Ques 1: What is a flow chart? Define. Also describe the various symbols that are used in a flowchart, with a suitable example. Discuss the various advantages and limitations of flow charting in C.

Ans: A flow chart in C is a graphical representation of a computer program written in the C programming language. It uses symbols to represent various statements and operations in the program, and arrows to indicate the flow of the program.

Some common symbols used in a flow chart for a C program include:

- Start/End: This symbol is used to indicate the start or end of the program.

- Input/Output: These symbols represent the input or output of data to or from the program.

- Process: This symbol is used to represent an operation or action in the program.

- Decision: This symbol is used to represent a decision point in the program.

- Loop: This symbol is used to represent a loop or repetitive task in the program.

For example, a flow chart for a simple program that calculates the sum of two numbers might include a start symbol, followed by input symbols for the two numbers, a process symbol for adding the numbers together, and an output symbol for displaying the result.

Some advantages of using flow charts for C programs include:

- They can help visualize the overall structure and logic of the program.

- They can help identify potential errors or inefficiencies in the program.

- They can be useful for documenting the program or for communicating its structure to others.

However, there are also some limitations to using flow charts for C programs. For example:

- They can be time-consuming to create, especially for complex programs.

- They may not be suitable for programs with many branches or complex logic.

- They may become outdated quickly if the program is updated or revised.

Overall, flow charting can be a useful tool for visualizing and analyzing C programs, but it is important to use it in conjunction with other methods and to be aware of its limitations.

Ques 2: Describe various types of operators available in C language in detail.

Ans: In C language, operators are used to perform various operations on variables and values. There are different types of operators available in C language, and they can be classified into the following categories:

1. Arithmetic Operators: Arithmetic operators are used to perform mathematical operations on numeric values. These include addition (+), subtraction (-), multiplication (), division (/), and modulus (%).

2. Relational Operators: Relational operators are used to compare two values and return a Boolean result based on whether the comparison is true or false. The relational operators include less than (<), greater than (>), less than or equal to (<=), greater than or equal to (>=), equal to (==), and not equal to (!=).

3. Logical Operators: Logical operators are used to combine two or more relational expressions and return a Boolean result based on whether the combined expression is true or false. The logical operators include AND (&&), OR (||), and NOT (!).

4. Bitwise Operators: Bitwise operators are used to perform operations on individual bits of values. These include AND (&), OR (|), XOR (^), left shift (<<), and right shift (>>).

5. Assignment Operators: Assignment operators are used to assign a value to a variable. These include equals (=), plus-equals (+=), minus-equals (-=), multiply-equals (=), divide-equals (/=), modulus-equals (%=), AND-equals (&=), OR-equals (|=), XOR-equals (^=), left-shift-equals (<<=), and right-shift-equals (>>=).

6. Conditional Operators: Conditional operators are used to evaluate a Boolean expression and return one of two values based on whether the expression is true or false. The conditional operator is represented by a question mark (?) and a colon (:).

7. Comma Operators: Comma operators are used to separate multiple expressions in a single statement. The expressions are evaluated from left to right, and the final value is the value of the last expression.

These are the various types of operators available in C language and their functions.

Ques 3: Differentiate between syntax error and sematic error in C.

Ans: Syntax errors and semantic errors are both types of errors that can occur in C programming but are quite different from each other:

1. Syntax Errors: These errors occur due to mistakes in the syntax or grammar of the program, such as missing semicolons, misplaced brackets, or incorrect use of keywords. The compiler detects these errors while compiling the program and throws a compilation

error. Typically, these errors prevent the program from being compiled or executed, as the syntax is incorrect.

2. Semantic Errors: These errors occur due to logical errors in the program, which can cause incorrect output or behaviour. These errors are harder to detect and are typically detected during the execution of the program. An example of a semantic error is dividing a number by zero, which would cause a runtime error and can crash the program.

To differentiate, syntax errors occur due to mistakes in the syntax or grammar of the program, and semantic errors occur due to logical errors. Syntax errors can be detected by the compiler and prevent the program from being compiled, while semantic errors can cause incorrect output or behaviour during runtime.

Ques 4: What is algorithm? Write an algorithm to find out middle number after sorting in given 5 input no. in C.

Ans: An algorithm is a step-by-step procedure for solving a problem or performing a task. In programming, algorithms help in implementing solutions to problems efficiently.

