

**Question bank Pattern**

**[subject code] [ subject name]**

**Unit 5 [LIBRARY IMPLEMENTATION & COMMUNICATION MODULE WITH C]**

**PART A (10x2)**

Question 1: What is the purpose of multiple file compilation in C programming?

Answer 1: Multiple file compilation in C allows the program to be split into multiple source code files. This promotes modularity, reusability, and ease of maintenance. Each source file can be compiled separately and linked together to create the final executable.

Question 2: Explain the difference between static libraries and dynamic libraries in C.

Answer 2:

* Static Libraries: Static libraries are linked at compile-time, and the library code is copied into the executable. This results in a larger executable but does not require the presence of the library at runtime.
* Dynamic Libraries: Dynamic libraries are linked at runtime, and the library code is loaded into memory when the program starts. This results in smaller executables but requires the library to be present on the system.

Question 3: What is UART communication in C? How is it used for serial communication?

Answer 3: UART (Universal Asynchronous Receiver/Transmitter) communication in C is a commonly used method for serial communication with external devices or other systems. It involves sending and receiving data one bit at a time over a physical wire or wireless connection, typically using a UART hardware module. UART communication is widely used for tasks like interfacing with sensors, microcontrollers, and communication between computers and peripherals.

Question 4: Explain the steps involved in setting up UART communication in C for sending and receiving data.

Answer 4: Setting up UART communication in C involves the following steps:

1. Configure the UART hardware module (baud rate, data bits, stop bits, etc.).
2. Initialize UART communication using appropriate library functions.
3. Send data by writing to the UART transmit buffer.
4. Receive data by reading from the UART receive buffer.

Question 5: What is terminal programming in C? How is it used for interactive communication with users or external devices?

Answer 5: Terminal programming in C involves interacting with a command-line interface or terminal window. It allows a program to receive user input, display output, and communicate with users interactively. Terminal programming is commonly used for creating command-line applications, text-based games, and configuring embedded systems.

Question 6: Explain the purpose of the #include directive in C.

Answer 6: The #include directive in C is used to include header files in a program. Header files contain function prototypes, constants, and declarations that are necessary for using functions and libraries. It helps in organizing code, promoting modularity, and ensuring that the required declarations are available.

Question 7: What is the role of the linker in the compilation process of a C program with multiple files?

Answer 7: The linker in C is responsible for combining multiple object files and libraries generated during the compilation process into a single executable file. It resolves external references, such as function calls, across different source files and ensures that the program can run seamlessly.

Question 8: What are the advantages of using dynamic libraries (shared libraries) over static libraries in C?

Answer 8: Advantages of dynamic libraries:

1. Smaller executables: Dynamic libraries result in smaller executable files.
2. Shared resource: Multiple programs can use the same dynamic library, saving memory.
3. Easy updates: Libraries can be updated without recompiling programs.
4. Encapsulation: Libraries can encapsulate functionality and hide implementation details.

Question 9: What is a header file in C, and why is it used?

Answer 9: A header file in C is a file containing declarations, function prototypes, and other necessary information for using functions, constants, and structures defined in other source files or libraries. It allows code to be organized, promotes modularity, and ensures that the necessary declarations are available to the compiler.

Question 10: Explain the concept of dynamic linking in C and its advantages.

Answer 10: Dynamic linking in C involves linking external libraries at runtime. Advantages include:

1. Smaller executables: Only necessary library code is loaded.
2. Updates: Libraries can be updated independently.
3. Shared resources: Multiple programs can use the same library, saving memory.
4. Versioning: Different versions of a library can coexist.

**PART A (2x13)**

Question 1: Create a C program that uses a static library to perform arithmetic operations (addition, subtraction, multiplication, and division). Demonstrate how to create and use a static library for these operations.

Answer:

// arithmetic.h

#ifndef ARITHMETIC\_H

#define ARITHMETIC\_H

int add(int a, int b);

int subtract(int a, int b);

int multiply(int a, int b);

int divide(int a, int b);

#endif

// arithmetic.c

#include "arithmetic.h"

int add(int a, int b) {

return a + b;

}

int subtract(int a, int b) {

return a - b;

}

int multiply(int a, int b) {

return a \* b;

}

int divide(int a, int b) {

if (b == 0) {

return -1; // Error: Division by zero

}

return a / b;

}

// main.c

#include <stdio.h>

#include "arithmetic.h"

int main() {

int x = 10, y = 5;

printf("Addition: %d\n", add(x, y));

printf("Subtraction: %d\n", subtract(x, y));

printf("Multiplication: %d\n", multiply(x, y));

printf("Division: %d\n", divide(x, y));

return 0;

}

Question 2: Create a C program that uses a dynamic library to perform string manipulation functions (e.g., string length, concatenation). Demonstrate how to create and use a dynamic library for these functions.

