String Pointer

1. Reverse a String

Write a function void reverseString(char *str) that takes a pointer to a string and reverses the string in place.

```
#include <stdio.h>
#include<string.h>

void reverseString(char *str);
int main(){
    char str[] = "To be or not to be";
    printf("Original string : %s \n",str);
    reverseString(str);
    printf("Reversed string :  %s \n",str);

}
void reverseString(char *str){
    int len = strlen(str);
    int start =0;
    int end =len -1;
    while(start <end){
        char temp = str[start];
        str[start] = str[end];
        str[end] = temp;
        start++;
        end--;
    }
}
```

2. Concatenate Two Strings

Implement a function void concatenateStrings(char *dest, const char *src) that appends the source string to the destination string using pointers.

```
#include <stdio.h>
#include <string.h>

void concatenateStrings(char *dest, const char *src);
int main(){
   char dest[] = "The Power of ";
   char src[] = "Lords of Ring ";
   printf("The source string is : %s \n",src);
   printf("The destination string is : %s \n",dest);
   concatenateStrings(dest,src);
   printf("The concatenation of two strings : \n %s ",dest);
}
void concatenateStrings(char *dest, const char *src){
   strcat(dest,src);
}
```

3. String Length

Create a function int stringLength(const char *str) that calculates and returns the length of a string using pointers.

```
#include<stdio.h>
#include <string.h>
int stringLength(const char *str);
int main(){
```

```
    const char str[100];
    int len =0;
    printf("Enter a string ");
    scanf("%s",str);
    len = stringLength(str);

    printf("The length of a given string is %d",len);
    return 0;
}
int stringLength(const char *str){
    int len = strlen(str);
    return len;
}
```

4. Compare Two Strings

Write a function int compareStrings(const char *str1, const char *str2) that compares two strings

lexicographically and returns 0 if they are equal, a positive number if str1 is greater, or a negative number if str2 is greater.

```
#include <stdio.h>
# include <string.h>
int compareStrings(const char *str1, const char *str2) ;

int main(){
    const char str1[100];
    const char str2[100];
    printf("Enter the first string: ");
    scanf("%s", str1);
    printf("Enter the second string: ");
```

```
    scanf("%s", str2);

    int result = compareStrings(str1,str2);
    if(result == 0){
        printf("The strings are equal ");
    }
    else if(result >0){
        printf("The first string is lexicographically greater.\n");
    }
    else {
        printf("The second string is lexicographically greater.\n");
    }
    return 0;
}
int compareStrings(const char *str1, const char *str2){
    int result = strcmp(str1,str2);
    return result;
}
```

5. Find Substring

Implement char* findSubstring(const char *str, const char *sub) that returns a pointer to the first occurrence of the substring sub in the string str, or NULL if the substring is not found.

```
#include <stdio.h>
#include <string.h>

char* findSubstring(char *str, char *sub);
int main(){
    char str[100];
```

```
    char sub[100];
    printf("Enter a main string \n");
    fgets(str,sizeof(str),stdin);   // Use fgets to read a full line (including spaces)
    str[strcspn(str, "\n")] = '\0';  // Remove the newline character added by fgets
    printf("Enter a sub string \n");
    fgets(sub,sizeof(sub),stdin);
    sub[strcspn(sub, "\n")] = '\0';
    char *result = findSubstring(str,sub);
    if(result != '\0'){
        printf("Substring found at position: %d \n ",result-str);
    }else{
        printf("Not found \n");
    }
    return 0;
}
char* findSubstring( char *str,  char *sub){
    return strstr(str,sub);

}
```

6. Replace Character in String

Write a function void replaceChar(char *str, char oldChar, char newChar) that replaces all occurrences of oldChar with newChar in the given string.

```
#include <stdio.h>
#include <string.h>

void replaceChar(char *str, char oldChar, char newChar);
int main(){
    char str[100];
```

```
    char oldChar,newChar;


    printf("Enter a string: \n");
    fgets(str,sizeof(str),stdin);
    str[strcspn(str,"\n")] ='\0';
    printf("Enter the character to be replaced: \n");
    scanf("%c", &oldChar);
    getchar();    // To consume the newline character left by scanf
    printf("Enter the new character: \n");
    scanf("%c", &newChar);
    replaceChar(str,oldChar,newChar);
    printf("Updated string: \n %s \n",str);
    return 0;


}
void replaceChar(char *str, char oldChar, char newChar){
    int len = strlen(str);
    for(int i = 0; i < len; i++){
        if(str[i] == oldChar){
            str[i] = newChar;
        }
    }


}
```