Here's an algorithm in C to find the middle number after sorting the given 5 input numbers:

1. Start

2. Declare 5 integer variables (a, b, c, d, e) to store the input numbers.

3. Take input for 5 integers.

4. Use a sorting algorithm (for example, a bubble sort) to sort the input numbers in ascending or descending order.

5. Find the middle number by calculating (a+b+c+d+e)/5.

6. Print the middle number.

7. End.

Here's the code for the algorithm:

```
#include <stdio.h>

int main()

{

   int a, b, c, d, e;
```

```
printf("Enter five integers:");
scanf("%d %d %d %d %d", &a, &b, &c, &d, &e);
int arr[5] = {a, b, c, d, e};
int temp;
for (int i = 0; i < 5; i++)
{
    for (int j = i+1; j < 5; j++)
    {
        if (arr[i] > arr[j])
        {
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
}
int middle = arr[2];
printf("The middle number is: %d", middle);
return 0;
}
```

This algorithm first takes input of the five integers, sorts them using bubble sort, and then finds the middle number by accessing the third index (2nd index for array starting from 0) of the sorted array. Finally, the middle number is printed to the console.

Ques 5: Differentiate between switch case and nested if else structure in C.

Ans: Switch case and nested if else structures are both conditional constructs that are used in C programming to execute different statements based on different conditions. However, they differ from each other in terms of syntax and functionality:

1. Switch case:

- Syntax:

```
switch (expression) {
   case value1:
      statement1;
      break;
   case value2:
      statement2;
      break;
   default:
      statementN;
}
```

- Functionality:

The switch case structure is used to execute different statements based on the value of the expression evaluated to different constant values. The switch statement evaluates the expression once and then tests the value of the expression against each case label. If there is a match, the code block associated with that case is executed. If there is no match, the code block associated with the default label is executed. We can also include a break statement to interrupt the switch statement execution and jump to the next statement outside the switch statement.

2. Nested if else:

- Syntax:

```
if (expression1)
{
   // statement 1
```

```
    if (expression2)
    {
        // statement 2
    }
}
else if (expression3)
{
    // statement 3
    if (expression4)
    {
        // statement 4
    }
}
else {
    // statement 5
}
```

- Functionality:

A nested if else structure is used to execute different statements based on multiple conditions. These conditions can be evaluated using multiple if else statements that are nested inside each other. In the case of nested if else structures, each condition is evaluated sequentially, and the first matching condition will execute the associated block of code.

In summary, switch case and nested if else structures are conditional constructs that help execute different statements based on different conditions. However, switch case structures are used to test an expression against different values, while nested if else structures are used to test multiple conditions against each other.

Ques 6: Differentiate between Do While and while control structure in C.

Ans: Do-while and while control structures are both loop structures used in C programming to execute a block of code repeatedly until a certain condition is met. However, they differ from each other in terms of when the initial test condition is checked:

1. While:

- Syntax:

```
while (test_condition) {

    // code to be executed

}
```

- Functionality:

The while structure executes the code inside the loop as long as the test_condition is true. The test_condition is evaluated before the code inside the loop is executed. If the test_condition is false, the loop is skipped entirely and the program moves on to the next statement.

2. Do-While:

- Syntax:

```
do {
    // code to be executed
} while (test_condition);
```

- Functionality:

The do-while structure executes the code inside the loop first and then checks the test_condition. Even if the test_condition is false, the code inside the loop is executed at least once before the program moves on to the next statement.

In summary, the key difference between the while and do-while structures is that the while structure checks the test_condition before the code inside the loop is executed, while the do-while structure executes the code inside the loop first and then checks the test_condition. This means that the do-while loop will always execute at least once, while the while loop may not execute at all if the test_condition is false from the outset.

Ques 7: What is statement in C? Explain various looping statement with suitable example.

Ans: In C programming language, a statement is an instruction that performs a specific action. C programs are made up of one or more statements, and each statement ends with a semicolon (;).

There are three types of looping statements in C: for, while, and do-while loops.

1. For Loop:
The for loop is used to execute a block of code repeatedly for a specified number of times. It has the following syntax:

```
for (initialization; test_condition; update) {
    // code to be executed
```

```
}
```

where,

initialization: This statement initializes the loop counter before the loop starts.

test_condition: This statement is evaluated before each iteration. If the condition is true, the loop will continue to execute. If it is false, the loop will terminate.

update: This statement is executed after each iteration. It is used to update the loop counter.

For example, the following loop prints the numbers from 1 to 10:

```
for (int i = 1; i <= 10; i++) {
    printf("%d ", i);
}
```

2. While Loop:
The while loop is used to execute a block of code repeatedly until a test condition becomes false. It has the following syntax:

```
while (test_condition) {
    // code to be executed
}
```

where,

test_condition: This statement is evaluated before each iteration. If the condition is true, the loop will continue to execute. If it is false, the loop will terminate.

