**Example 1:**

#include <stdio.h>

#include <stdlib.h>

int main() {

FILE \*fptr;

char str[100];

// Writing to a file

fptr = fopen("my\_file.txt", "w");

if (fptr == NULL) {

printf("Error opening file!\n");

exit(1);

}

printf("Enter a string to write to the file: ");

fgets(str, 100, stdin);

fprintf(fptr, "%s", str);

fclose(fptr);

// Reading from the file and printing to the screen

fptr = fopen("my\_file.txt", "r");

if (fptr == NULL) {

printf("Error opening file!\n");

exit(1);

}

printf("Contents of the file:\n");

while (fgets(str, 100, fptr) != NULL) {

printf("%s", str);

}

fclose(fptr);

return 0;

}



**Example 2:**

#include <iostream>

#include <fcntl.h>

#include <unistd.h>

using namespace std;

int main() {

const char \*filename = "my\_file.txt";

char buffer[100];

// Writing to a file

int fd = open(filename, O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

if (fd == -1) {

cerr << "Error opening file for writing!" << endl;

return 1;

}

cout << "Enter a string to write to the file: ";

cin.getline(buffer, 100);

write(fd, buffer, strlen(buffer));

close(fd);

// Reading from the file and printing to the screen

fd = open(filename, O\_RDONLY);

if (fd == -1) {

cerr << "Error opening file for reading!" << endl;

return 1;

}

cout << "Contents of the file:\n";

int bytesRead;

while ((bytesRead = read(fd, buffer, sizeof(buffer))) > 0) {

write(STDOUT\_FILENO, buffer, bytesRead);

}

close(fd);

return 0;

}

**Example 3:**

#include <iostream>

#include <fstream>

#include <string>

#include <cerrno>

#include <cstring>

void handleError(const std::string& action) {

std::cerr << "Error during " << action << ": " << std::strerror(errno) << std::endl;

exit(EXIT\_FAILURE);

}

int main() {

std::string filename = "example.txt";

std::string userContent;

// Step 1: Create or open the file

std::ofstream outFile(filename, std::ios::app);

if (!outFile.is\_open()) {

handleError("opening the file for writing");

}

// Step 2: Get user input and write to the file

std::cout << "Enter text to write to the file: ";

std::getline(std::cin, userContent);

outFile << userContent << std::endl;

if (outFile.fail()) {

handleError("writing to the file");

}

outFile.close();

// Step 3: Open the file for reading

std::ifstream inFile(filename);

if (!inFile.is\_open()) {

handleError("opening the file for reading");

}

// Step 4: Read and display the content of the file

std::string fileContent;

std::cout << "File contents:\n";

while (std::getline(inFile, fileContent)) {

std::cout << fileContent << std::endl;

}

if (inFile.fail() && !inFile.eof()) {

handleError("reading from the file");

}

inFile.close();

return 0;

}



**Example 4:**

#include <iostream>

#include <fcntl.h>

#include <unistd.h>

#include <cstring>

#include <errno.h>

void copyFile(const char \*sourcePath, const char \*destPath) {

int srcFD = open(sourcePath, O\_RDONLY);

if (srcFD == -1) {

std::cerr << "Error opening source file: " << strerror(errno) << std::endl;

return;

}

int destFD = open(destPath, O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

if (destFD == -1) {

std::cerr << "Error opening destination file: " << strerror(errno) << std::endl;

close(srcFD);

return;

}

char buffer[1024];

ssize\_t bytesRead;

while ((bytesRead = read(srcFD, buffer, sizeof(buffer))) > 0) {

if (write(destFD, buffer, bytesRead) != bytesRead) {

std::cerr << "Error writing to destination file: " << strerror(errno) << std::endl;

close(srcFD);

close(destFD);

return;

}

}

if (bytesRead == -1) {

std::cerr << "Error reading from source file: " << strerror(errno) << std::endl;

