Question Bank

Structured and Object Oriented Programming Language

Mohatamim Haque

Review, 2023

1. (a) **Justify the statement, "C is a mid-level Language".**
   Ans : C is considered as a middle-level language because it supports the feature of both low-level and high-level languages. C language program is converted into assembly code, it supports pointer arithmetic (low-level), but it is machine independent (a feature of high-level).
   • A Low-level language is specific to one machine, i.e., machine dependent. It is machine dependent, fast to run. But it is not easy to understand.
   • A High-Level language is machine independent. It is easy to understand.

1. (b) Compare between 'iterative programming style' and 'recursive programming style'.

	Recursion	**Iteration**
Definition	A technique in which the function calls itself in its body to solve the problem, typically breaking into smaller and more manageable sub-problems.	Iteration is a technique that repetitively executes a code block until the condition is unmet.
Syntax	There is a termination condition specified.	The iteration format includes initialization, condition, and increment/decrement of a variable.
Control Structure	Function call stack	Looping Constructs (for, while etc.)
Problem-Solving Approach	Divide and Conquer	Sequential Execution
Time Complexity	Higher due to the overhead of maintaining the function call stack.	Lower than recursion due to the absence of the overhead of function calls.
Space Complexity	Higher than iteration.	Generally lower than recursion due to the absence of the overhead function calls.
Memory	Uses more memory	Uses less memory

1. **(c) Write a C language program to calculate the sum of digits in your inputted number. The sum should be returned by using a function.**
   Ans: Solution

2. **(a) Compare between a 'structure' and a 'class'. Show with an appropriate example, how does a class achieve data hiding**

Class	Structure
1. Members of a class are private by default.	1. Members of a structure are public by default.
2. An instance of a class is called an 'object'.	2. An instance of structure is called the 'structure variable'.
4. It is declared using the class keyword.	4. It is declared using the struct keyword.
5. It is normally used for data abstraction and further inheritance.	5. It is normally used for the grouping of data
6. NULL values are possible in Class.	6. NULL values are not possible

In C++, data hiding is achieved through access specifiers within a class definition. Here's an example that showcases how a class can hide its internal data members:

```
class Account {
private:
  double balance;
public:
 void deposit(double amount) {
   if (amount > 0) {
     balance += amount;
   }
 }
 bool withdraw(double amount) {
   if (amount > 0 && amount <= balance) {
     balance -= amount;
```

```
    return true;
  }
  return false;
 }
};
```

3. **(b) When should you use a const member function in your C++ class design? Provide real world scenarios where const member functions are beneficial.**

Ans: Using const member functions in C++ class design is crucial for ensuring that certain functions do not modify the state of the object. This helps in maintaining the integrity of objects and provides clear guarantees to the users of the class.

When to Use const Member Functions

1. Accessors: Use const member functions for getters or any function that retrieves data from the object without modifying it.
2. Utility Functions: Any function that performs operations or computations without altering the member variables should be marked as const.
3. Operator Overloads: Overloading operators that do not change the state of the object should be const.

Example:

```
class Point {
    private:
      int x;
      int y;
    public:
      Point(int xCoord, int yCoord) : x(xCoord), y(yCoord) {}
      int getX() const {
        return x;
      }
      int getY() const {
        return y;
      }
      void move(int dx, int dy) {
        x += dx;
        y += dy;
      }
};
```

In this example: getX and getY are const because they do not modify the state of the Point object.

4. **(c) Create a class called Counter to model a counter that counts things. The class should have one constructor function to initialize its only data member count to 0, and another constructor should initialize it to a fixed value. Another member function show() should display its value. The final meml:>er function incCount() should increment the value of the member data by 1. A mainO function should create two initialized and one uninitialized Counter objects. Then increment the Counter object values and display the final value of the Counter objects.**
   Ans : Solution

