**Exercise-III: Structures, Unions and Pointer**

***Structures***♣

**1. Write a program to define and initialize a structure for a student with attributes such as name,roll number, and marks.**

#include <stdio.h>

struct Student {

char name[50];

int rollNumber;

float marks;

};

int main() {

struct Student student1 = {"John Doe", 101, 85.5};

printf("Student Details:\n");

printf("Name: %s\n", student1.name);

printf("Roll Number: %d\n", student1.rollNumber);

printf("Marks: %.2f\n", student1.marks);

return 0;

}

**2. Write a program to read and display the details of a student using a structure.**

#include <stdio.h>

struct Student {

char name[50];

int rollNumber;

float marks;

};

int main() {

struct Student student;

printf("Enter student details:\n");

printf("Name: ");

gets(student.name);

printf("Roll Number: ");

scanf("%d", &student.rollNumber);

printf("Marks: ");

scanf("%f", &student.marks);

printf("\nStudent Details:\n");

printf("Name: %s\n", student.name);

printf("Roll Number: %d\n", student.rollNumber);

printf("Marks: %.2f\n", student.marks);

return 0;

}

**3. Write a program to find the average marks of a group of students using a structure.**

#include <stdio.h>

struct Student {

char name[50];

int rollNumber;

float marks;

};

int main() {

int n;

printf("Enter the number of students: ");

scanf("%d", &n);

struct Student students[n];

float totalMarks = 0;

for (int i = 0; i < n; i++) {

printf("Enter details for student %d:\n", i + 1);

printf("Name: ");

scanf("%s", students[i].name);

printf("Roll Number: ");

scanf("%d", &students[i].rollNumber);

printf("Marks: ");

scanf("%f", &students[i].marks);

totalMarks += students[i].marks;

}

float averageMarks = totalMarks / n;

printf("\nAverage Marks of the group: %.2f\n", averageMarks);

return 0;

}

**4. Write a program to find the total marks of a student using a structure and a function.**

#include <stdio.h>

struct Student {

char name[50];

int rollNumber;

float marks;

};

float calculateTotalMarks(struct Student student);

int main() {

struct Student student;

printf("Enter student details:\n");

printf("Name: ");

gets(student.name);

printf("Roll Number: ");

scanf("%d", &student.rollNumber);

printf("Marks: ");

scanf("%f", &student.marks);

float totalMarks = calculateTotalMarks(student);

printf("\nTotal Marks of %s: %.2f\n", student.name, totalMarks);

return 0;

}

float calculateTotalMarks(struct Student student) {

return student.marks;

}

**5. Write a program to sort a group of students based on their marks using structures and bubble**

#include <stdio.h>

#include <string.h>

struct Student {

char name[50];

int rollNumber;

float marks;

};

void bubbleSort(struct Student students[], int n);

int main() {

int n;

printf("Enter the number of students: ");

scanf("%d", &n);

struct Student students[n];

for (int i = 0; i < n; i++) {

printf("Enter details for student %d:\n", i + 1);

printf("Name: ");

getchar(); // Consume the newline character left in the buffer

fgets(students[i].name, sizeof(students[i].name), stdin);

students[i].name[strcspn(students[i].name, "\n")] = '\0'; // Remove the newline character

printf("Roll Number: ");

scanf("%d", &students[i].rollNumber);

printf("Marks: ");

scanf("%f", &students[i].marks);

}

bubbleSort(students, n);

printf("\nSorted List of Students based on Marks:\n");

for (int i = 0; i < n; i++) {

printf("Name: %s, Roll Number: %d, Marks: %.2f\n", students[i].name, students[i].rollNumber, students[i].marks);

}

return 0;

}

void bubbleSort(struct Student students[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (students[j].marks > students[j + 1].marks) {

// Swap the students

struct Student temp = students[j];

students[j] = students[j + 1];

students[j + 1] = temp;

}

}

}

}

**6. sort algorithm.**

#include <stdio.h>

void bubbleSort(int arr[], int n);

int main() {

int n;

printf("Enter the number of elements in the array: ");

scanf("%d", &n);

int arr[n];

printf("Enter the elements of the array:\n");

for (int i = 0; i < n; i++) {

scanf("%d", &arr[i]);

}

bubbleSort(arr, n);

printf("Sorted array in ascending order:\n");

for (int i = 0; i < n; i++) {

printf("%d ", arr[i]);