}

close(srcFD);

close(destFD);

}

int main(int argc, char \*argv[]) {

if (argc != 3) {

std::cerr << "Usage: " << argv[0] << " <source\_file> <destination\_file>" << std::endl;

return 1;

}

copyFile(argv[1], argv[2]);

return 0;

}



**Example 5:**

#include <iostream>

#include <fcntl.h>

#include <unistd.h>

#include <cstring>

#include <vector>

#include <algorithm>

void reverseFile(const char \*filePath) {

int fd = open(filePath, O\_RDWR);

if (fd == -1) {

std::cerr << "Error opening file: " << strerror(errno) << std::endl;

return;

}

off\_t fileSize = lseek(fd, 0, SEEK\_END);

if (fileSize == -1) {

std::cerr << "Error determining file size: " << strerror(errno) << std::endl;

close(fd);

return;

}

std::vector<char> buffer(fileSize);

lseek(fd, 0, SEEK\_SET);

ssize\_t bytesRead = read(fd, buffer.data(), fileSize);

if (bytesRead == -1) {

std::cerr << "Error reading file: " << strerror(errno) << std::endl;

close(fd);

return;

}

std::vector<std::string> lines;

std::string currentLine;

for (char c : buffer) {

if (c == '\n') {

if (!currentLine.empty()) {

std::reverse(currentLine.begin(), currentLine.end());

lines.push\_back(currentLine);

currentLine.clear();

}

} else {

currentLine += c;

}

}

if (!currentLine.empty()) {

std::reverse(currentLine.begin(), currentLine.end());

lines.push\_back(currentLine);

}

lseek(fd, 0, SEEK\_SET);

for (const std::string &line : lines) {

if (write(fd, line.c\_str(), line.size()) == -1) {

std::cerr << "Error writing to file: " << strerror(errno) << std::endl;

close(fd);

return;

}

if (write(fd, "\n", 1) == -1) {

std::cerr << "Error writing newline to file: " << strerror(errno) << std::endl;

close(fd);

return;

}

}

if (ftruncate(fd, lseek(fd, 0, SEEK\_CUR)) == -1) {

std::cerr << "Error truncating file: " << strerror(errno) << std::endl;

}

close(fd);

}

int main(int argc, char \*argv[]) {

if (argc != 2) {

std::cerr << "Usage: " << argv[0] << " <file\_path>" << std::endl;

return 1;

}

reverseFile(argv[1]);

return 0;

}



**Example 6:**

#include <iostream>

#include <fcntl.h>

#include <unistd.h>

#include <cstring>

#include <string>

#include <vector>

const int BUFFER\_SIZE = 1024;

void mergeSortedFiles(const char \*file1Path, const char \*file2Path, const char \*outputFilePath) {

int fd1 = open(file1Path, O\_RDONLY);

if (fd1 == -1) {

std::cerr << "Error opening file1: " << strerror(errno) << std::endl;

return;

}

int fd2 = open(file2Path, O\_RDONLY);

if (fd2 == -1) {

std::cerr << "Error opening file2: " << strerror(errno) << std::endl;

close(fd1);

return;

}

int fdOut = open(outputFilePath, O\_WRONLY | O\_CREAT | O\_TRUNC, 0666);

if (fdOut == -1) {

std::cerr << "Error opening output file: " << strerror(errno) << std::endl;

close(fd1);

close(fd2);

return;

}

std::string line1, line2;

char buffer[BUFFER\_SIZE];

ssize\_t bytesRead1, bytesRead2;

bool hasLine1 = false, hasLine2 = false;

auto readLine = [&](int fd, std::string &line) -> ssize\_t {

line.clear();

while ((bytesRead1 = read(fd, buffer, 1)) > 0) {

if (buffer[0] == '\n') {

break;

}

line += buffer[0];

}

return bytesRead1;