7. Copy String

Create a function void copyString(char *dest, const char *src) that copies the content of the source string src to the destination string dest.

```
#include <stdio.h>
```

```
#include <string.h>

void copyString(char *dest, const char *src);

int main(){
   char dest[] = "World of Vampires";
   char src[] = "It will be thrilling";
   copyString(dest,src);
   printf("The copied string is : %s \n",dest);
   return 0;
}
void copyString(char *dest, const char *src){
   strcpy(dest,src);
}
```

8. Count Vowels in a String

Implement int countVowels(const char *str) that counts and returns the number of vowels in a given string.

```
#include <stdio.h>
#include <string.h>

int countVowels(const char *str);
int main(){
   const char str [100];
   printf("Enter the string \n");
   scanf("%s",str);
   getchar();
   int count = countVowels(str);
   printf("Number of vowels in string is %d ",count);
```

```
    return 0;
}

int countVowels(const char *str){

    int count =0;
    int len = strlen(str);

    for(int i=0;i<len;i++){
        if (str[i] == 'a' || str[i] == 'e' || str[i] == 'i' || str[i] == 'o' || str[i] == 'u' || str[i] == 'A' || str[i] == 'E'|| str[i] == 'I'|| str[i] == 'O'|| str[i] == 'U') {
            count++;
        }
    }
    return count;
}
```

9. Check Palindrome

Write a function int isPalindrome(const char *str) that checks if a given string is a palindrome and returns 1 if true, otherwise 0.

```
#include <stdio.h>
#include<string.h>

int isPalindrome(const char *str);
int main(){
    const char str[] = "To be or not to be";
    printf("Enter a String \n");
    scanf("%s",str);
    printf(str);
    if(isPalindrome(str)){
```

```
        printf("\nThe string is a palindrome.\n");
    } else {
        printf("\nThe string is not a palindrome.\n");
    }

    return 0;
}

int isPalindrome(const char *str){
    int len = strlen(str);
    int start =0;
    int end =len-1;
    while(start<end){
        if(str[start] != str[end]){
            return 0;
        }
        start++;
        end--;
    }
    return 1;

}
```

10. Tokenize String

Create a function void tokenizeString(char *str, const char *delim, void (*processToken)(const char *)) that tokenizes the string str using delimiters in delim, and for each token, calls processToken.

```
#include <stdio.h>
#include <string.h>
```

```
void tokenizeString(char *str, const char *delim, void (*processToken )(const char *));
void processToken(const char *token);
int main(){
    char str[] = "Hello, witch how r u ?";
    const char *delim = " ,?";

    tokenizeString(str,delim,processToken);
    return 0;
}

void processToken(const char *token){
    printf("Process token: %s \n",token);
}

void tokenizeString(char *str, const char *delim ,void (*processToken) (const char *)){
    char *token = strtok(str,delim);
    while(token != NULL){
        processToken(token);
        token = strtok(NULL,delim);
    }
}
```

Double Pointer

Double Pointers

1. Swap Two Numbers Using Double Pointers

Write a function void swap(int **a, int **b) that swaps the values of two integer pointers using double pointers.

```
#include <stdio.h>
void swap(int **a, int **b);

int main(){
   int x =2,y =5;
   int *ptr1 =&x,*ptr2 =&y;
   printf("Before swapping %d %d \n",*ptr1,*ptr2);
   swap(&ptr1,&ptr2);
   printf("After swapping %d %d \n",*ptr1,*ptr2);
   return 0;
}
void swap(int **a,int **b){
  int *temp = *a;
  *a =*b;
  *b = temp;
}
```