For example, the following loop prints the numbers from 1 to 10:

```
int i = 1;
while (i <= 10) {
    printf("%d ", i);
    i++;
}
```

3. Do-While Loop:
The do-while loop is used to execute a block of code repeatedly until a test condition becomes false. It is similar to the while loop, but the test condition is evaluated after the first iteration. It has the following syntax:

```
do {

    // code to be executed

} while (test_condition);
```

where,

test_condition: This statement is evaluated after each iteration. If the condition is true, the loop will continue to execute. If it is false, the loop will terminate.

For example, the following loop prints the numbers from 1 to 10:

```
int i = 1;

do {

    printf("%d ", i);

    i++;

} while (i <= 10);
```

Overall, looping statements are an important part of C programming and help to execute a block of code repeatedly.

Ques 8: Explain Break, Continue, switch in C.

Ans: 1. Break Statement: The break statement is a control statement in C used to terminate the execution of a loop, switch, or entire program based on some condition. When the break statement is encountered, the program execution jumps outside the current loop or switch case. The break statement is commonly used to exit a loop when a certain condition is met. Here is an example:

```
for (int i = 0; i < 10; i++) {

    if (i == 5) {
        // Exit the loop when i is 5
        break;
    }
```

```
    printf("%d ", i);
}
```

2. Continue Statement: The continue statement is used in C to skip the current iteration of a loop and move on to the next iteration. When the continue statement is encountered, the program skips the remaining statements in the current iteration of the loop and moves to the next iteration. The continue statement is commonly used to skip certain iterations of a loop based on some condition. Here is an example:

```
for (int i = 0; i < 10; i++) {

    if (i == 5) {
        // Skip the iteration when i is 5
        continue;
    }

    printf("%d ", i);
}
```

3. Switch Statement: The switch statement is a control statement in C used to perform different actions based on different conditions. The switch statement evaluates a variable or expression and matches its value against various cases. When a match is found, the corresponding case statement is executed. If no match is found, the default case is executed. Here is an example:

```
int day = 2;
switch (day) {
    case 1:
        printf("Monday");
        break;
    case 2:
        printf("Tuesday");
        break;
    case 3:
        printf("Wednesday");
        break;
    default:
        printf("Invalid Day");
}
```

In this example, the variable day is evaluated using the switch statement. Since day is equal to 2, the statement printf("Tuesday"); is executed. If day were equal to any other value, the corresponding case statement would be executed, or the default case would be executed if no match is found.

Ques 9: Explain an array in C. Discuss one and two dimensional array with an example. Write a C program for add and multiplication of two 3*3 array.

Ans: In C, an array is a collection of elements of the same data type stored in contiguous memory locations. An array is represented by a name and an index. Each element in an array can be accessed through its index.

1. One-dimensional array:
A one-dimensional array is an array that has a single row of elements. It can be declared using the following syntax:

```
datatype arrayName[arraySize];
```

For example, to declare an array of 5 integers, we can use the following code:

```
int numbers[5];
```

Each element in the array can be accessed using its index, starting from 0. The following code initializes the first element of the array to 10 and prints its value:

```
numbers[0] = 10;
printf("The first element of the array is %d", numbers[0]);
```

2. Two-dimensional array:
A two-dimensional array is an array that has rows and columns of elements. It can be declared using the following syntax:

```
datatype arrayName[rowSize][columnSize];
```

For example, to declare a 2D array of 3 rows and 3 columns, we can use the following code:

```
int matrix[3][3];
```

Each element in the array can be accessed using its row index and column index. The first index corresponds to the row, and the second index corresponds to the column. The following code initializes the element in the second row and third column to 20 and prints its value:

```
matrix[1][2] = 20;
```

```
printf("The element at row 2 and column 3 is %d", matrix[1][2]);
```

Here is a C program demonstrating the addition and multiplication of two 3x3 arrays:

```
#include <stdio.h>

int main() {

  int A[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
  int B[3][3] = {{9, 8, 7}, {6, 5, 4}, {3, 2, 1}};
  int sum[3][3], product[3][3];
  int i, j, k;

  // Add the two arrays
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++) {
      sum[i][j] = A[i][j] + B[i][j];
    }
  }

  // Multiply the two arrays
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++) {
      product[i][j] = 0;
      for (k = 0; k < 3; k++) {
        product[i][j] += A[i][k] * B[k][j];
      }
    }
  }

  // Print the sum array
  printf("The sum of the two arrays is:\n");
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++) {
      printf("%d ", sum[i][j]);
    }
    printf("\n");
  }

  // Print the product array
  printf("The product of the two arrays is:\n");
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++) {
      printf("%d ", product[i][j]);
    }
    printf("\n");
```

```
  }

  return 0;
}
```

This program initializes two 3x3 arrays, A and B, and then calculates the sum and product of the two arrays. The sum and product arrays are then printed to the console.