Answer:

// stringmanip.h

#ifndef STRINGMANIP\_H

#define STRINGMANIP\_H

int string\_length(const char \*str);

void string\_concat(char \*dest, const char \*src);

#endif

// stringmanip.c

#include "stringmanip.h"

#include <string.h>

int string\_length(const char \*str) {

return strlen(str);

}

void string\_concat(char \*dest, const char \*src) {

strcat(dest, src);

}

// main.c

#include <stdio.h>

#include <stdlib.h>

#include "stringmanip.h"

int main() {

char str1[50] = "Hello, ";

char str2[] = "World!";

printf("String 1: %s\n", str1);

printf("String 2: %s\n", str2);

printf("String 1 Length: %d\n", string\_length(str1));

string\_concat(str1, str2);

printf("Concatenated String: %s\n", str1);

return 0;

}

Question 3: Simulate UART communication in a C program by sending and receiving data between two virtual UART devices (e.g., terminals). Implement a simple chat program using UART-like communication.

Answer:

// uart\_simulation.h

#ifndef UART\_SIMULATION\_H

#define UART\_SIMULATION\_H

void uart\_send(const char \*data);

void uart\_receive(char \*data, int max\_length);

#endif

// uart\_simulation.c

#include "uart\_simulation.h"

#include <stdio.h>

#include <string.h>

void uart\_send(const char \*data) {

printf("Transmitting: %s\n", data);

}

void uart\_receive(char \*data, int max\_length) {

printf("Waiting for data...\n");

fgets(data, max\_length, stdin);

}

// main.c

#include <stdio.h>

#include "uart\_simulation.h"

int main() {

char message[100];

printf("Terminal 1:\n");

printf("Enter a message to send: ");

fgets(message, sizeof(message), stdin);

uart\_send(message);

printf("\nTerminal 2:\n");

uart\_receive(message, sizeof(message));

printf("Received Message: %s", message);

return 0;

}

Question 4: Create a C program that acts as a terminal program for file manipulation. Implement commands for listing files in a directory, creating new files, reading and writing to files, and deleting files.

Answer:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void list\_files() {

system("ls -l");

}

void create\_file(const char \*filename) {

FILE \*file = fopen(filename, "w");

if (file == NULL) {

perror("File creation failed");

return;

}

fclose(file);

printf("File '%s' created successfully.\n", filename);

}

void read\_file(const char \*filename) {

FILE \*file = fopen(filename, "r");

if (file == NULL) {

perror("File opening failed");

return;

}

char buffer[100];

while (fgets(buffer, sizeof(buffer), file) != NULL) {

printf("%s", buffer);

}

fclose(file);

}

void write\_file(const char \*filename, const char \*content) {

FILE \*file = fopen(filename, "a");

if (file == NULL) {

perror("File opening failed");

return;

}

fprintf(file, "%s", content);

fclose(file);

printf("Data written to '%s' successfully.\n", filename);

}

void delete\_file(const char \*filename) {

if (remove(filename) == 0) {

printf("File '%s' deleted successfully.\n", filename);

} else {

perror("File deletion failed");

}

}

int main() {

int choice;

char filename[50];

char content[100];

while (1) {

printf("\nFile Manipulation Menu:\n");

printf("1. List Files\n");

printf("2. Create File\n");

printf("3. Read File\n");

printf("4. Write to File\n");

printf("5. Delete File\n");

printf("6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

list\_files();

break;

case 2:

printf("Enter filename: ");

scanf("%s", filename);

create\_file(filename);

break;

case 3:

printf("Enter filename: ");

scanf("%s", filename);

read\_file(filename);

break;

case 4:

printf("Enter filename: ");

scanf("%s", filename);

printf("Enter content: ");

getchar(); // Clear newline character from previous input

fgets(content, sizeof(content), stdin);

write\_file(filename, content);

break;

case 5:

printf("Enter filename: ");

scanf("%s", filename);

delete\_file(filename);

break;

case 6:

exit(0);

default:

printf("Invalid choice. Try again.\n");

}

}

return 0;

}

Question 5: Create a C program that simulates file transfer over UART-like communication. Implement a sender program that reads a file and sends it in chunks to a receiver program, which then reconstructs the file.

Answer:

**Sender Program:**

// sender.c

#include <stdio.h>

#include <string.h>

#include "uart\_simulation.h"

#define CHUNK\_SIZE 128

int main() {

FILE \*file = fopen("file\_to\_send.txt", "rb");

if (file == NULL) {

perror("File opening failed");

return 1;

}

char buffer[CHUNK\_SIZE];

size\_t bytes\_read;

while ((bytes\_read = fread(buffer, 1, sizeof(buffer), file)) > 0) {

uart\_send(buffer);

}

fclose(file);

return 0;

}

**Receiver Program:**

// receiver.c

#include <stdio.h>

#include <string.h>

#include "uart\_simulation.h"

#define CHUNK\_SIZE 128

int main() {

FILE \*file = fopen("received\_file.txt", "wb");

if (file == NULL) {

perror("File creation failed");

return 1;

}

char buffer[CHUNK\_SIZE];

size\_t bytes\_received;

while (1) {

uart\_receive(buffer, sizeof(buffer));

if (strcmp(buffer, "EOF") == 0) {

break; // End of file transfer

}

fwrite(buffer, 1, strlen(buffer), file);

}

fclose(file);

printf("File received successfully.\n");

return 0;

}