2. Dynamic Memory Allocation Using Double Pointer

Implement a function void allocateArray(int **arr, int size) that dynamically allocates memory for an array of integers using a double pointer.

```
#include <stdio.h>
#include <stdlib.h>
void allocateArray(int **arr, int size);
```

```
int main(){
   int size =5;
   int *arr = NULL;
   allocateArray(&arr,size);

   printf("Filling and displaying the array:\n");
   for (int i = 0; i < size; i++) {
      arr[i] = i + 1;  // Assign values
      printf("Array[%d] ==> %d\n", i, arr[i]);
   }
   free(arr);
   printf("Memory dellocated");
   return 0;

}
void allocateArray(int **arr, int size){
   *arr = (int *)malloc(size * sizeof(int));
}
```

3. Modify a String Using Double Pointer

Write a function void modifyString(char **str) that takes a double pointer to a string, dynamically allocates a new string, assigns it to the pointer, and modifies the original string.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void modifyString(char **str);
```

```
int main(){
   char *str = (char *)malloc(50 * sizeof(char));
   strcpy(str, "The Original string") ;
   printf(" Before modification %s \n",str);
   modifyString(&str);
   printf(" After modification %s \n",str);
   free(str);
   return 0;
}
void modifyString(char **str){
   *str  = (char *) realloc(*str,50 *sizeof(char));

   strcpy(*str,"I'm modified string");
}
```

4. Pointer to Pointer Example

Create a simple program that demonstrates how to use a pointer to a pointer to access and modify the value of an integer.

```
# include <stdio.h>

int main(){
   int x =10,y=50;
   int *ptr = &x;
   int **ptr2 = &ptr;
   printf("Before modification %d \n",**ptr2);
   *ptr2 =&y;
   printf("After modification %d \n",**ptr2);

}
```

5. 2D Array Using Double Pointer

Write a function int create2DArray(int rows, int cols) that dynamically allocates memory for a 2D array of integers using a double pointer and returns the pointer to the array.**

```
#include <stdio.h>
#include <stdlib.h>

int** create2DArray(int rows, int cols);
void free2DArray(int** array, int rows);

int main() {
    int rows, cols;

    // Input dimensions
    printf("Enter the number of rows: ");
    scanf("%d", &rows);
    printf("Enter the number of columns: ");
    scanf("%d", &cols);

    // Create a 2D array
    int** array = create2DArray(rows, cols);

    // Fill the array with values
    printf("Enter elements for the 2D array:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("Element at (%d, %d): ", i, j);
            scanf("%d", &array[i][j]);
        }
```

```
    }

    // Print the 2D array
    printf("The 2D array is:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", array[i][j]);
        }
        printf("\n");
    }

    // Free the allocated memory
    free2DArray(array, rows);

    return 0;
}

int** create2DArray(int rows, int cols) {
    // Allocate memory for row pointers
    int** array = (int**)malloc(rows * sizeof(int*));
    // Allocate memory for each row
    for (int i = 0; i < rows; i++) {
        array[i] = (int*)malloc(cols * sizeof(int));
        if (array[i] == NULL) {
            printf("Memory allocation failed for columns.\n");
            exit(1);
        }
    }

    return array;
}
```

```
void free2DArray(int** array, int rows) {
    for (int i = 0; i < rows; i++) {
        free(array[i]);
    }
    free(array);

    printf("Memory deallocated.\n");
}
```

6. Freeing 2D Array Using Double Pointer

Implement a function void free2DArray(int **arr, int rows) that deallocates the memory allocated for a 2D array using a double pointer.

```
#include <stdio.h>
#include <stdlib.h>

void free2DArray(int **arr, int rows);

int main() {
    int rows = 3, cols = 4;

    // Dynamically allocate memory for a 2D array
    int **arr = (int **)malloc(rows * sizeof(int *));
    for (int i = 0; i < rows; i++) {
        arr[i] = (int *)malloc(cols * sizeof(int));
    }

    // Fill the array with values
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
```

```
            arr[i][j] = (i + 1) * (j + 1);
        }
    }

    // Print the 2D array
    printf("2D Array:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }

    // Free the allocated memory
    free2DArray(arr, rows);

    return 0;
}

void free2DArray(int **arr, int rows) {
    for (int i = 0; i < rows; i++) {
        free(arr[i]);
    }

    free(arr);

    printf("Memory deallocated.\n");
}
```

7. Pass a Double Pointer to a Function

Write a function void setPointer(int **ptr) that sets the pointer passed to it to point to a dynamically allocated integer.

