File I/O

Prof. Jyotiprakash Mishra
mail@jyotiprakash.org

Writing to Text File

Program 1:

```c
#include <stdio.h>
int main() {
  FILE *fp;
  fp = fopen("output.txt", "w");
  if (fp == NULL) {
    printf("Error opening file\n");
    return 1;
  }
  fprintf(fp, "Hello, World!\n");
  fprintf(fp, "Number: %d\n", 42);
  fprintf(fp, "Float: %.2f\n", 3.14);
  fclose(fp);
  printf("File written successfully\n");
  return 0;
}
```

Output:

```
File written successfully
```

output.txt:

```
Hello, World!
Number: 42
Float: 3.14
```

Note:

```
fopen with "w" creates/overwrites
file. fprintf writes formatted
data. Always check for NULL and
close file with fclose.
```

Reading from Text File

Program 2:

```c
#include <stdio.h>
int main() {
  FILE *fp;
  char line[100];
  fp = fopen("output.txt", "r");
  if (fp == NULL) {
    printf("Error opening file\n");
    return 1;
  }
  while (fgets(line, sizeof(line), fp)) {
    printf("%s", line);
  }
  fclose(fp);
  return 0;
}
```

Output:

```
Hello, World!
Number: 42
Float: 3.14
```

Note:

```
fopen with "r" opens for reading.
fgets reads one line at a time
including newline. Returns NULL
at end of file.
```

Appending to File

Program 3:

```c
#include <stdio.h>
int main() {
  FILE *fp;
  fp = fopen("output.txt", "a");
  if (fp == NULL) {
    printf("Error opening file\n");
    return 1;
  }
  fprintf(fp, "Appended line 1\n");
  fprintf(fp, "Appended line 2\n");
  fclose(fp);
  printf("Data appended\n");
  return 0;
}
```

Output:

```
Data appended
```

output.txt:

```
Hello, World!
Number: 42
Float: 3.14
Appended line 1
Appended line 2
```

Note:

```
Mode "a" appends to end of file.
Creates file if doesn’t exist.
Existing content preserved.
```

Character I/O

Program 4:

```c
#include <stdio.h>
int main() {
  FILE *fp;
  char ch;
  fp = fopen("chars.txt", "w");
  if (fp == NULL) return 1;
  fputc('A', fp);
  fputc('B', fp);
  fputc('\n', fp);
  fclose(fp);
  fp = fopen("chars.txt", "r");
  if (fp == NULL) return 1;
  while ((ch = fgetc(fp)) != EOF) {
    printf("%c", ch);
  }
  fclose(fp);
  return 0;
}
```

Output:

```
AB
```

Note:

```
fputc writes single character.
fgetc reads single character.
Returns EOF at end of file.
Character-by-character processing.
```

String I/O

Program 5:

```c
#include <stdio.h>
int main() {
  FILE *fp;
  char line[100];
  fp = fopen("strings.txt", "w");
  if (fp == NULL) return 1;
  fputs("First line\n", fp);
  fputs("Second line\n", fp);
  fclose(fp);
  fp = fopen("strings.txt", "r");
  if (fp == NULL) return 1;
  while (fgets(line, sizeof(line), fp)) {
    fputs(line, stdout);
  }
  fclose(fp);
  return 0;
}
```

Output:

```
First line
Second line
```

Note:

```
fputs writes string to file.
fgets reads line from file.
fputs to stdout prints to screen.
String-based I/O.
```

Formatted Input

Program 6:

```c
#include <stdio.h>
int main() {
  FILE *fp;
  int num;
  float fnum;
  char str[50];
  fp = fopen("data.txt", "w");
  if (fp == NULL) return 1;
  fprintf(fp, "100 3.14 Hello");
  fclose(fp);
  fp = fopen("data.txt", "r");
  if (fp == NULL) return 1;
  fscanf(fp, "%d %f %s", &num, &fnum, str);
  printf("Int: %d\n", num);
  printf("Float: %.2f\n", fnum);
  printf("String: %s\n", str);
  fclose(fp);
  return 0;
}
```

Output:

```
Int: 100
Float: 3.14
String: Hello
```

Note:

```
fscanf reads formatted data from
file like scanf from stdin.
Parses according to format string.
Returns number of items read.
```

Binary File Write

Program 7:

```c
#include <stdio.h>
struct Record {
  int id;
  float value;
};
int main() {
  FILE *fp;
  struct Record r1 = {101, 45.5};
  struct Record r2 = {102, 67.8};
  fp = fopen("data.bin", "wb");
  if (fp == NULL) return 1;
  fwrite(&r1, sizeof(struct Record), 1, fp);
  fwrite(&r2, sizeof(struct Record), 1, fp);
  fclose(fp);
  printf("Binary data written\n");
  return 0;
}
```

Output:

```
Binary data written
```

Note:

```
Mode "wb" opens binary file for
writing. fwrite writes raw bytes.
Parameters: pointer, size, count,
file pointer. Efficient for structs.
```