}

return 0;

}

void bubbleSort(int arr[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

// Swap the elements

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

**7. Write a program to define and initialize a structure for a book with attributes such as title, author, and price.**

#include <stdio.h>

struct Book {

char title[100];

char author[100];

float price;

};

int main() {

struct Book book1 = {"The Catcher in the Rye", "J.D. Salinger", 10.99};

printf("Book Details:\n");

printf("Title: %s\n", book1.title);

printf("Author: %s\n", book1.author);

printf("Price: %.2f\n", book1.price);

return 0;

}

**8. Write a program to read and display the details of a book using a structure.**

#include <stdio.h>

struct Book {

char title[100];

char author[100];

float price;

};

int main() {

struct Book book;

printf("Enter book details:\n");

printf("Title: ");

gets(book.title);

printf("Author: ");

gets(book.author);

printf("Price: ");

scanf("%f", &book.price);

printf("\nBook Details:\n");

printf("Title: %s\n", book.title);

printf("Author: %s\n", book.author);

printf("Price: %.2f\n", book.price);

return 0;

}

**9. Write a program to update the price of a book in a library using a structure.**

#include <stdio.h>

#include <string.h>

struct Book {

char title[100];

char author[100];

float price;

};

void updatePrice(struct Book \*book, float newPrice);

int main() {

struct Book book = {"To Kill a Mockingbird", "Harper Lee", 12.99};

printf("Original Book Details:\n");

printf("Title: %s\n", book.title);

printf("Author: %s\n", book.author);

printf("Price: %.2f\n", book.price);

updatePrice(&book, 15.99);

printf("\nUpdated Book Details:\n");

printf("Title: %s\n", book.title);

printf("Author: %s\n", book.author);

printf("Price: %.2f\n", book.price);

return 0;

}

void updatePrice(struct Book \*book, float newPrice) {

// Update the price of the book

book->price = newPrice;

}

**10. Write a program to delete a book from a library using a structure and array manipulation techniques.**

#include <stdio.h>

#include <string.h>

struct Book {

char title[100];

char author[100];

float price;

};

void deleteBook(struct Book library[], int \*size, int indexToDelete);

int main() {

int n;

printf("Enter the number of books in the library: ");

scanf("%d", &n);

struct Book library[n];

for (int i = 0; i < n; i++) {

printf("Enter details for book %d:\n", i + 1);

printf("Title: ");

getchar(); // Consume the newline character left in the buffer

fgets(library[i].title, sizeof(library[i].title), stdin);

library[i].title[strcspn(library[i].title, "\n")] = '\0'; // Remove the newline character

printf("Author: ");

fgets(library[i].author, sizeof(library[i].author), stdin);

library[i].author[strcspn(library[i].author, "\n")] = '\0'; // Remove the newline character

printf("Price: ");

scanf("%f", &library[i].price);

}

int indexToDelete;

printf("\nEnter the index of the book to delete: ");

scanf("%d", &indexToDelete);

if (indexToDelete >= 0 && indexToDelete < n) {

deleteBook(library, &n, indexToDelete);

printf("\nLibrary after deleting the book:\n");

for (int i = 0; i < n; i++) {

printf("Title: %s, Author: %s, Price: %.2f\n", library[i].title, library[i].author, library[i].price);

}

} else {

printf("Invalid index for deletion.\n");

}

return 0;

}

void deleteBook(struct Book library[], int \*size, int indexToDelete) {

// Shift elements to fill the gap

for (int i = indexToDelete; i < \*size - 1; i++) {

library[i] = library[i + 1];

}

(\*size)--;

}

***Unions :***

**1. Write a program to define and initialize a union for a variable with attributes such as integer,float, and character.**

#include <stdio.h>

union MyUnion {

int intValue;

float floatValue;

char charValue;

};

int main() {

union MyUnion myUnion = {10}; // Initializing with integer value

printf("Union Values:\n");

printf("Integer Value: %d\n", myUnion.intValue);

printf("Float Value: %.2f\n", myUnion.floatValue); // Accessing the same memory as integer

printf("Character Value: %c\n", myUnion.charValue); // Accessing the same memory as integer

return 0;

}

**2. Write a program to read and display the values of a union variable with multiple data types.**

#include <stdio.h>

union MyUnion {

int intValue;

float floatValue;

char charValue;