5. **(a) What is this pointer? Write down its usefulness with a suitable example.**
   Ans : The this pointer is a special keyword in C++ that acts as a hidden argument passed implicitly to all non-static member functions of a class. It provides a reference to the current object instance upon which the member function is being called.
   The usefulness of the "this" pointer lies in scenarios where you need to differentiate between a local variable and a member variable with the same name, or when you need to pass a reference to the current object to another function.
   Example:

```
class Counter {
public:
   int count;  // Data member
   void increment(int value) {  // Local variable 'value'
      count += value;  // Accesses data member count using this-> is not required here
      count += this->count;  // Explicitly accesses member variable count
   }
};
```

3 **(b) Is operator overloading feature of OOP programming paradigm a kind of polymorphism? Validate your answer.**
   Ans: Yes, operator overloading is a form of polymorphism in object-oriented programming (OOP).
   Polymorphism refers to the ability of different objects to respond to the same message (method call) in different ways. Operator overloading allows the same operator (like +, -, *, /, etc.) to behave differently depending on the operands involved, which is a form of polymorphism. For instance, the + operator can be overloaded to perform addition for numbers, concatenation for strings, or even combine objects in a user-defined class.

Therefore, operator overloading is indeed a kind of polymorphism in OOP.

3 (c) Referring to the Counter class mentioned in Q2(c), o~erload increment operator(++) in prefix form replacing the incCount() function. Also overload the minus operator (-) to deduce the difference between two Counter objects. Design the overloaded function such a way that all functions should return an object of the same class.

Ans: Solution

6. (a) State how a template function is different from an overloaded function?

Ans : Here's a breakdown of the key differences between template functions and overloaded functions in C++:

Template Functions:

- Use template parameters (T) to represent placeholder data types.
- A single template function can be used with various data types by specifying the actual data type during function call.
- Useful for writing generic code that works with different data types without code duplication.

Overloaded Functions:

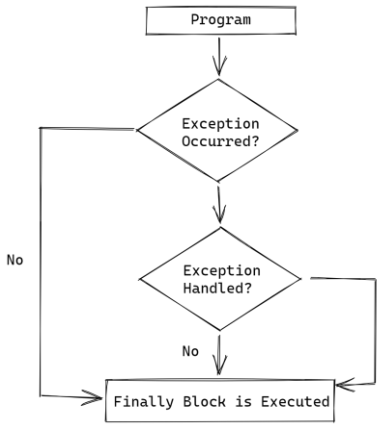
- Have the same name but different parameter lists (number, types, or order of parameters).
- The compiler selects the appropriate function based on the arguments provided during the call.
- Useful for functions that perform similar operations but on different data types or with a varying number of arguments.

4 (b) Write down a ~++ program that uses a template function that returns the average of all the elements of an array. The arguments to the function should be the array name and the size of the array (type int). In main() method, call the function with arrays of type int, long, double, and char.

Ans : Solution

4 (c) Show the general forms of try, catch, and throw. In your own words, describe their interaction with the help of block diagram.

```
try {
    - - - -- - - - - - - -- - - - - - --
    - - - -- - - - - - - -- - - - - - --
    throw exception_object;
    - - - -- - - - - - - -- - - - - - --
    - - - -- - - - - - - -- - - - - - --
}catch (exception_type variable) {
   // Code to handle the exception
}
```



Interaction Description:

- **try block**: The program enters the try block and executes the code within it.
- **throw statement**: If an exceptional situation occurs (e.g., division by zero), the throw statement is executed, and an exception object is thrown.
- **catch block**: The control is transferred to the catch block that matches the type of the thrown exception. The catch block contains the code to handle the exception.
- If no exception is thrown, the catch block is skipped.
- If an exception is thrown but no matching catch block is found, the program terminates.

5 (a) How many loop control structures exist in C programming? Provide a comprehensive description of each.

In C programming, there are primarily three types of loop control structures:

- **for loop : The for loop is used when the number of iterations is known before entering the loop. It includes initialization, condition, and increment/decrement expressions all in one line.**
- **while loop : The while loop is used when the number of iterations is not known beforehand. It checks the condition before executing the loop body.**
- **do-while loop : The do-while loop is similar to the while loop but checks the condition after executing the loop body.**

5 (b) Develop a program that identifies and prints all integer numbers within the range of 1 to 100, excluding those divisible by either 2 or 3. The program should also calculate and display the count of such integers.

