Q1

1. While and Do while loop both have conditions, but if the condition is false on first execution while loop wont execute code in it, but do while loop will execute. Do while loop always executes minimum of 1 time.

* While loop checks condition before the execution of the loops body.
* Do while loop checks condition after the execution of a loop body.

1. Create an array of size 10. Using a for loop, ask the user to enter 10 integers. Store these integers in the array.

1. #include <stdio.h>

2.

3. int main()

4. {

5. int arrA[10];

6. for (int i = 0; i < 10; i++)

7. {

8. printf("Enter integer %d: ", i);

9. scanf("%d", &arrA[i]);

10. }

11. return 0;

12. }

1. Create a 2D array to store the populations of 3 countries over 2 years, i.e. 2 rows and 3 columns. Count how many countries had a population increase between the first and second year. Print the answer to the screen. Use the data provided below to test your programme

1. #include <stdio.h>

2. int main()

3. {

4. // Index 0 year 2020

5. // Index 1 year 2021

6. int countries[2][3] = {{423000, 88260000, 988000}, {452000, 87452000, 997000}};

7. int increase = 0;

8. for (int i = 0; i < 3; i++)

9. {

10. increase += countries[1][i] > countries[0][i];

11. }

12. printf("%d", increase);

13. return 0;

14. }

Q2

1. Write a function called ‘calcDist’ to calculate the distance travelled in a car. The following parameters must be passed to the function as floating point numbers: speed (km/hr) and travel time (hr). The function should calculate the distance travelled as: distance = speed X travel time

1. void calcDist(float speed, float time)

2. {

3.     float distance = speed \* time;

4.     printf("%f", distance);

5. }

1. Write a function called ‘purchaseItems’ to compare the prices of different goods from two different sellers, seller 1 and seller 2. The function should have the following prototype:

1. #include <stdio.h>

2. float purchaseItems(float seller1Prices[], float seller2Prices[], int numSize);

3.

4. int main()

5. {

6. float seller1Prices[] = {22.35, 58.60, 88.40, 5.99};

7. float seller2Prices[] = {20.99, 59.99, 85.00, 9.99};

8. int size = 4;

9. float cost = purchaseItems(seller1Prices, seller2Prices, size);

10. printf("%f", cost);

11. return 0;

12. }

13.

14. float purchaseItems(float seller1Prices[], float seller2Prices[], int numSize)

15. {

16. float cost = 0.0;

17. for (int i = 0; i < numSize; i++)

18. {

19.

20. // Assuming items can not cost the same. If they did if statement must be modified.

21. if (seller1Prices[i] < seller2Prices[i])

22. {

23. cost += seller1Prices[i];

24. }

25. else

26. {

27. cost += seller2Prices[i];

28. }

29. }

30. return cost;

31. }

Q3.

1. Write a C programme to create a float and set its value = 2.5. Create a pointer to point to the float. Print the value of the float to the screen by dereferencing the float pointer

1. #include <stdio.h>

2. int main()

3. {

4.     float a = 2.5;

5.     float \*ptr = &a;

6.     printf("%f", \*ptr);

7.     return 0;

8. }

1. Write a C programme that creates the following array of integers: [22, 23, 24, 3, 5]. Create an int pointer. Use a while loop to loop through the array and check if each int in the array is odd or even. If the array element is odd, update the value of the pointer to point to the address of the current odd int in the array. Print out the value of the pointer for all odd integers in the array.

1. #include <stdio.h>

2.

3. int main()

4. {

5.     int array[] = {22, 23, 24, 3, 5};

6.     int \*ptr = NULL; // Initialize pointer to NULL

7.

8.     int i = 0;

9.     while (i < sizeof(array) / sizeof(int))

10.     {

11.         if (array[i] % 2 != 0)

12.         {                    // Check if the number is odd

13.             ptr = &array[i]; // Update pointer to point to the current odd integer

14.             printf("Pointer value for odd integer %d: %p\n", \*ptr, ptr);

15.         }

16.         i++;

17.     }

18.

19.     return 0;

20. }

1. What does it mean to increment a pointer?

Incrementing a pointer in C means changing the memory address it points to so that it points to the next memory location of its type. The amount by which the pointer is incremented depends on the size of the data type it points to.

Q4

1. #include <stdio.h>

2. #include <string.h>

3.

4. typedef struct

5. {

6.     int numOfBedrooms;

7.     float price;

8.     char address[100];

9. } apartment;

10. int numOfPreperties = 0;

11. apartment properties[10];

12.

13. void newProperty();

14. void displayProperties();

15.

16. int main()

17. {

18.     newProperty();

19.     newProperty();

20.     newProperty();

21.     displayProperties();

22.

23.     return 0;

24. }

25. void newProperty(){

26. if (numOfPreperties < 10) { // Check if there's space in the array

27.     printf("Enter the number of bedrooms for the new apartment: ");

28.     scanf("%d", &properties[numOfPreperties].numOfBedrooms);

29.     printf("Enter the sale price for the new apartment: ");

30.     scanf("%f", &properties[numOfPreperties].price);

31.     printf("Enter the address for the new apartment: ");

32.     getchar(); // Clear input buffer

33.     fgets(properties[numOfPreperties].address, sizeof(properties[numOfPreperties].address), stdin);

34.     // Remove newline character from address if present

35.     properties[numOfPreperties].address[strcspn(properties[numOfPreperties].address, "\n")] = '\0';

36.