};

int main() {

union MyUnion myUnion;

printf("Enter an integer value: ");

scanf("%d", &myUnion.intValue);

printf("\nUnion Values:\n");

printf("Integer Value: %d\n", myUnion.intValue);

printf("Float Value: %.2f\n", myUnion.floatValue); // Accessing the same memory as integer

printf("Character Value: %c\n", myUnion.charValue); // Accessing the same memory as integer

return 0;

}

**3. Write a program to define a union for a currency with attributes such as dollars, euros, and yen.**

#include <stdio.h>

union Currency {

float dollars;

float euros;

float yen;

};

int main() {

union Currency money = {.dollars = 100.50}; // Initializing with dollars

printf("Currency Values:\n");

printf("Dollars: %.2f\n", money.dollars);

printf("Euros: %.2f\n", money.euros); // Accessing the same memory as dollars

printf("Yen: %.2f\n", money.yen); // Accessing the same memory as dollars

return 0;

}

**4. Write a program to read and display the values of a union variable with different currencies.**

#include <stdio.h>

union Currency {

float dollars;

float euros;

float yen;

};

int main() {

union Currency money;

printf("Enter amount in dollars: ");

scanf("%f", &money.dollars);

printf("\nCurrency Values:\n");

printf("Dollars: %.2f\n", money.dollars);

printf("Euros: %.2f\n", money.euros); // Accessing the same memory as dollars

printf("Yen: %.2f\n", money.yen); // Accessing the same memory as dollars

return 0;

}

**5. Write a program to perform arithmetic operations on a union variable with multiple data type.**

#include <stdio.h>

union ArithmeticUnion {

int intValue;

float floatValue;

};

int main() {

union ArithmeticUnion num1, num2, result;

// Reading integer values

printf("Enter an integer value: ");

scanf("%d", &num1.intValue);

printf("Enter another integer value: ");

scanf("%d", &num2.intValue);

// Performing addition

result.intValue = num1.intValue + num2.intValue;

// Displaying the result

printf("\nAddition Result (as Integer): %d\n", result.intValue);

// Reading float values

printf("Enter a float value: ");

scanf("%f", &num1.floatValue);

printf("Enter another float value: ");

scanf("%f", &num2.floatValue);

// Performing addition

result.floatValue = num1.floatValue + num2.floatValue;

// Displaying the result

printf("Addition Result (as Float): %.2f\n", result.floatValue);

return 0;

}

***Pointers:♣***

1. **Write a program that uses pointers to swap two integer variables.**

#include <stdio.h>

void swap(int \*a, int \*b);

int main() {

int num1 = 5, num2 = 10;

printf("Before swapping: num1 = %d, num2 = %d\n", num1, num2);

swap(&num1, &num2);

printf("After swapping: num1 = %d, num2 = %d\n", num1, num2);

return 0;

}

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

**2. Write a program that uses pointers to find the largest and smallest numbers in an array of integers.**

#include <stdio.h>

void findMinMax(int arr[], int size, int \*min, int \*max);

int main() {

int arr[] = {12, 5, 7, 20, 8, 15};

int min, max;

findMinMax(arr, sizeof(arr) / sizeof(arr[0]), &min, &max);

printf("Minimum: %d\n", min);

printf("Maximum: %d\n", max);

return 0;

}

void findMinMax(int arr[], int size, int \*min, int \*max) {

\*min = \*max = arr[0];

for (int i = 1; i < size; i++) {

if (arr[i] < \*min) {

\*min = arr[i];

}

if (arr[i] > \*max) {

\*max = arr[i];

}

}

}

**3. Write a program that uses pointers to find the length of a string.**

#include <stdio.h>

int findLength(const char \*str);

int main() {

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

int length = findLength(message);

printf("Length of the string: %d\n", length);

return 0;

}

int findLength(const char \*str) {

const char \*ptr = str;

while (\*ptr != '\0') {

ptr++;

}

return ptr - str;

}

**4. Write a program that uses pointers to concatenate two strings.**

#include <stdio.h>

void concatenateStrings(char \*dest, const char \*src);

int main() {

char str1[50] = "Hello, ";

const char \*str2 = "World!";

concatenateStrings(str1, str2);

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

return 0;

}

void concatenateStrings(char \*dest, const char \*src) {

while (\*dest != '\0') {

dest++;