};

if ((bytesRead1 = readLine(fd1, line1)) > 0) hasLine1 = true;

if ((bytesRead2 = readLine(fd2, line2)) > 0) hasLine2 = true;

while (hasLine1 || hasLine2) {

if (hasLine1 && (!hasLine2 || line1 <= line2)) {

line1 += "\n";

write(fdOut, line1.c\_str(), line1.size());

if ((bytesRead1 = readLine(fd1, line1)) > 0) {

hasLine1 = true;

} else {

hasLine1 = false;

}

} else if (hasLine2) {

line2 += "\n";

write(fdOut, line2.c\_str(), line2.size());

if ((bytesRead2 = readLine(fd2, line2)) > 0) {

hasLine2 = true;

} else {

hasLine2 = false;

}

}

}

close(fd1);

close(fd2);

close(fdOut);

}

int main(int argc, char \*argv[]) {

if (argc != 4) {

std::cerr << "Usage: " << argv[0] << " <sorted\_file1> <sorted\_file2> <output\_file>" << std::endl;

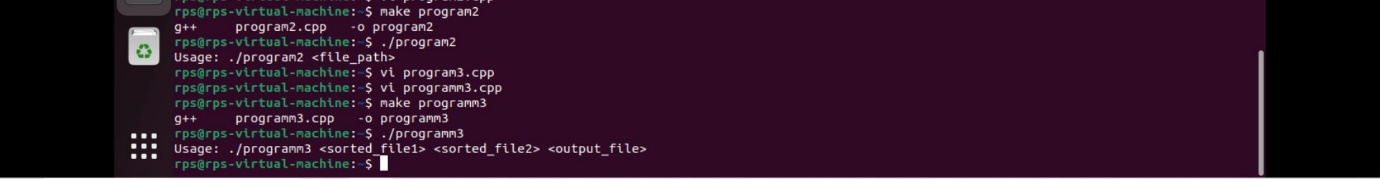
return 1;

}

mergeSortedFiles(argv[1], argv[2], argv[3]);

return 0;

}



**Example 7:**

#include <iostream>

#include <unistd.h>

#include <sys/wait.h>

int main() {

pid\_t pid = fork();

if (pid < 0) {

std::cerr << "Fork failed!" << std::endl;

return 1;

} else if (pid == 0) {

std::cout << "Child Process" << std::endl;

} else {

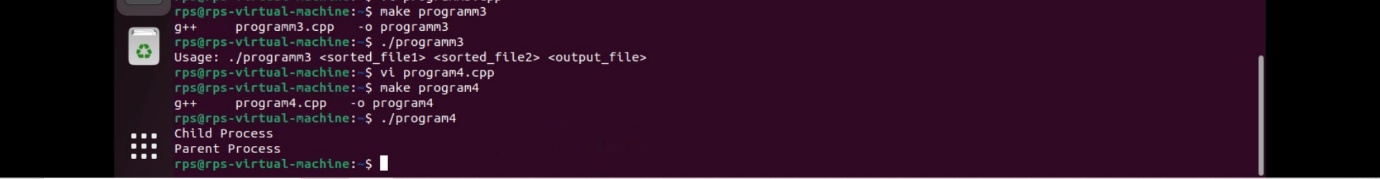
wait(NULL);

std::cout << "Parent Process" << std::endl;

}

return 0;

}



**Example 8:**

#include <iostream>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <cstring>

int main(int argc, char\* argv[]) {

if (argc < 2) {

std::cerr << "Usage: " << argv[0] << " <command>" << std::endl;

return 1;

}

pid\_t pid = fork();

if (pid < 0) {

std::cerr << "Fork failed!" << std::endl;

return 1;

} else if (pid == 0) {

if (execlp(argv[1], argv[1], (char\*)NULL) == -1) {

std::cerr << "Command execution failed: " << strerror(errno) << std::endl;

return 1;

}

} else {

wait(NULL);

}

return 0;