Ques 10: Define Static Variable, Register Variable, External Variable, Local Variable in C with suitable Program.

Ans: In C, there are several types of variables that are defined using different storage classes. The four main storage classes of variables are:

1. Static Variables:
Static variables are variables whose values persist throughout the life of a program, meaning they are visible and accessible throughout the program. They are initiated only once when the program is started and retain their value even if they go out of scope. By default, static variables are initialized to zero. Here is an example:

```
#include <stdio.h>

void func() {
  static int count = 0;
  count++;
  printf("The value of count is %d\n", count);
}

int main() {
  func();
  func();
  func();
  return 0;
}
```

In this program, count is a static variable declared inside the func() function. Each time func() is called, the value of count is incremented, and the new value is printed to the console. Since count is static, it retains its value between function calls.

2. Register Variables:
Register variables are variables that are stored in the CPU's register instead of RAM. The register is the fastest storage location in the CPU, which makes register variables faster to access than other variables. Register variables are declared using the register

keyword. However, the use of register variables is limited, and it depends on the CPU architecture and the number of available registers. Here is an example:

```
#include <stdio.h>

int main() {
  register int i;
  int sum = 0;
  for (i = 1; i <= 100; i++) {
    sum += i;
  }
  printf("The sum of numbers from 1 to 100 is %d\n", sum);
  return 0;
}
```

In this program, i is a register variable that is used to store the loop counter in a loop that calculates the sum of numbers from 1 to 100.

3. External Variables:
External variables are variables that are defined outside of any function and can be accessed by any function in the program. External variables are declared using the extern keyword. They are useful when multiple files need to access the same data. Here is an example:

In file1.c:

```
#include <stdio.h>

int count = 0;

void func() {
  count++;
}

int main() {
  func();
  func();
  func();
  printf("The value of count is %d\n", count);
  return 0;
}
```

In file2.c:

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

```
extern int count;

void func() {
  count++;
}

int main() {
  func();
  printf("The value of count is %d\n", count);
  return 0;
}
```

In this example, count is declared as an external variable in file2.c using the extern keyword. It is defined and initialized in file1.c. Both files can access count, and any changes made to the variable in one file will be reflected in the other file.

4. Local Variables:
Local variables are variables that are declared inside a function and are accessible only within that function. Local variables are allocated in the stack memory, and their memory is freed when the function returns. Here is an example:

```
#include <stdio.h>

void func() {
  int i = 5;
  printf("The value of i is %d\n", i);
}

int main() {
  func();
  return 0;
}
```

In this program, i is a local variable declared inside the func() function. It is accessible only within the function and is not accessible outside of it. Once func() returns, the memory allocated for i is freed from the stack.

Ques 11: Discuss following terms in C with suitable examples- Pointers to pointers, Pointers to Functions, Pointers to structures.

Ans: 1. Pointers to Pointers:
Pointers to pointers are also called double pointers. They are used to store the address of a pointer variable. In C, the pointer to pointer is declared with two asterisks () and is used to store the address of a pointer variable. Here's an example:

```
#include <stdio.h>

int main() {
  int a = 10;
  int *p;
  int q;

  p = &a;
  q = &p;

  printf("The value of a is %d\n", a);
  printf("The value of p is %d\n", p);
  printf("The value of q is %d\n", **q);

  return 0;
}
```

In this program, a is an integer variable, p is a pointer pointing to a, and q is a pointer pointing to p. The value of a is accessed using the variable name, p` is accessed using the pointer operator (), and q is accessed using the double pointer operator (*).

2. Pointers to Functions:

Pointers to functions are used to store the memory address of a function. In C, a function pointer is declared using the function prototype with an asterisk () before the function name. Here's an example:

```
#include <stdio.h>

void sayHello() {
  printf("Hello, World!\n");
}

int main() {
  void (*ptr)();
  ptr = &sayHello;
```

```
  (*ptr)();
  return 0;
}
```

In this program, ptr is a pointer to a function sayHello(). The memory address of this function is assigned to ptr, and the function is called using the pointer.

3. Pointers to Structures:

Pointers to structures are used to access the members of a structure through a pointer. In C, a pointer to a structure is declared using the structure name and an asterisk () before the pointer name. Here's an example:

```
#include <stdio.h>

struct Person {
  char name[50];
  int age;
};

int main() {
  struct Person p = {"John Doe", 25};
  struct Personptr = &p;
  printf("Name: %s\n", ptr->name);
  printf("Age: %d\n", ptr->age);
  return 0;
}
```

In this program, p is a structure of type Person with two members, name and age. A pointer ptr of type Person is declared and assigned the address of p. The members of p` are accessed using the pointer variable with the arrow operator (->). This allows us to access a member of the structure using a pointer to that structure.

