
  size = ftell(fp);
  for (i = size - 1; i >= 0; i--) {
    fseek(fp, i, SEEK_SET);
    ch = fgetc(fp);
    putchar(ch);
  }
  fclose(fp);
  return 0;
}
```

file.txt:

```
Hello World
```

Run:

```
./reverse file.txt
```

Output:

```
dlroW olleH
```

Note:

```
Read file backwards. Get file size,
seek to end, then read backwards
one char at a time.
```

Configurable Output

Program 16:

```c
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[]) {
  FILE *out = stdout;
  int i;
  for (i = 1; i < argc; i++) {
    if (strcmp(argv[i], "-o") == 0 &&
      i + 1 < argc) {
      out = fopen(argv[++i], "w");
      if (out == NULL) {
        out = stdout;
      }
    }
  }
  fprintf(out, "Output line 1\n");
  fprintf(out, "Output line 2\n");
  if (out != stdout) fclose(out);
  return 0;
}
```

Run 1:

```
./prog
```

Output:

```
Output line 1
Output line 2
```

Run 2:

```
./prog -o result.txt
```

result.txt:

```
Output line 1
Output line 2
```

Note:

```
-o flag redirects output to file.
Default to stdout. Same code writes
to screen or file.
```

Exit Codes

Program 17:

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
  FILE *fp;
  if (argc != 2) {
    fprintf(stderr, "Usage: %s <file>\n",
      argv[0]);
    return 1;
  }
  fp = fopen(argv[1], "r");
  if (fp == NULL) {
    fprintf(stderr, "Error: cannot open\n");
    return 2;
  }
  printf("File opened successfully\n");
  fclose(fp);
  return 0;
}
```

Shell:

```
./prog
echo $?
```

Output:

```
Usage: ./prog <file>
1
```

Shell:

```
./prog nofile.txt
echo $?
```

Output:

```
Error: cannot open
2
```

Note:

```
Return different codes for different
errors. 0 = success. Non-zero =
error. Shell can check $?.
```

Long Options

Program 18:

```c
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[]) {
  int verbose = 0, debug = 0;
  int i;
  for (i = 1; i < argc; i++) {
    if (strcmp(argv[i], "--verbose") == 0)
      verbose = 1;
    else if (strcmp(argv[i],
      "--debug") == 0)
      debug = 1;
    else if (strcmp(argv[i],
      "--help") == 0) {
      printf("Options: --verbose --debug\n");
      return 0;
    }
  }
  printf("Running\n");
  if (verbose) printf("Verbose on\n");
  if (debug) printf("Debug on\n");
  return 0;
}
```

Run:

```
./prog --verbose --debug
```

Output:

```
Running
Verbose on
Debug on
```

Note:

Long options with -- prefix. More descriptive than single letter. --help, --verbose common conventions.

Combining Short Options

Program 19:

```c
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[]) {
  int i, j;
  int flags[26] = {0};
  for (i = 1; i < argc; i++) {
    if (argv[i][0] == '-' &&
      argv[i][1] != '-') {
      for (j = 1; argv[i][j]; j++) {
        if (argv[i][j] >= 'a' &&
          argv[i][j] <= 'z') {
          flags[argv[i][j] - 'a'] = 1;
        }
      }
    }
  }
  if (flags['v'-'a'])
    printf("Verbose on\n");
  if (flags['d'-'a'])
    printf("Debug on\n");
  return 0;
}
```

Run:

```
./prog -vd
```

Output:

```
Verbose on
Debug on
```

Note:

```
Combine short flags: -vd = -v -d.
Parse each character after -.
Store in array indexed by letter.
Unix convention.
```

Reading from stdin or File

Program 20:

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
  FILE *input = stdin;
  char line[256];
  if (argc > 1) {
    input = fopen(argv[1], "r");
    if (input == NULL) {
      printf("Cannot open %s\n", argv[1]);
      return 1;
    }
  }
  while (fgets(line, sizeof(line), input)) {
    printf("Read: %s", line);
  }
  if (input != stdin) fclose(input);
  return 0;
}
```

Run 1:

```
echo "test" | ./prog
```

Output:

```
Read: test
```

Run 2:

```
./prog data.txt
```

Output:

```
Read: (contents of data.txt)
```

Note:

```
Read from file or stdin. If no args,
use stdin. Enables piping. Flexible
input source.
```