Ans: Solution

7. (c) Write a C program that reads a string and print if it is a palindrome or not.

Ans: Solution

8. (a) What are the foundational concepts of Object-Oriented Programming (OOP), and how do classes and objects play essential roles in OOP? Please provide examples to illustrate these concepts.

- Encapsulation: Encapsulation is a way to restrict the direct access to some components of an object, so users cannot access state values for all of the variables of a particular object.
- Polymorphism: Polymorphism is the method in an object-oriented programming language that performs different things as per the object's class, which calls it
- **Inheritance** :Inheritance is a feature or a process in which, new classes are created from the existing classes. The new class created is called "derived class" or "child class" and the existing class is known as the "base class" or "parent class". The derived class now is said to be inherited from the base class.

6 (b) Write a program that demonstrate the utilization of constructor and destructor functions in OOP.

Ans : Solution

6 (c) Write a class called Distance to support the operations that are given in the following main function.

```
int main() {
    Distance dist1, dist3; \\define two lengths
     Distance dist2 (11, 6.25); \\ define and initialize dist2
    dist1.getdist(); // get dist1 from user
    dist3.add_dist (dist1, dist2) ; / /dist3 = dist1 + dist 2
    // display all lengths
    cout << "\ndist1 = "; dist1 .showdist();
    cout << "\ndist2 = "; dist2.showdist();
    cout << "\ndist3 = "; dist3.showdist();
    cout << endl; return O;
}
```

Ans: Solution

9. (a) Explain the difference between public, private, and protected inheritance in C++ with an example.

Ans: public, protected and private inheritance in C++

- **public inheritance** makes public members of the base class public in the derived class, and the protected members of the base class remain protected in the derived class.
- **protected inheritance** makes the public and protected members of the base class protected in the derived class.
- **private inheritance** makes the public and protected members of the base class private in the derived class.

Note: private members of the base class are inaccessible to the derived class.

7 (b) How does ambiguity arise in multiple inheritances in C++? Explain how this problem can be solved with an example.

Ans: Ambiguity arises in multiple inheritances in C++ when two or more base classes have a member function or data member with the same name. This can create confusion for the compiler as to which member should be accessed when the derived class tries to use that name.

Solution: Virtual Functions: We can solve this problem using the virtual keyword. When a member function is declared as virtual in the base class, it becomes a virtual function. This instructs the compiler to use a technique called virtual table dispatch to determine which function to call at runtime based on the object's actual type.

```
#include <iostream>
using namespace std;
class Base1 {
public:
   virtual void display() { cout << "Display from Base1\n"; }
};
class Base2 {
public:
   virtual void display() { cout << "Display from Base2\n"; }
};
class Derived : public Base1, public Base2 {
public:
   void display() override { Base1::display(); }  // or Base2::display()
};
int main() {
   Derived obj;
   obj.display();
```

}

7 **(c) Create a C++ program that employs virtual functions to determine a country's development status. A country is classified as 'developed' if it has an area of at least 200,000 square kilometers or savings of $100,000 or more in the World Bank. The program should use a base class 'World' and two derived classes, 'BigCountry' and 'DevelopedCountry ,' to represent different countries and their development statuses.**

Ans: solution

8. **(a) When and why would you use a friend function in a C++ class? Provide examples of scenarios where a friend function is beneficial.**

Ans: In C++, a friend function is a non-member function that is granted special access to the private and protected members of a class.

Accessing Private Members of Multiple Classes:

- When you need to access private members of multiple classes from outside, friend functions can be helpful.

```
class ClassA {
private:
   int privateDataA;
   friend void friendFunction(ClassA objA, ClassB objB);
};
class ClassB {
private:
   int privateDataB;
   friend void friendFunction(ClassA objA, ClassB objB);
};
void friendFunction(ClassA objA, ClassB objB) {
   cout << "Data from ClassA: " << objA.privateDataA << endl;
   cout << "Data from ClassB: " << objB.privateDataB << endl;
}
```

Operator Overloading:

- When overloading operators such as +, -, *, etc., and accessing private members of a class is necessary, friend functions are used.

```
class Temperature {
private:
   double value;
public:
   Temperature(double v) : value(v) {}
```

```
    friend double getFahrenheit(const Temperature& temp);
};

double getFahrenheit(const Temperature& temp) {
    return (temp.value * 9 / 5) + 32;
}
```

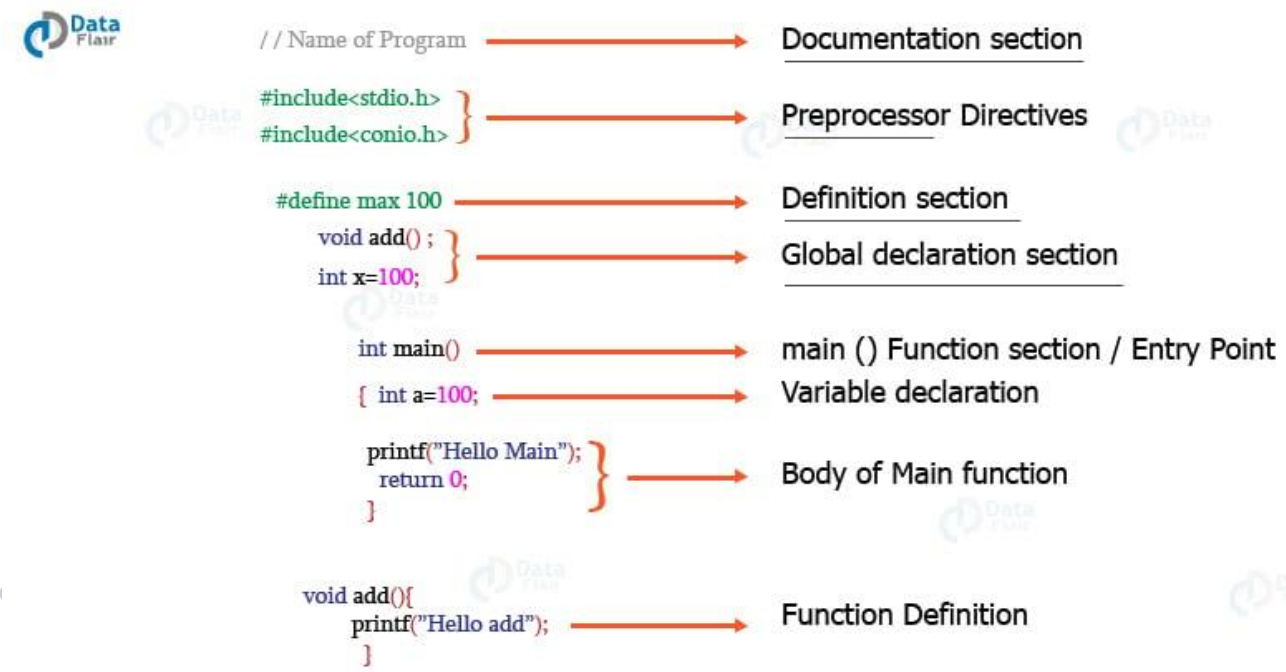
(Review) Examination, 2022

1. **(a) "C is a structured programming language"- Justify the statement.**

   Ans: The statement "C is a structured programming language" can be justified based on several key characteristics of the C programming language:

   - Functions: C allows defining functions that encapsulate specific tasks. These functions can take arguments (parameters) and return values, promoting modularity and code reuse.
   - Control Flow Statements: C offers a rich set of control flow statements like if, else if, switch, for, while, and do-while loops. These structures help control the program's execution flow based on conditions and repetitions.
   - Compound Statements: C allows grouping multiple statements using curly braces ({}). This creates code blocks that can be used as part of control flow statements or to define the body of a function, promoting structured organization.

1. **(b) Write the general structure of a C program. What is the significance of "stdio.h"?**



Regarding the significance of stdio.h:

- stdio.h stands for "Standard Input/Output Header."
- It is one of the most commonly used header files in C programming.
- It contains declarations of functions such as printf(), scanf(), getchar(), putchar(), etc., which are used for input and output operations.
- By including stdio.h in a C program, you gain access to these input/output functions, making it easier to perform console input and output operations.