```
#include <stdio.h>

#include <stdlib.h>


void setPointer(int **ptr);


int main() {
   int *p = NULL;


   setPointer(&p);


   *p = 42; // Set the dynamically allocated value

   printf("Value in dynamically allocated memory: %d\n", *p);


   free(p); // Free the memory

   return 0;

}


void setPointer(int **ptr) {

   *ptr = (int *)malloc(sizeof(int)); // Allocate memory for an integer

   if (*ptr == NULL) {

      printf("Memory allocation failed.\n");

      exit(1);

   }

}
```

8. Dynamic Array of Strings

Create a function void allocateStringArray(char ***arr, int n) that dynamically allocates memory for an array of n strings using a double pointer.

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>

void allocateStringArray(char ***arr, int n);

int main() {
    char **stringArray;
    int n = 3;

    allocateStringArray(&stringArray, n);

    // Assign and print strings
    for (int i = 0; i < n; i++) {
        snprintf(stringArray[i], 50, "String %d", i + 1); // Assign a string
        printf("String %d: %s\n", i + 1, stringArray[i]);
    }

    // Free memory
    for (int i = 0; i < n; i++) {
        free(stringArray[i]);
    }
    free(stringArray);

    return 0;
}

void allocateStringArray(char ***arr, int n) {
    *arr = (char **)malloc(n * sizeof(char *));
    if (*arr == NULL) {
        printf("Memory allocation for array failed.\n");
        exit(1);
```

```
    }

    for (int i = 0; i < n; i++) {
        (*arr)[i] = (char *)malloc(50 * sizeof(char)); // Allocate space for each string
        if ((*arr)[i] == NULL) {
            printf("Memory allocation for string %d failed.\n", i);
            exit(1);
        }
    }
}
```

9. String Array Manipulation Using Double Pointer

Implement a function void modifyStringArray(char **arr, int n) that modifies each string in an array of strings using a double pointer.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void modifyStringArray(char **arr, int n);

int main() {
    int n = 3;
    char **stringArray = (char **)malloc(n * sizeof(char *));

    // Allocate and initialize strings
    for (int i = 0; i < n; i++) {
        stringArray[i] = (char *)malloc(50 * sizeof(char));
        snprintf(stringArray[i], 50, "Original String %d", i + 1);
```

```
    }

    printf("Before modification:\n");
    for (int i = 0; i < n; i++) {
        printf("%s\n", stringArray[i]);
    }

    modifyStringArray(stringArray, n);

    printf("\nAfter modification:\n");
    for (int i = 0; i < n; i++) {
        printf("%s\n", stringArray[i]);
    }

    // Free memory
    for (int i = 0; i < n; i++) {
        free(stringArray[i]);
    }
    free(stringArray);

    return 0;
}

void modifyStringArray(char **arr, int n) {
    for (int i = 0; i < n; i++) {
        snprintf(arr[i], 50, "Modified String %d", i + 1); // Modify each string
    }
}
```

Function Pointer

Function Pointers

1. Basic Function Pointer Declaration

Write a program that declares a function pointer for a function int add(int, int) and uses it to call the function and print the result.

```
#include <stdio.h>

int add(int a,int b);

int main(){
    int (*funptr)(int,int);
    funptr = &add;
    int res = funptr(15,20);
    printf("Addition => %d",res);
}

int add(int a,int b){
    return a+b;
}
```

2. Function Pointer as Argument

Implement a function void performOperation(int (*operation)(int, int), int a, int b) that takes a function pointer as an argument and applies it to two integers, printing the result.