Ques 12: Explain the different prototype of function. Also differentiate between call by value and call by reference approach with suitable examples.

Ans: In C programming, a function prototype is a declaration of the function that provides details such as the function name, return type, and parameter list to the compiler. There are different types of function prototypes in C:

1. No Argument Prototype: This type of prototype specifies that the function takes no arguments. For example:

```
int functionName(void);
```

2. Fixed Argument Prototype: This type of prototype specifies the number and type of arguments that the function takes. For example:

```
int functionName(int arg1, float arg2, char arg3);
```

3. Variable Argument Prototype: This type of prototype is used for functions that can take a variable number of arguments. For example:

```
int functionName(int count, ...);
```

Now, let's differentiate between call by value and call by reference approach:

1. Call by Value: In call by value approach, the value of the actual parameter is passed to the function, and the function works on a copy of that value. This means that any changes made in the function do not affect the actual value of the parameter. For example:

```
#include <stdio.h>

void swap(int x, int y) {
  int temp = x;
  x = y;
  y = temp;
}

int main() {
  int a = 10, b = 20;
  printf("Before Swap: a = %d, b = %d\n", a, b);
  swap(a, b);
  printf("After Swap: a = %d, b = %d\n", a, b);
  return 0;
}
```

In this example, the swap() function takes two integer parameters, x and y, and swaps their values. But since the values of a and b are passed by value to the function, any changes made to x and y do not affect a and b.

2. Call by Reference: In call by reference approach, the memory address of the actual parameter is passed to the function, and the function works on the original value of the parameter. This means that any changes made in the function affect the actual value of the parameter. For example:

```
#include <stdio.h>

void swap(int *x, int *y) {
  int temp = *x;
  *x = *y;
  *y = temp;
}

int main() {
  int a = 10, b = 20;
  printf("Before Swap: a = %d, b = %d\n", a, b);
  swap(&a, &b);
  printf("After Swap: a = %d, b = %d\n", a, b);
  return 0;
}
```

In this example, the swap() function takes two pointers to integer parameters, x and y, and swaps their values by using the pointer dereference operator (*). When the swap() function is called with the addresses of a and b, the values of a and b are swapped in memory, and these changes are reflected outside the function.

Ques 13: What is recursion in C? Explain your answer with example of printing 1 to 10.

Ans: Recursion is a process in which a function calls itself repeatedly until a base condition is met. In C programming, recursion is a powerful technique that can be used to solve complex problems that are difficult to solve using iterative methods. Recursive functions provide an elegant way of representing complex problems by breaking them into simpler sub-problems.

Here is a recursive function that prints the numbers from 1 to 10:

```
#include <stdio.h>

void printNum(int n) {
   if (n > 0) {
      printNum(n - 1);
      printf("%d ", n);
   }
}

int main() {
   int n = 10;
   printNum(n);
   return 0;
}
```

In this code, the function printNum() takes an integer parameter n. The function first checks if n is greater than 0. If it is, the function calls itself with the argument n - 1. This process continues until the base case is reached, which is when n becomes 0. At this point, no further recursive calls are made, and the function starts returning from each recursive call, printing the numbers from 1 to 10.

When the program is run, the output will be:

```
1 2 3 4 5 6 7 8 9 10
```

In summary, recursion in C allows us to solve complex problems by breaking them into simpler sub-problems, and it is achieved by a function calling itself with a smaller input.

Ques 14: What is the need of structure in C? Define the relation between function and structure.

Ans: In C programming, a structure is a user-defined data type that allows us to group together related data of different data types into a single unit. Structures provide a way to represent real-world entities in a program by grouping their characteristics together.

The need for structures in C arises when we want to group related data of different data types together. For example, if we want to represent a student in a program, we would need to store information such as the student's name, roll number, and marks in different data types. Using a structure, we can group these pieces of information into a single entity and access them as a single unit.

Here's an example of declaring a structure in C:

```
struct student {
```

```
  char name[50];
  int roll;
  float marks;
};
```

In this example, we have declared a structure student that contains three members: name, roll, and marks.

The relationship between functions and structures in C is that functions can operate on structures. This means that we can pass structures as arguments to functions, return structures from functions, and use structures inside functions.