1. **(c) What is output when the following code of fragments are executed?**

i.
```
int i= 5, j = 6, k = 7, n = 3;
printf("%d\n", i + j * k - k % n);
printf("%d\n", i / n);
```
Output: 46,1

ii.
```
int A =1;
int B =2;
if(((A==1) || (B==2)) && (B==0))
    printf("This exam was difficult");
else
   printf("This exam was easy");
```
Output: This exam was easy

iii.
```
string A = "Good morning";
string B = "Good afternoon";
if (A > B)
    printf("%s", A);
else
   printf("%s", B);
```
Output: Good morning

2. **(a) What is variable? Write the conditions for naming a variable. What is the difference between declaring a variable and defining a variable?**

Ans: A variable is a named location in memory that is used to hold a value that can be modified by the program.

Conditions for naming a variable:

- Variable names must start with a letter (uppercase or lowercase) or an underscore (_).
- Subsequent characters can be letters, underscores, or digits (0-9).
- Names are case-sensitive (e.g., age is different from Age).
- Reserved keywords (like int, if, while) cannot be used as variable names.
- It's strongly recommended to use descriptive and meaningful names to enhance code readability (e.g., totalCost, studentName).

Declaring a Variable:

- Declaration refers to stating the type of the variable without allocating memory or initializing it.
- Syntax: type variableName;
- Example: int x; - This declares a variable x of type int without assigning any value to it.

Defining a Variable:

- **Definition refers to both declaring the variable and allocating memory for it.**
- **Syntax: type variableName = value;**
- **Example: int y = 10; - This defines a variable y of type int and initializes it with the value 10.**

2. **(b) What is conditional operator? Write a C program to find the largest number among three integer numbers using conditional operator.**

Ans: The conditional operator in C, also known as the ternary operator, is a shorthand way of writing simple if-else statements.

Syntax: condition ? expression_if_true : expression_if_false;

Program : Solution

2 (c) Write a complete C program that will help a company to adjust salary of its employees. The program should: -

- First request for the salary

- Then request for P - if Part-time, F - if Full-time

If the employees is working as Full-time, then increase 10% of the current salary and display the -new salary and if the employee is working as a Part -time, then the current salary remains.

Ans: Solution

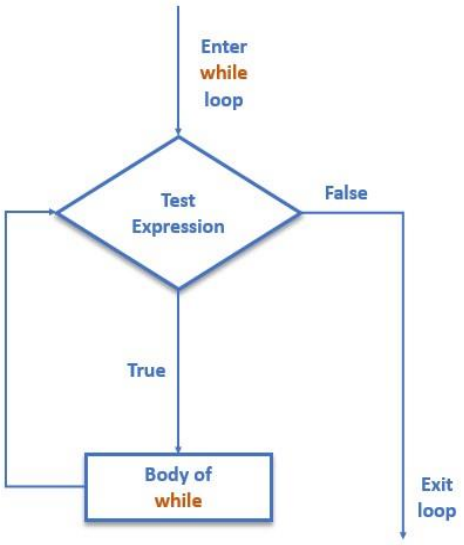
3. (a) Differentiate- between "entry-controlled loop" and "exit-controlled loop" with appropriate example and flow diagram. Specify their applications in problem solving.

Difference Between Entry Controlled and Exit Controlled Loops in C

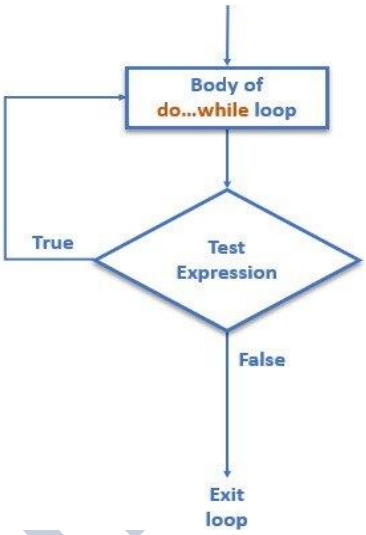
Entry Controlled Loop	Exit Controlled Loop
Test condition is checked first, and then loop body will be executed.	Loop body will be executed first, and then condition is checked.
If Test condition is false, loop body will not be executed.	If Test condition is false, loop body will be executed once.
for loop and while loop are the examples of Entry Controlled Loop.	do while loop is the example of Exit controlled loop.