```
#include<stdio.h>

void performOperation(int (*operation)(int, int), int a, int b);

int add(int a, int b) {
    return a + b;
}

int multiply(int a, int b) {
    return a * b;
```

```
}

int main(){
    int a =10,b=20;
    performOperation(add,a,b);
    performOperation(multiply,a,b);
    return 0;
}
void performOperation(int (*operation)(int, int), int a, int b){
    int result = operation(a,b);
    printf("The result of the operation is: %d\n", result);

}
```

3. Function Pointer Returning Pointer

Write a program with a function int* max(int *a, int *b) that returns a pointer to the larger of two integers, and use a function pointer to call this function.

```
#include <stdio.h>

int* max(int *a, int *b);

int main(){
    int a=10,b=20;
    int* (*funptr)(int*,int *) =max;
    int *result = funptr(&a,&b);
    printf("The maximum is %d",*result);
    return 0;
```

```
}
int* max(int *a, int *b){
    if(*a>*b){
        return a;
    }
    return b;
}
```

4. Function Pointer with Different Functions

Create a program that defines two functions int add(int, int) and int multiply(int, int) and uses a function pointer to dynamically switch between these functions based on user input.

```
#include <stdio.h>

int add(int ,int);
int multiply(int,int);

int main(){
    int choice,a,b,result;
    int (*ptr)(int,int);
    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);
    printf("Choose an operation:\n");
    printf("1. Addition\n");
    printf("2. Multiplication\n");
    printf("Enter your choice (1/2): ");
    scanf("%d", &choice);
    if(choice ==1){
        ptr = add;
```

```
    }else if(choice ==2){
        ptr =multiply;
    }
    else{
        printf("Invalid choice");
    }
    result = ptr(a,b);

    if(choice ==1){
        printf("The result of addition is: %d\n", result);
    }
    else if (choice == 2) {
        printf("The result of multiplication is: %d\n", result);
    }
    return 0;

}
int add(int a, int b) {
    return a + b;
}

int multiply(int a, int b) {
    return a * b;
}
```

5. Array of Function Pointers

Implement a program that creates an array of function pointers for basic arithmetic operations (addition, subtraction, multiplication, division) and allows the user to select and execute one operation.

```c
#include <stdio.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);

int main() {
    int (*operations[])(int, int) = {add, subtract, multiply, divide};
    int choice, num1, num2, result;

    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    printf("Choose an operation:\n");
    printf("0. Addition\n");
    printf("1. Subtraction\n");
    printf("2. Multiplication\n");
    printf("3. Division\n");
    printf("Enter your choice (0-3): ");
    scanf("%d", &choice);
    if (choice < 0 || choice > 3) {
        printf("Invalid choice!\n");
        return 1;
    }

    if (choice == 3 && num2 == 0) {
        printf("Error: Division by zero is not allowed.\n");
        return 1;
```

```
    }

    result = operations[choice](num1, num2);
    switch (choice) {
        case 0:
            printf("The result of addition is: %d\n", result);
            break;
        case 1:
            printf("The result of subtraction is: %d\n", result);
            break;
        case 2:
            printf("The result of multiplication is: %d\n", result);
            break;
        case 3:
            printf("The result of division is: %d\n", result);
            break;
    }

    return 0;
}

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) {
    return a / b;
}
```

6. Using Function Pointers for Sorting

Write a function void sort(int *arr, int size, int (*compare)(int, int)) that uses a function pointer to compare elements, allowing for both ascending and descending order sorting.

```
#include <stdio.h>
void sort(int *arr, int size, int (*compare)(int, int));
int ascending(int a, int b);
int descending(int a, int b);
void printArray(int *arr, int size);

int main(){
    int arr[]= {8,5,6,2,1,7,4};
    int size = sizeof(arr)/sizeof(arr[0]);
    printf("Sorting in ascending order:\n");
    sort(arr, size, ascending);
    printArray(arr, size);
    printf("\nSorting in descending order:\n");
    sort(arr, size, descending);
    printArray(arr, size);

    return 0;
```

```
}

void sort(int *arr, int size, int (*compare)(int, int)){
    for (int i = 0; i < size - 1; i++) {
        for (int j = i + 1; j < size; j++) {
            if (compare(arr[i], arr[j]) > 0) {
                // Swap arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}

int ascending(int a, int b) {
    return a - b;       // Negative if a < b, 0 if a == b, positive if a > b
}

int descending(int a, int b) {
    return b - a;  // Negative if a > b, 0 if a == b, positive if a < b
}

void printArray(int *arr, int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