Here's an example of defining a function that operates on a structure:

```
#include <stdio.h>

struct student {
  char name[50];
  int roll;
  float marks;
};

void display(struct student s) {
  printf("Name: %s\nRoll: %d\nMarks: %.2f\n", s.name, s.roll, s.marks);
}

int main() {
  struct student s1 = {"John Doe", 101, 89.5};
  display(s1);
  return 0;
}
```

In this code, we have defined a function display that takes a structure student as an argument and displays its contents to the console. We have also defined a structure student with three members: name, roll, and marks. Inside the main function, we have declared a variable s1 of type student and initialized its members. We then call the display function and pass the s1 variable as an argument.

When the program is run, the output will be:

```
Name: John Doe
Roll: 101
Marks: 89.50
```

In summary, the relationship between functions and structures in C allows us to operate on structures using functions and enables us to group related data of different data types together into a single entity.

Ques 15: Differentiate between Union and structure with suitable examples in C.

Ans: In C programming, structures and unions are user-defined data types that allow the grouping of related data of different data types into a single unit. However, they differ in how they store and access their members.

1. Structures:
A structure is a user-defined data type that allows the grouping of related data of different data types into a single unit. The members of a structure are stored in contiguous memory locations, and each member can be accessed independently. Here's an example of defining and using a structure in C:

```
#include <stdio.h>

struct Books {
  char  title[50];
  char  author[50];
  int   book_id;
};

int main() {
  struct Books book1;

  strcpy(book1.title, "C Programming");
  strcpy(book1.author, "Dennis Ritchie");
  book1.book_id = 1001;

  printf("Book 1 title : %s\n", book1.title);
  printf("Book 1 author : %s\n", book1.author);
  printf("Book 1 book_id : %d\n", book1.book_id);

  return 0;
}
```

In this example, we have defined a structure Books that contains three members: title, author, and book_id. We have declared a variable book1 of type Books and initialized its members using the strcpy() function and the assignment operator.

2. Unions:
A union is a user-defined data type that allows the grouping of related data of different data types into a single unit. However, unlike structures, the members of a union share the same memory location, and only one member can be accessed at a time. Here's an example of defining and using a union in C:

```
#include <stdio.h>

union Data {
  int i;
  float f;
  char str[20];
};

int main() {
  union Data data;

  data.i = 10;
  printf("data.i : %d\n", data.i);

  data.f = 3.14;
  printf("data.f : %f\n", data.f);

  strcpy(data.str, "C Programming");
  printf("data.str : %s\n", data.str);

  return 0;
}
```

In this example, we have defined a union Data that contains three members: i, f, and str. We have declared a variable data of type Data and initialized its members using the assignment operator and the strcpy() function. Since the members of a union share the same memory location, changing the value of one member will affect the values of the other members.

In summary, structures and unions are user-defined data types that allow the grouping of related data of different data types into a single unit. The main difference between the two is in how they store and access their members. Structures store their members in contiguous memory locations that can be accessed independently, while unions store their members in the same memory location and only one member can be accessed at a time.

Ques 16: Define the following in relation to C, typedef, self-referential structures.

Ans: In C programming, typedef and self-referential structures are used to create new data types.

1. Typedef:
Typedef is a C keyword that can be used to assign new and more meaningful names to existing data types. A typedef declaration creates a new name for an existing type.

```
typedef int mynum;
```

In this example, we have created a new name mynum for the int data type. We can use the mynum keyword to declare variables of type int.

2. Self-Referential Structures:
A self-referential structure is a structure that contains a pointer to another structure of the same type. Self-referential structures are useful when we need to create data structures that have a recursive nature.

Here is an example of a self-referential structure used to create a linked list:

```
struct node {
   int data;
   struct node *next;
};
```

In this example, the node structure contains two members: data and next. The data member stores the data for the node, and the next member is a pointer to another node structure.

By using a self-referential structure, we can create a linked list in which each node contains a pointer to the next node in the list.

In summary, typedef and self-referential structures are powerful features of C that allow us to create new data types and structures with recursive properties. Typedef allows us to assign new and more meaningful names to existing data types, while self-referential structures allow us to create data structures with a recursive nature, such as linked lists.

Ques 17: Differentiate between an array and a structure in C. Also discuss memory management in a 3d array.

Ans: In C programming, arrays and structures are both user-defined data types.

1. Array:
An array is a collection of elements of the same data type that are stored in contiguous memory locations. The elements of an array can be accessed using their index. Arrays can be of one, two, or three dimensions.

Here is an example of an array:

```
#include <stdio.h>

int main() {
  int arr[5] = {1, 2, 3, 4, 5};
  printf("Third element of the array is %d\n", arr[2]);
  return 0;
}
```

In this example, we have declared an array arr of 5 integers and initialized its contents. We have then accessed the third element of the array using its index.

2. Structure:
A structure is a user-defined data type that allows us to group related data of different data types into a single unit. The members of a structure can be accessed using the dot operator.