}

while (\*src != '\0') {

\*dest = \*src;

dest++;

src++;

}

\*dest = '\0'; // Null-terminate the concatenated string

}

**5. Write a program that uses pointers to reverse a string.**

#include <stdio.h>

void reverseString(char \*str);

int main() {

char message[] = "Hello, World!";

reverseString(message);

printf("Reversed String: %s\n", message);

return 0;

}

void reverseString(char \*str) {

char \*start = str;

char \*end = str;

while (\*end != '\0') {

end++;

}

end--; // Move back to the last character (excluding null terminator)

while (start < end) {

// Swap characters

char temp = \*start;

\*start = \*end;

\*end = temp;

// Move pointers

start++;

end--;

}

}

**6. Write a program that uses pointers to find the sum and average of an array of floating-point numbers.**

#include <stdio.h>

void calculateSumAndAverage(float \*arr, int size, float \*sum, float \*average);

int main() {

float numbers[] = {2.5, 4.5, 6.0, 8.0, 10.5};

int size = sizeof(numbers) / sizeof(numbers[0]);

float sum, average;

calculateSumAndAverage(numbers, size, &sum, &average);

printf("Sum: %.2f\n", sum);

printf("Average: %.2f\n", average);

return 0;

}

void calculateSumAndAverage(float \*arr, int size, float \*sum, float \*average) {

\*sum = 0.0;

for (int i = 0; i < size; i++) {

\*sum += arr[i];

}

\*average = \*sum / size;

}

**7. Write a program that uses pointers to count the number of vowels in a string.**

#include <stdio.h>

int countVowels(const char \*str);

int main() {

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

int vowelCount = countVowels(message);

printf("Number of vowels: %d\n", vowelCount);

return 0;

}

int countVowels(const char \*str) {

int count = 0;

while (\*str != '\0') {

char c = \*str;

if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' ||

c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U') {

count++;

}

str++;

}

return count;

}

**8. Write a program that uses pointers to sort an array of integers in ascending order.**

#include <stdio.h>

void sortArray(int \*arr, int size);

int main() {

int numbers[] = {5, 2, 8, 1, 3};

int size = sizeof(numbers) / sizeof(numbers[0]);

sortArray(numbers, size);

printf("Sorted Array in Ascending Order:\n");

for (int i = 0; i < size; i++) {

printf("%d ", numbers[i]);

}

return 0;

}

void sortArray(int \*arr, int size) {

for (int i = 0; i < size - 1; i++) {

for (int j = 0; j < size - i - 1; j++) {

if (\*(arr + j) > \*(arr + j + 1)) {

// Swap elements

int temp = \*(arr + j);

\*(arr + j) = \*(arr + j + 1);

\*(arr + j + 1) = temp;

}

}

}

}

**9. Write a program that uses pointers to implement a stack data structure.**

#include <stdio.h>

#define MAX\_SIZE 5

struct Stack {

int items[MAX\_SIZE];

int top;

};

void initializeStack(struct Stack \*stack);

void push(struct Stack \*stack, int value);

int pop(struct Stack \*stack);

int peek(const struct Stack \*stack);

int isEmpty(const struct Stack \*stack);

int isFull(const struct Stack \*stack);

int main() {

struct Stack myStack;

initializeStack(&myStack);

push(&myStack, 1);

push(&myStack, 2);

push(&myStack, 3);

printf("Top of the stack: %d\n", peek(&myStack));

printf("Popped element: %d\n", pop(&myStack));

printf("Is the stack empty? %s\n", isEmpty(&myStack) ? "Yes" : "No");

printf("Is the stack full? %s\n", isFull(&myStack) ? "Yes" : "No");

return 0;

}

void initializeStack(struct Stack \*stack) {

stack->top = -1;

}

void push(struct Stack \*stack, int value) {

if (!isFull(stack)) {

stack->items[++stack->top] = value;

printf("Pushed %d to the stack\n", value);

} else {

printf("Stack overflow! Cannot push %d\n", value);

}

}

int pop(struct Stack \*stack) {

if (!isEmpty(stack)) {

printf("Popping %d from the stack\n", stack->items[stack->top]);

return stack->items[stack->top--];

} else {

printf("Stack underflow! Cannot pop from an empty stack\n");

return -1; // Return a sentinel value for an empty stack

}

}

int peek(const struct Stack \*stack) {

if (!isEmpty(stack)) {

return stack->items[stack->top];