}



**Example 9:**

#include <iostream>

#include <unistd.h>

#include <cstring>

#include <sys/types.h>

#include <sys/wait.h>

int main() {

int pipefd[2];

pid\_t pid;

const char\* message = "Hello from the parent process!";

char buffer[256];

if (pipe(pipefd) == -1) {

std::cerr << "Pipe failed!" << std::endl;

return 1;

}

pid = fork();

if (pid < 0) {

std::cerr << "Fork failed!" << std::endl;

return 1;

} else if (pid == 0) {

close(pipefd[1]);

ssize\_t bytesRead = read(pipefd[0], buffer, sizeof(buffer) - 1);

if (bytesRead > 0) {

buffer[bytesRead] = '\0';

std::cout << "Child process received: " << buffer << std::endl;

} else {

std::cerr << "Read failed!" << std::endl;

}

close(pipefd[0]);

} else {

close(pipefd[0]);

write(pipefd[1], message, strlen(message));

close(pipefd[1]);

wait(NULL);

}

return 0;

}



**Example 10:**

#include <iostream>

#include <fcntl.h>

#include <unistd.h>

#include <cstring>

bool isDelimiter(char ch) {

return ch == ' ' || ch == '\n' || ch == '\t' || ch == '\r' || ch == '\v' || ch == '\f';

}

int main(int argc, char\* argv[]) {

if (argc != 2) {

std::cerr << "Usage: " << argv[0] << " <filename>" << std::endl;

return 1;

}

const char\* filename = argv[1];

int fd = open(filename, O\_RDONLY);

if (fd < 0) {

std::cerr << "Error: Could not open file " << filename << std::endl;

return 1;

}

const size\_t BUFFER\_SIZE = 1024;

char buffer[BUFFER\_SIZE];

ssize\_t bytesRead;

int wordCount = 0;

bool inWord = false;

while ((bytesRead = read(fd, buffer, BUFFER\_SIZE)) > 0) {

for (ssize\_t i = 0; i < bytesRead; ++i) {

if (isDelimiter(buffer[i])) {

if (inWord) {

wordCount++;

inWord = false;

}

} else {

inWord = true;

}

}

}

if (inWord) {

wordCount++;

}

if (bytesRead < 0) {

std::cerr << "Error: Could not read file " << filename << std::endl;

close(fd);

return 1;