7. Callback Function

Create a program with a function void execute(int x, int (*callback)(int)) that applies a callback function to an integer and prints the result. Demonstrate with multiple callback functions (e.g., square, cube).

```
#include <stdio.h>

void execute(int x, int (*callback)(int));
int square(int x);
int cube(int x);

int main() {
    int number = 5;

    printf("Square of %d: ", number);
    execute(number, square);

    printf("Cube of %d: ", number);
    execute(number, cube);

    return 0;
}

// Function to execute callback
void execute(int x, int (*callback)(int)) {
    printf("%d\n", callback(x));
}

// Function to calculate square
```

```
int square(int x) {
    return x * x;
}

// Function to calculate cube
int cube(int x) {
    return x * x * x;
}
```

8. Menu System Using Function Pointers

Implement a simple menu system where each menu option corresponds to a different function, and a function pointer array is used to call the selected function based on user input.

```
#include <stdio.h>

// Function declarations
void option1();
void option2();
void option3();
void option4();

int main() {
    // Array of function pointers
    void (*menu[4])() = {option1, option2, option3, option4};

    int choice;
    printf("Menu:\n");
    printf("1. Option 1\n");
    printf("2. Option 2\n");
```

```
    printf("3. Option 3\n");
    printf("4. Option 4\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    // Check if the choice is valid and call the respective function
    if (choice >= 1 && choice <= 4) {
        menu[choice - 1]();
    } else {
        printf("Invalid choice!\n");
    }

    return 0;
}

// Functions corresponding to each menu option
void option1() {
    printf("You selected Option 1\n");
}

void option2() {
    printf("You selected Option 2\n");
}

void option3() {
    printf("You selected Option 3\n");
}

void option4() {
```

```
    printf("You selected Option 4\n");
}
```

9. Dynamic Function Selection

Write a program where the user inputs an operation symbol (+, -, *, /) and the program uses a function pointer to call the corresponding function.

```
#include <stdio.h>

// Function declarations for operations
int add(int a, int b);
int subtract(int a, int b);
int multiply(int a, int b);
int divide(int a, int b);

int main() {
    int a, b;
    char operator;

    printf("Enter first number: ");
    scanf("%d", &a);
    printf("Enter second number: ");
    scanf("%d", &b);
    printf("Enter operator (+, -, *, /): ");
    getchar();  // To consume newline character left by previous input
    scanf("%c", &operator);

    // Array of function pointers for operations
    int (*operation)(int, int);
```

```
    // Select the appropriate operation based on the operator
    switch(operator) {
        case '+':
            operation = add;
            break;
        case '-':
            operation = subtract;
            break;
        case '*':
            operation = multiply;
            break;
        case '/':
            operation = divide;
            break;
        default:
            printf("Invalid operator!\n");
            return 1;
    }

    // Call the selected operation
    int result = operation(a, b);
    printf("Result: %d\n", result);

    return 0;
}

// Functions for basic arithmetic operations
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 a / b;
    } else {
        printf("Error! Division by zero.\n");
        return 0;
    }
}
```

10. State Machine with Function Pointers

Design a simple state machine where each state is represented by a function, and transitions are handled using function pointers. For example, implement a traffic light system with states like Red, Green, and Yellow.

```
#include <stdio.h>

// State function declarations
void red();
```

```
void green();
void yellow();

// Function pointer declaration for state transitions
void (*state)();

int main() {
    // Initial state is Red
    state = red;

    // Run the state machine in a loop
    for (int i = 0; i < 5; i++) {
        state();  // Call the current state function

        // Transition to the next state
        if (state == red) {
            state = green;
        } else if (state == green) {
            state = yellow;
        } else if (state == yellow) {
            state = red;
        }
    }

    return 0;
}

// Function for Red state
void red() {
```

```
    printf("Red light - Stop\n");
}

// Function for Green state
void green() {
    printf("Green light - Go\n");
}

// Function for Yellow state
void yellow() {
    printf("Yellow light - Caution\n");
}
```