} else {

return -1; // Return a sentinel value for an empty stack

}

}

int isEmpty(const struct Stack \*stack) {

return stack->top == -1;

}

int isFull(const struct Stack \*stack) {

return stack->top == MAX\_SIZE - 1;

}

**10. Write a program that uses pointers to implement a queue data structure.**

#include <stdio.h>

#define MAX\_SIZE 5

struct Queue {

int items[MAX\_SIZE];

int front, rear;

};

void initializeQueue(struct Queue \*queue);

void enqueue(struct Queue \*queue, int value);

int dequeue(struct Queue \*queue);

int peek(const struct Queue \*queue);

int isEmpty(const struct Queue \*queue);

int isFull(const struct Queue \*queue);

int main() {

struct Queue myQueue;

initializeQueue(&myQueue);

enqueue(&myQueue, 1);

enqueue(&myQueue, 2);

enqueue(&myQueue, 3);

printf("Front of the queue: %d\n", peek(&myQueue));

printf("Dequeued element: %d\n", dequeue(&myQueue));

printf("Is the queue empty? %s\n", isEmpty(&myQueue) ? "Yes" : "No");

printf("Is the queue full? %s\n", isFull(&myQueue) ? "Yes" : "No");

return 0;

}

void initializeQueue(struct Queue \*queue) {

queue->front = -1;

queue->rear = -1;

}

void enqueue(struct Queue \*queue, int value) {

if (!isFull(queue)) {

if (isEmpty(queue)) {

queue->front = 0; // Set front to 0 for the first element

}

queue->items[++queue->rear] = value;

printf("Enqueued %d to the queue\n", value);

} else {

printf("Queue overflow! Cannot enqueue %d\n", value);

}

}

int dequeue(struct Queue \*queue) {

if (!isEmpty(queue)) {

int dequeuedValue = queue->items[queue->front];

printf("Dequeued %d from the queue\n", dequeuedValue);

if (queue->front == queue->rear) {

// Reset front and rear when the last element is dequeued

initializeQueue(queue);

} else {

queue->front++;

}

return dequeuedValue;

} else {

printf("Queue underflow! Cannot dequeue from an empty queue\n");

return -1; // Return a sentinel value for an empty queue

}

}

int peek(const struct Queue \*queue) {

if (!isEmpty(queue)) {

return queue->items[queue->front];

} else {

return -1; // Return a sentinel value for an empty queue

}

}

int isEmpty(const struct Queue \*queue) {

return queue->front == -1;

}

int isFull(const struct Queue \*queue) {

return queue->rear == MAX\_SIZE - 1;

}

**11. Write a program that uses pointers to implement a linked list data structure.**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node \*next;

};

struct LinkedList {

struct Node \*head;

};

void initializeList(struct LinkedList \*list);

void insertNode(struct LinkedList \*list, int value);

void displayList(const struct LinkedList \*list);

int main() {

struct LinkedList myList;

initializeList(&myList);

insertNode(&myList, 1);

insertNode(&myList, 2);

insertNode(&myList, 3);

printf("Linked List: ");

displayList(&myList);

return 0;

}

void initializeList(struct LinkedList \*list) {

list->head = NULL;

}

void insertNode(struct LinkedList \*list, int value) {

struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));

newNode->data = value;

newNode->next = list->head;

list->head = newNode;

}

void displayList(const struct LinkedList \*list) {

struct Node \*current = list->head;

while (current != NULL) {

printf("%d ", current->data);

current = current->next;

}

printf("\n");

}

**12. Write a program that uses pointers to create a dynamic array.**

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*dynamicArray;

int size;

printf("Enter the size of the dynamic array: ");

scanf("%d", &size);

// Allocate memory for the dynamic array

dynamicArray = (int \*)malloc(size \* sizeof(int));

if (dynamicArray == NULL) {

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

return 1; // Exit with an error code

}

// Input values into the dynamic array

printf("Enter %d integers:\n", size);

for (int i = 0; i < size; i++) {

scanf("%d", &dynamicArray[i]);

}

// Display the values in the dynamic array

printf("Dynamic Array: ");

for (int i = 0; i < size; i++) {

printf("%d ", dynamicArray[i]);

}

// Free the allocated memory

free(dynamicArray);

return 0;

}