Here is an example of a structure:

```
#include <stdio.h>

struct Person {
  char name[50];
  int age;
};

int main() {
  struct Person p1 = {"John Doe", 25};
  printf("Name: %s\n", p1.name);
  printf("Age: %d\n", p1.age);
  return 0;
}
```

In this example, we have declared a structure Person with two members: name and age. We have then created a variable p1 of type Person and initialized its members. We have then accessed the members of the p1 variable using the dot operator.

3. Memory Management in a 3D Array:
A 3D array is an array that has three dimensions. The elements of a 3D array can be accessed using their indices. In memory, a 3D array is stored as a contiguous block of memory.

Here is an example of declaring and accessing a 3D array:

```
#include <stdio.h>

int main() {
  int arr[3][3][3] = {
     {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}},
     {{10, 11, 12}, {13, 14, 15}, {16, 17, 18}},
     {{19, 20, 21}, {22, 23, 24}, {25, 26, 27}}
  };
  printf("The element at arr[1][2][1] is %d\n", arr[1][2][1]);
  return 0;
}
```

In this example, we have declared a 3D array arr of size 3x3x3 and initialized its contents. We have then accessed the element at indices 1, 2, and 1 using their respective indices.

In memory, the 3D array arr would be stored as a contiguous block of memory with a total of 27 elements. The first element would be stored at the beginning of the memory block, and each subsequent element would be stored after the previous one in memory.

In summary, arrays and structures are user-defined data types in C. Arrays are collections of elements of the same data type that are stored in contiguous memory locations, while structures are collections of related data of different data types that are accessed using the dot operator. A 3D array is a multi-dimensional array that is stored in memory as a contiguous block of memory.

Ques 18: What are command line arguments in C language.

Ans: In C programming, command line arguments are parameters passed to a program when it is invoked from the command line. Command line arguments are useful for passing information to a program that cannot be entered through normal input channels.

In C, the command line arguments are passed to the main() function as two arguments: argc and argv.

- argc is an integer that represents the number of command line arguments passed to the program.
- argv is an array of strings that contains the command line arguments passed to the program.

Here's an example of a program that uses command line arguments:

```
#include <stdio.h>

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

  printf("The program name is %s\n", argv[0]);

  for(i = 1; i < argc; i++) {
    printf("Argument %d: %s\n", i, argv[i]);
  }

  return 0;
}
```

In this program, the main() function takes two arguments, argc and argv. The program name is stored in the first element of the argv array (argv[0]), and the rest of the command line arguments are stored in the subsequent elements of the array.

The for loop in the program iterates through each command line argument and prints its position (i) and its value (argv[i]).

For example, if we run the program with the command ./program argument1 argument2 argument3, the output would be:

```
The program name is ./program
Argument 1: argument1
Argument 2: argument2
Argument 3: argument3
```

In summary, in C programming, command line arguments are parameters passed to a program when it is invoked from the command line. Command line arguments are passed to the main() function as two arguments: argc and argv. argc represents the number of command line arguments passed to the program, and argv is an array of strings that contains the command line arguments.

Ques 19: What do you understand by the formatted file input and output?

Ans: Formatted input/output is a method of reading and writing data to a file in a specific format. It is useful when we want to read or write data that is not in the simple text format but has a specific structure. Formatted input/output is done using the functions fscanf(), fprintf(), fread(), and fwrite().

Formatted output to a file is done using the fprintf() function. The syntax of the fprintf() function is:

```
int fprintf(FILE *stream, const char *format, ...)
```

Here, stream is the pointer to the file where the output will be written, and format is the string that specifies the format of the output. The ... operator is used to pass the data that needs to be printed in the specified format.

Here's an example of how to use the fprintf() function:

```
#include <stdio.h>

int main() {
  FILE *fp;
  char name[50] = "John Doe";
  int age = 25;

  fp = fopen("person.txt", "w");
  fprintf(fp, "Name: %s, Age: %d", name, age);
  fclose(fp);

  return 0;
}
```

In this example, we have opened a file named person.txt in write mode using the fopen() function. We have then used the fprintf() function to write formatted output to the file. We have printed the name and age of a person in the format Name: John Doe, Age: 25.

Formatted input from a file is done using the fscanf() function. The syntax of the fscanf() function is:

```
int fscanf(FILE *stream, const char *format, ...)
```

Here, stream is the pointer to the file from which the input will be read, and format is the string that specifies the format of the input. The ... operator is used to pass the variables that need to be assigned the input.