Binary File Read

Program 8:

```c
#include <stdio.h>
struct Record {
  int id;
  float value;
};
int main() {
  FILE *fp;
  struct Record r;
  fp = fopen("data.bin", "rb");
  if (fp == NULL) return 1;
  while (fread(&r, sizeof(struct Record),
    1, fp) == 1) {
    printf("ID: %d, Value: %.1f\n",
      r.id, r.value);
  }
  fclose(fp);
  return 0;
}
```

Output:

```
ID: 101, Value: 45.5
ID: 102, Value: 67.8
```

Note:

```
Mode "rb" opens binary file for
reading. fread reads raw bytes.
Returns number of items read.
Returns 0 at end of file.
```

File Positioning - fseek

Program 9:

```c
#include <stdio.h>
int main() {
  FILE *fp;
  int arr[] = {10, 20, 30, 40, 50};
  int val, i;
  fp = fopen("nums.bin", "wb");
  fwrite(arr, sizeof(int), 5, fp);
  fclose(fp);
  fp = fopen("nums.bin", "rb");
  fseek(fp, 2 * sizeof(int), SEEK_SET);
  fread(&val, sizeof(int), 1, fp);
  printf("Value at index 2: %d\n", val);
  fclose(fp);
  return 0;
}
```

Output:

```
Value at index 2: 30
```

Note:

```
fseek moves file position.
SEEK_SET: from beginning
SEEK_CUR: from current position
SEEK_END: from end
Random access to file data.
```

File Positioning - ftell and rewind

Program 10:

```c
#include <stdio.h>
int main() {
  FILE *fp;
  long pos;
  fp = fopen("test.txt", "w");
  fprintf(fp, "Hello");
  pos = ftell(fp);
  printf("Position: %ld\n", pos);
  fprintf(fp, " World");
  pos = ftell(fp);
  printf("Position: %ld\n", pos);
  rewind(fp);
  pos = ftell(fp);
  printf("After rewind: %ld\n", pos);
  fclose(fp);
  return 0;
}
```

Output:

```
Position: 5
Position: 11
After rewind: 0
```

Note:

```
ftell returns current file position
in bytes. rewind moves position to
beginning. Equivalent to
fseek(fp, 0, SEEK_SET).
```

File Size

Program 11:

```c
#include <stdio.h>
int main() {
  FILE *fp;
  long size;
  fp = fopen("output.txt", "r");
  if (fp == NULL) {
    printf("Error opening file\n");
    return 1;
  }
  fseek(fp, 0, SEEK_END);
  size = ftell(fp);
  rewind(fp);
  printf("File size: %ld bytes\n", size);
  fclose(fp);
  return 0;
}
```

Output:

```
File size: 60 bytes
```

Note:

```
Seek to end, get position with
ftell to determine file size.
Then rewind to beginning.
Common idiom for file size.
```

Error Checking - ferror and feof

Program 12:

```c
#include <stdio.h>
int main() {
  FILE *fp;
  int ch;
  fp = fopen("output.txt", "r");
  if (fp == NULL) return 1;
  while ((ch = fgetc(fp)) != EOF) {
    if (ferror(fp)) {
      printf("Read error occurred\n");
      break;
    }
  }
  if (feof(fp)) {
    printf("End of file reached\n");
  }
  fclose(fp);
  return 0;
}
```

Output:

```
End of file reached
```

Note:

ferror checks for errors during I/O operations. feof checks if end of file reached. Distinguish EOF from actual errors.

File Copy

Program 13:

```c
#include <stdio.h>
int main() {
  FILE *src, *dst;
  int ch;
  src = fopen("output.txt", "r");
  if (src == NULL) return 1;
  dst = fopen("copy.txt", "w");
  if (dst == NULL) {
    fclose(src);
    return 1;
  }
  while ((ch = fgetc(src)) != EOF) {
    fputc(ch, dst);
  }
  fclose(src);
  fclose(dst);
  printf("File copied successfully\n");
  return 0;
}
```

Output:

```
File copied successfully
```

Note:

```
Open source for reading, destination
for writing. Read character by
character and write to destination.
Close both files when done.
```

Line Count

Program 14:

```c
#include <stdio.h>
int main() {
  FILE *fp;
  int ch, lines = 0;
  fp = fopen("output.txt", "r");
  if (fp == NULL) {
    printf("Error opening file\n");
    return 1;
  }
  while ((ch = fgetc(fp)) != EOF) {
    if (ch == '\n') {
      lines++;
    }
  }
  fclose(fp);
  printf("Number of lines: %d\n", lines);
  return 0;
}
```

Output:

```
Number of lines: 5
```

Note:

```
Read file character by character.
Count newline characters to
determine number of lines.
Simple text processing.
```