Flow diagram:

Entry Controlled Loop:



Exit Controlled Loop:



Applications:

Entry Controlled Loop:

- Iterating a known number of times.
- Processing a list or array of elements until a specific condition is met.
- Getting validated user input.

Exit Controlled Loop:

- Ensuring at least one iteration (e.g., initialization, priming the loop).
- User input validation (repeat until valid input is provided).
- Menu-driven programs (loop continues until user chooses to exit).
- Iterative algorithms (repeat until a certain condition is met).

3 (b) Consider the following program fragment that involves a for loop.

I. Write exactly what this loop will print.

II. Rewrite this loop as a while loop that will print exactly the same thing as the for loop.

```
int k;
for(k=5;k>=1;k--){
        printf("%i",k);
        if(k != 1)
                printf(", ");
}
printf ("\n");
```

II.

```
int k=5;
while(k){
        printf("%i",k);
        if(k != 1)
                printf(", ");
        k--;
}
```

Output: 5, 4, 3, 2, 1

3 (c) Write a C program to print the number pattern of"n" heights. For example, if n is inputted as 5, then the output would be as follow:

```
A
A B
A B C
A B C D
A B C D E
```

Ans: Solution

4. (a) What is the difference between call by value and call by reference? Explain with the help of an example?

Call by Value

- When you call a function by value, a copy of the actual argument (variable) is passed to the function.
- Any changes made to the parameter within the function do not affect the original variable in the calling code.

Example :

```
void incrementByValue(int x) {
    x++; // Modify the copy of x passed to the function
}
int main() {
   int num = 5;
   cout << "Before function call: " << num << endl;
   incrementByValue(num);  // Pass a copy of num
   cout << "After function call: " << num << endl;
   return 0;
}
```

In this example, the incrementByValue function receives a copy of num (which is 5). Inside the function, x is incremented to 6, but this modification only affects the local copy x. The original num in main remains unchanged.

Call by Reference

- When you call a function by reference, you pass the **memory address** of the original variable to the function.
- Any changes made to the parameter within the function **directly modify the original variable**.
- Example:

```
void incrementByReference(int& ref) {
   ref++; // Modify the original variable through the reference
}
int main() {
   int num = 5;
   cout << "Before function call: " << num << endl;
   incrementByReference(num);  // Pass the address of num using reference
   cout << "After function call: " << num << endl;
   return 0;
}
```

Here, incrementByReference takes an int reference (ref). When num is passed using the reference, the function receives the memory address of num. Any changes to ref within the function will directly modify the variable at that memory location, which is the original num.

4 (b) Write a C program that uses a non-recursive function Reverse that takes an integer value from user and returns the number with its digits reversed. For example, if the given number is 5678, then function should return 8765.

Ans: Solution

4 (c) Write a program to reverse an integer array elements, i.e. element [0] would be element [n], element [1] would be element [n-1] and so on.

Ans: Solution

5. (a) Consider the following three strings with memory allocation:

STRING1 =

C	O	M	P	U	T	E	R		\0										

STRING2=

D	E	P	A	R	T	M	E	N	T		\0		

STRING3 =

S	T	U	D	E	N	T		\0								

What will be the output after execution the following statements:

i) strcat(STRINGl, STRING2)

ii) strcat(STRINGl, STRING3)

iii) strcJ\t(SJ:RING3, " GOOD")

Ans:

i. **STRING1 = COMPUTER DEPARTMENT**
ii. **STRING1 = COMPUTER DEPARTMENT STUDENT**
iii. **STRING3 = STUDENT GOOD**

5 (b) Explain the string handling functions with an 'example of each.