Here's an example of how to use the fscanf() function:

```
#include <stdio.h>

int main() {
  FILE *fp;
  char name[50];
  int age;

  fp = fopen("person.txt", "r");
  fscanf(fp, "Name: %s, Age: %d", name, &age);
  fclose(fp);

  printf("Name: %s\nAge: %d\n", name, age);

  return 0;
}
```

In this example, we have opened a file named person.txt in read mode using the fopen() function. We have then used the fscanf() function to read formatted input from the file. We have assigned the name and age of a person to the variables name and age, respectively. We have then printed the values of these variables to the console using the printf() function.

In summary, formatted input/output is a method of reading and writing data to a file in a specific format. Formatted output is done using the fprintf() function, while formatted input is done using the fscanf() function.

Ques 20: What is file handling & file in C? How to create a file in C? Explain the different file operations in C program.

Ans: File handling is the mechanism of reading and writing data to and from a file in a computer. File handling in C programming language provides a way to store data permanently on a hard disk.

A file is a collection of related information that is stored on a secondary memory device, such as a hard disk. In C programming language, a file is represented by a pointer of type FILE.

To create a file in C, the fopen() function is used. The fopen() function takes two arguments: the file name and the mode. The mode specifies the purpose of opening a file, whether we want to read, write, or append data to the file.

Here's an example of how to create a file in C:

```
#include <stdio.h>

int main() {
  FILE *fp;
  fp = fopen("file.txt", "w");
  fprintf(fp, "This is a new file\n");
  fclose(fp);
  return 0;
}
```

In this example, we have used the fopen() function to create a file named file.txt in write mode. We have then used the fprintf() function to write a string to the file and then closed the file using the fclose() function.

Different file operations include reading and writing data to a file. In C programming language, the following file operation modes are available:

1. "r": opens a file for reading. If the file is not present, it returns an error.

2. "w": opens a file for writing. If the file does not exist, it creates a new file.

3. "a": opens a file for appending. If the file does not exist, it creates a new file.

4. "r+": opens a file for both reading and writing.

5. "w+": opens a file for both reading and writing. If the file does not exist, it creates a new file.

6. "a+": opens a file for both reading and appending. If the file does not exist, it creates a new file.

Here's an example of how to read and write data to a file in C:

```
#include <stdio.h>

int main() {
  FILE *fp;
  char buffer[1024];

  fp = fopen("file.txt", "r+");
  fprintf(fp, "This is a new file\n");
  rewind(fp);
  fgets(buffer, 1024, fp);
  printf("The contents of the file are: %s\n", buffer);
  fclose(fp);
```

```
  return 0;
}
```

In this example, we have opened a file named file.txt in read and write mode using the fopen() function. We have written a string to the file using the fprintf() function, then we have read the string back from the file using the fgets() function. We have then printed the contents of the file to the console.

In summary, file handling is the process of reading and writing data to and from a file in a computer. To create a file in C, we use the fopen() function, which takes two arguments: the file name and the mode. The different file operations include reading and writing data to a file using different modes of operation, such as read, write and append.

Ques 21: Discuss the concept of text and binary files in respect of C language with an example.

Ans: In C programming language, files can be broadly classified into two types: text files and binary files.

1. Text Files:
Text files are files that contain data that is readable text. Text files are composed of characters and are usually created in a plain text format. They can be opened using a basic text editor, such as Notepad.

Here's an example of how to create a text file and write data to it:

```
#include <stdio.h>

int main() {
  FILE *fptr;
  char str[] = "This is a text file\n";

  fptr = fopen("textfile.txt", "w");
  fprintf(fptr, "%s", str);
  fclose(fptr);

  return 0;
}
```

In this example, we have used the fopen() function to create a text file named textfile.txt in write mode. We have then used the fprintf() function to write a string to the file and then closed the file using the fclose() function.

2. Binary Files:
Binary files are files that contain data in binary format. Binary files are usually created by programs that write binary data to a file. Binary files cannot be opened using a basic text editor, but must be opened using a specific program or application.

Here's an example of how to create a binary file and write data to it:

```
#include <stdio.h>

int main() {
  FILE *fptr;
  int num = 100;

  fptr = fopen("binaryfile.bin", "wb");
  fwrite(&num, sizeof(num), 1, fptr);
  fclose(fptr);

  return 0;
}
```

In this example, we have used the fopen() function to create a binary file named binaryfile.bin in write mode. We have then used the fwrite() function to write an integer to the file and then closed the file using the fclose() function.

In summary, text files in C are files that contain readable text and are composed of characters This makes it easy for a human to read and interpret. Binary files, on the other hand, are files that contain data in binary format. They are usually created by programs that write binary data to a file. Text files are stored in a plain text format and can be opened using a basic text editor, while binary files cannot be opened using a basic text editor and must be opened using a specific program or application that understands their binary format.