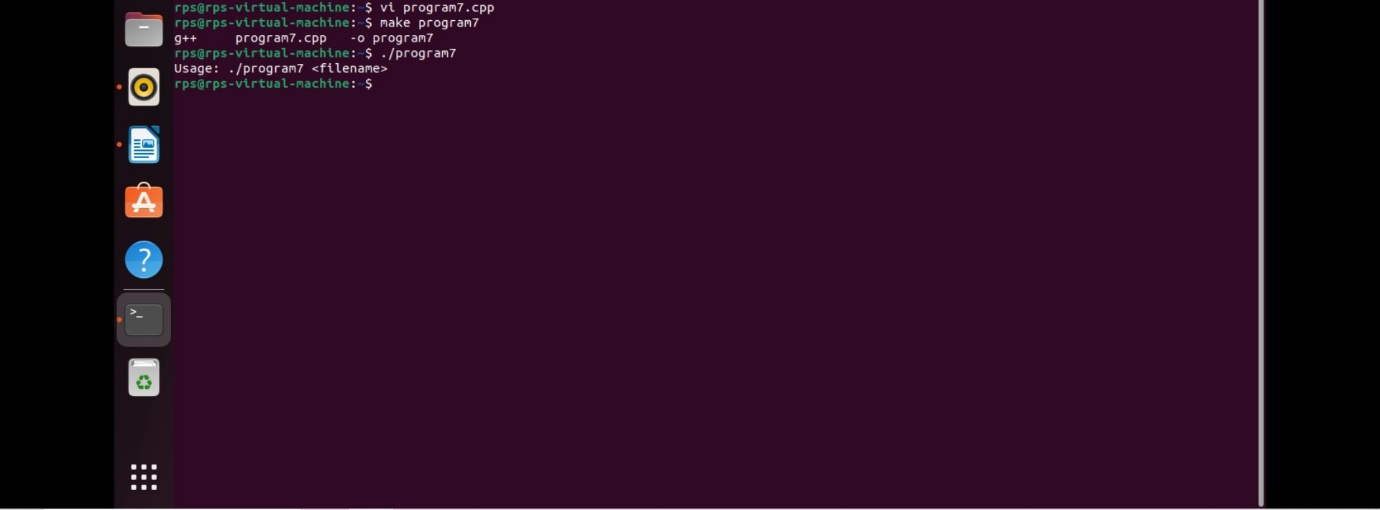
}

close(fd);

std::cout << "The file contains " << wordCount << " words." << std::endl;

return 0;

}



**Example 11:**

#include <iostream>

#include <fstream>

#include <string>

#include <iomanip>

void formatUptime(double uptimeSeconds) {

int days = uptimeSeconds / (24 \* 3600);

uptimeSeconds = uptimeSeconds - (days \* 24 \* 3600);

int hours = uptimeSeconds / 3600;

uptimeSeconds = uptimeSeconds - (hours \* 3600);

int minutes = uptimeSeconds / 60;

uptimeSeconds = uptimeSeconds - (minutes \* 60);

int seconds = uptimeSeconds;

std::cout << "Uptime: "

<< days << " days, "

<< hours << " hours, "

<< minutes << " minutes, "

<< seconds << " seconds"

<< std::endl;

}

int main() {

std::ifstream uptimeFile("/proc/uptime");

if (!uptimeFile) {

std::cerr << "Error: Could not open /proc/uptime" << std::endl;

return 1;

}

double uptimeSeconds;

uptimeFile >> uptimeSeconds;

uptimeFile.close();

formatUptime(uptimeSeconds);

return 0;

}



**Example 12:**

#include <iostream>

#include <cstring>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#define PORT 8080

#define BACKLOG 5

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

const char \*message = "Hello, client!";

server\_fd = socket(AF\_INET, SOCK\_STREAM, 0);

if (server\_fd == 0) {

perror("Socket failed");

exit(EXIT\_FAILURE);

}

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(PORT);

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("Bind failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

if (listen(server\_fd, BACKLOG) < 0) {

perror("Listen failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

std::cout << "Server is listening on port " << PORT << "..." << std::endl;

new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t \*)&addrlen);

if (new\_socket < 0) {

perror("Accept failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

send(new\_socket, message, strlen(message), 0);

std::cout << "Message sent to client: " << message << std::endl;

close(new\_socket);

close(server\_fd);

std::cout << "Connection closed." << std::endl;

return 0;

}

**Example 13:**

#include <iostream>

#include <cstring>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define PORT 8080

#define SERVER\_IP "127.0.0.1"

int main() {

int sock = 0;

struct sockaddr\_in serv\_addr;

const char \*message = "Hi from client!";

char buffer[1024] = {0};

sock = socket(AF\_INET, SOCK\_STREAM, 0);

if (sock < 0) {

perror("Socket creation error");

return -1;

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

if(inet\_pton(AF\_INET, SERVER\_IP, &serv\_addr.sin\_addr) <= 0) {

perror("Invalid address/ Address not supported");

return -1;

}

if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) {

perror("Connection failed");

return -1;

}

send(sock, message, strlen(message), 0);

std::cout << "Message sent to server: " << message << std::endl;

int valread = recv(sock, buffer, 1024, 0);

if (valread > 0) {

std::cout << "Message received from server: " << buffer << std::endl;

} else {

perror("Receive failed");

}

close(sock);

std::cout << "Connection closed." << std::endl;

return 0;

}