37.     numOfPreperties++; // Increment the number of properties

38.     } else {

39.     printf("Cannot add more properties. Array is full.\n");

40.     }

41. }

42.

43. void displayProperties()

44. {

45.     printf("List of Properties:\n");

46.     for (int i = 0; i < numOfPreperties; ++i)

47.     {

48.         if (properties[i].numOfBedrooms != 0)

49.         { // Check if the property exists

50.             printf("Property %d:\n", i + 1);

51.             printf("Number of Bedrooms: %d\n", properties[i].numOfBedrooms);

52.             printf("Sale Price: %.2f\n", properties[i].price);

53.             printf("Address: %s\n", properties[i].address);

54.             printf("\n");

55.         }

56.     }

57. }

58.

Q5

1. #include <stdio.h>

2. #include <stdlib.h>

3. #include <string.h>

4.

5. #define MAX\_LINE\_LENGTH 100

6.

7. int main() {

8. FILE \*file;

9. char line[MAX\_LINE\_LENGTH];

10. int totalCoffeeShops = 0;

11.

12. // Part (a): Reading and printing dataset

13. printf("Dataset from coffeeShops.txt:\n");

14. file = fopen("coffeeShops.txt", "r");

15. if (file == NULL) {

16. perror("Error opening file");

17. return 1;

18. }

19. while (fgets(line, MAX\_LINE\_LENGTH, file) != NULL) {

20. printf("%s", line);

21. totalCoffeeShops++;

22. }

23. fclose(file);

24. printf("\nTotal number of coffee shops in the city: %d\n\n", totalCoffeeShops);

25.

26. // Part (b): Adding new entry for 'Tim Hortons'

27. file = fopen("coffeeShops.txt", "a");

28. if (file == NULL) {

29. perror("Error opening file");

30. return 1;

31. }

32. fprintf(file, "Tim Hortons\t4\n");

33. fclose(file);

34.

35. printf("New entry for 'Tim Hortons' with 4 locations added successfully.\n");

36.

37. return 0;

38. }

39.

Q6

1. #include <stdio.h>

2. #include <stdlib.h>

3. #include <time.h>

4.

5. // Function to roll a pair of dice and return their sum

6. int roll\_pair() {

7. return (rand() % 6 + 1) + (rand() % 6 + 1);

8. }

9.

10. // Function to simulate rolling dice until the target sum is achieved

11. int rolls\_to\_get(int target) {

12. int rolls = 0;

13. int sum;

14. do {

15. sum = roll\_pair();

16. rolls++;

17. } while (sum != target);

18. return rolls;

19. }

20.

21. int main() {

22. srand(time(NULL)); // Seed the random number generator

23.

24. printf("Target\tAverage Rolls Needed\n");

25.

26. for (int target = 2; target <= 12; target++) {

27. int total\_rolls = 0;

28. for (int i = 0; i < 10000; i++) {

29. total\_rolls += rolls\_to\_get(target);

30. }

31. double average\_rolls = (double)total\_rolls / 10000;

32. printf("%d\t%.2f\n", target, average\_rolls);

33. }

34.

35. return 0;

36. }

37.

Q7

* Flexibility: Dynamic memory allocation allows the programmer to allocate memory during runtime.
* Memory Management: With dynamic memory allocation, memory can be allocated and deallocated as needed, preventing memory wastage and fragmentation. This helps in optimizing memory usage and avoiding memory leaks.

1. #include <stdio.h>

2. #include <stdlib.h>

3. #include <time.h>

4.

5. int main() {

6. int x;

7. printf("Enter the number of integers you want to generate: ");

8. scanf("%d", &x);

9.

10. // Allocate memory for the array

11. int \*arr = (int \*)malloc(x \* sizeof(int));

12.

13. // Check if memory allocation was successful

14. if (arr == NULL) {

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

16. return 1;

17. }

18.

19. // Seed the random number generator

20. srand(time(NULL));

21.

22. // Populate the array with random integers in the range 0 - 999

23. for (int i = 0; i < x; i++) {

24. arr[i] = rand() % 1000; // Generate random integer in the range 0 - 999

25. }

26.

27. // Display the array

28. printf("Array generated with %d random integers:\n", x);

29. for (int i = 0; i < x; i++) {

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

31. }

32. printf("\n");

33.

34. // Calculate and display the highest and lowest values in the array

35. int highest = arr[0];

36. int lowest = arr[0];

37. for (int i = 1; i < x; i++) {

38. if (arr[i] > highest) {

39. highest = arr[i];

40. }

41. if (arr[i] < lowest) {

42. lowest = arr[i];

43. }

44. }

45. printf("Highest value in the array: %d\n", highest);

46. printf("Lowest value in the array: %d\n", lowest);

47.

48. // Calculate and display the most frequently occurring value(s) in the array

49. int frequency[1000] = {0}; // Assuming the range of values is 0 - 999

50. int max\_frequency = 0;

51. printf("Most frequently occurring value(s) in the array: ");

52. for (int i = 0; i < x; i++) {

53. frequency[arr[i]]++;

54. if (frequency[arr[i]] > max\_frequency) {

55. max\_frequency = frequency[arr[i]];

56. }

57. }

58. for (int i = 0; i < 1000; i++) {

59. if (frequency[i] == max\_frequency) {

60. printf("%d ", i);

61. }

62. }

63. printf("\n");

64.

65. // Free dynamically allocated memory

66. free(arr);

67.

68. return 0;

69. }

70.