1. **strcpy:**
   a. **Purpose:** Copies a string from source to destination.
   b. **Prototype:** strcpy(char *dest, const char *src);

```
char src[] = "Hello, World!";
char dest[50]; // Ensure the destination array is large enough
strcpy(dest, src);
```

2. **strcat**
   a. **Purpose:** Concatenates (appends) one string to the end of another.
   b. **Prototype:** strcat(char *dest, const char *src);

```
char dest[50] = "Hello";
char src[] = ", World!";
strcat(dest, src);
```

3. **strcmp**
   a. **Purpose**: Compares two strings lexicographically.
   b. **Prototype:** strcmp(const char *str1, const char *str2);

```
char str1[] = "Hello";
char str2[] = "World";
 int result = strcmp(str1, str2);
if (result < 0) {
    printf("str1 is less than str2\n");
} else if (result > 0) {
    printf("str1 is greater than str2\n");
} else {
    printf("str1 is equal to str2\n");
}
```

4. **strlen**
   a. **Purpose:** Returns the length of a string.
   b. **Prototype:** strlen(const char *str);

```
char str[] = "Hello, World!";
length = strlen(str);
```

5. **strncpy**
    a. **Purpose:** Copies a specified number of characters from source to destination.
    b. **Prototype:** strncpy(char *dest, const char *src, size_t n);

```
char src[] = "Hello, World!";
char dest[50];
strncpy(dest, src, 5);
dest[5] = '\0'; // Null-terminate the destination string
```

6. **strncat**
    a. **Purpose**: Appends a specified number of characters from one string to another.
    b. **Prototype:** strncat(char *dest, const char *src, size_t n);

```
char dest[50] = "Hello";
char src[] = ", World!";
 strncat(dest, src, 7);
```

7. **strchr**
    a. **Purpose:** Finds the first occurrence of a character in a string.
    b. **Prototype:** strchr(const char *str, int c);

```
char str[] = "Hello, World!";
char ch = 'W';
char *pos = strchr(str, ch);
if (pos != NULL) {
    printf("Character '%c' found at position: %ld\n", ch, pos - str);
} else {
    printf("Character '%c' not found\n", ch);
}
```

8. **strstr**
    a. Purpose: Finds the first occurrence of a substring in a string.
    b. **Prototype:** strstr(const char *haystack, const char *needle);

```
char haystack[] = "Hello, World!";
 char needle[] = "World";
char *pos = strstr(haystack, needle);
if (pos != NULL) {
    printf("Substring '%s' found at position: %ld\n", needle, pos - haystack);
 } else {
    printf("Substring '%s' not found\n", needle);
}
```

5 (c) Write a program to find the ¥umber of vowels and consonants in a text string.
Ans : Solution

- **(a) Write the differences between structures and arrays.**

Structure in C	Array in C
A Structure is a data structure that can contain variables of different data types.	An Array is a data structure that can only contain variables of the same data type.
Structures do not use internal Pointers.	Arrays use internal Pointers that point to the first element in the array.
An object can be created from a Structure even after its declaration in the program.	Arrays do not allow object creation after their declaration.
Structures can include input variables of multiple data types.	Arrays can only include input variables of the same data type.
The operator to access elements in a Structure is the dot operator " **.** "	The operator to declare and access elements in an Array is the square bracket **[ ]**
Elements in a Structure can be of different sizes.	Elements in an Array are always of the same size.
The keyword "struct" is used to define a Structure.	Arrays do not require a keyword for declaration.
A Structure is a user-defined data type.	An Array is not user-defined and can be declared directly.

3 (b) How do you declare and access the structure variable? Explain arrays of structure with an example.

To declare and access structure variables, follow these steps:

- Define the Structure: Use the struct keyword.
- Declare Structure Variables: Create instances of the structure.
- Access Structure Members: Use the dot operator (.) to access members**.**

```
struct Person {// Define the Structure
   char name[50];
   int age;
   float height;
};
   struct Person person1;// Declare Structure
   strcpy(person1.name, "John Doe");
   person1.age = 30;// Access Structure Members
   person1.height = 5.9;// Access Structure Members
```

Arrays of Structures

An array of structures is a collection of structure variables. Each element in the array is a structure of the same type.