Read/Write Mode

Program 15:

```c
#include <stdio.h>
int main() {
  FILE *fp;
  char data[50];
  fp = fopen("rw.txt", "w+");
  if (fp == NULL) return 1;
  fprintf(fp, "Hello");
  rewind(fp);
  fgets(data, sizeof(data), fp);
  printf("Read: %s\n", data);
  fprintf(fp, " World");
  fclose(fp);
  return 0;
}
```

Output:

```
Read: Hello
```

rw.txt:

```
Hello World
```

Note:

```
Mode "w+" allows both reading and
writing. Creates new file. "r+"
requires existing file. "a+" appends
and allows reading.
```

Binary Array I/O

Program 16:

```c
#include <stdio.h>
int main() {
  FILE *fp;
  int write_arr[5] = {1, 2, 3, 4, 5};
  int read_arr[5];
  int i;
  fp = fopen("array.bin", "wb");
  fwrite(write_arr, sizeof(int), 5, fp);
  fclose(fp);
  fp = fopen("array.bin", "rb");
  fread(read_arr, sizeof(int), 5, fp);
  fclose(fp);
  for (i = 0; i < 5; i++) {
    printf("%d ", read_arr[i]);
  }
  printf("\n");
  return 0;
}
```

Output:

```
1 2 3 4 5
```

Note:

```
Write entire array with single
fwrite. Read entire array with
single fread. Efficient for large
arrays. Binary format.
```

Struct Array I/O

Program 17:

```c
#include <stdio.h>
#include <string.h>
struct Student {
  int id;
  char name[20];
  float grade;
};
int main() {
  FILE *fp;
  struct Student s[2] = {
    {1, "Alice", 85.5},
    {2, "Bob", 90.0}
  };
  struct Student read_s[2];
  int i;
  fp = fopen("students.bin", "wb");
  fwrite(s, sizeof(struct Student), 2, fp);
  fclose(fp);
  fp = fopen("students.bin", "rb");
  fread(read_s, sizeof(struct Student),
    2, fp);
  fclose(fp);
  for (i = 0; i < 2; i++) {
    printf("%d %s %.1f\n", read_s[i].id,
      read_s[i].name, read_s[i].grade);
  }
  return 0;
}
```

Output:

```
1 Alice 85.5
2 Bob 90.0
```

Note:

Write array of structs to binary file. Read back into array. All data preserved exactly including strings and floats.

Random Access Update

Program 18:

```c
#include <stdio.h>
struct Record {
  int id;
  int value;
};
int main() {
  FILE *fp;
  struct Record r;
  fp = fopen("records.bin", "r+b");
  if (fp == NULL) return 1;
  fseek(fp, 1 * sizeof(struct Record),
    SEEK_SET);
  fread(&r, sizeof(struct Record), 1, fp);
  printf("Old: ID=%d, Val=%d\n",
    r.id, r.value);
  r.value = 999;
  fseek(fp, 1 * sizeof(struct Record),
    SEEK_SET);
  fwrite(&r, sizeof(struct Record), 1, fp);
  fclose(fp);
  printf("Updated record 1\n");
  return 0;
}
```

Output:

```
Old: ID=102, Val=67
Updated record 1
```

Note:

Mode "r+b" opens for reading and writing. Seek to specific record, read it, modify, seek back, write. Random access database simulation.

Text File Processing

Program 19:

```c
#include <stdio.h>
#include <ctype.h>
int main() {
  FILE *in, *out;
  int ch;
  in = fopen("input.txt", "r");
  if (in == NULL) return 1;
  out = fopen("upper.txt", "w");
  if (out == NULL) {
    fclose(in);
    return 1;
  }
  while ((ch = fgetc(in)) != EOF) {
    fputc(toupper(ch), out);
  }
  fclose(in);
  fclose(out);
  printf("File converted to uppercase\n");
  return 0;
}
```

Output:

```
File converted to uppercase
```

If input.txt has:

```
hello world
```

upper.txt has:

```
HELLO WORLD
```

Note:

```
Read from input file, transform
each character, write to output.
Text processing pipeline.
```

CSV File Handling

Program 20:

```c
#include <stdio.h>
int main() {
  FILE *fp;
  int id, age;
  char name[30];
  float salary;
  fp = fopen("data.csv", "w");
  fprintf(fp, "ID,Name,Age,Salary\n");
  fprintf(fp, "1,Alice,25,50000.5\n");
  fprintf(fp, "2,Bob,30,60000.0\n");
  fclose(fp);
  fp = fopen("data.csv", "r");
  fscanf(fp, "ID,Name,Age,Salary\n");
  while (fscanf(fp, "%d,%[^,],%d,%f\n",
    &id, name, &age, &salary) == 4) {
    printf("ID:%d Name:%s Age:%d Sal:%.1f\n",
      id, name, age, salary);
  }
  fclose(fp);
  return 0;
}
```

Output:

```
ID:1 Name:Alice Age:25 Sal:50000.5
ID:2 Name:Bob Age:30 Sal:60000.0
```

Note:

```
CSV (Comma-Separated Values) format.
Write with fprintf using commas.
Read with fscanf using format
%[^,] to read until comma.
```