Example: Arrays of Structures

- Define the Structure: Define the structure type.
- Declare an Array of Structures: Declare an array where each element is a structure.
- Access and Manipulate Structure Members: Use array indexing and the dot operator.

```
struct Person { // Define a structure named Person
   char name[50];
   int age;
   float height;
};

 struct Person persons[3]; // Declare an array of structures of type Person
   strcpy(persons[0].name, "John Doe");    // Initialize the first person's details
   persons[0].age = 30;
   persons[0].height = 5.9;

   strcpy(persons[1].name, "Jane Smith");    // Initialize the second person's details
   persons[1].age = 25;
   persons[1].height = 5.5;

   strcpy(persons[2].name, "Alice Johnson");     // Initialize the third person's details
   persons[2].age = 27;
   persons[2].height = 5.7;

   for (int i = 0; i < 3; i++) {    // Access and print the details of each person
      printf("Person %d:\n", i + 1);
      printf("  Name: %s\n", persons[i].name);
      printf("  Age: %d\n", persons[i].age);
      printf("  Height: %.1f\n", persons[i].height);
```

6 (c) Describe three different approaches that can be used to pass structures as-function arguments

The three different approaches to passing structures as function arguments in C:

1. Passing by Value (Copying the Structure)

- A copy of the entire structure is created and passed to the function.
- Any changes made to the structure's members within the function do not affect the original structure in the calling code

2. Passing by Pointer

- The function receives the memory address of the original structure.
- Any changes made to the structure's members through the pointer directly modify the original structure.

```
struct Point {
   int x;
   int y;
};
 void modifyPointByValue(struct Point p) {
        p.x = 10;
        p.y = 20;
 }
void modifyPointByPointer(struct Point* ptr) {
   ptr->x = 10;
   ptr->y = 20;
}

int main() {
   struct Point pt = {5, 3};
   printf("Before function call: (%d, %d)\n", pt.x, pt.y);

   modifyPointByValue(pt);
   printf("After function call: (%d, %d)\n", pt.x, pt.y); // Output: (5, 3) (Original remains

   modifyPointByPointer(&pt); // Pass address using &
   printf("After function call: (%d, %d)\n", pt.x, pt.y); // Output: (10, 20) (Original is modified)
   return 0;
}
```

7 (a) Write a program using pointers to determine the length of a character string.

Ans: Solution

7. (b) How are pointers used with function? Explain with an example.

Ans: Pointers are used with functions in C and C++ to achieve various tasks such as passing data by reference, dynamic memory allocation, and creating more efficient code. They allow functions to indirectly access and modify variables outside their scope. Here's an example demonstrating the usage of pointers with functions:

```
#include <stdio.h>

// Function to swap two integers using pointers
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int x = 5, y = 10;
    printf("Before swap: x = %d, y = %d\n", x, y);

    // Pass the addresses of x and y to the swap function
    swap(&x, &y);

    printf("After swap: x = %d, y = %d\n", x, y);
    return 0;
}
```

- **swap Function**:
  - Accepts two integer pointers a and b.
  - Dereferences the pointers to access the values they point to.
  - Swaps the values indirectly by modifying the values at the addresses pointed to by a and b.
- **main Function**:
  - Declares two integer variables x and y.
  - Prints the values of x and y before swapping.
  - Calls the swap function, passing the addresses of x and y using the address-of operator (&).
  - Prints the values of x and y after swapping.

7. (c) Write the major benefits of using pointers in a C programming.
    1. **Dynamic Memory Allocation**:Pointers enable dynamic memory allocation, allowing programs to allocate memory at runtime as needed. This is crucial for tasks like creating dynamic data structures such as linked lists, trees, and dynamic arrays.

    2. **Efficient Memory Management**:Pointers allow for more efficient memory usage by enabling direct access to memory locations. This can lead to more optimized memory management compared to fixed-size data structures.

    3. **Passing Parameters by Reference**: Pointers enable passing parameters to functions by reference rather than by value. This allows functions to modify variables outside their scope, which is useful for functions that need to modify the original data.

    4. **Manipulating Arrays and Strings**: Pointers provide a powerful mechanism for manipulating arrays and strings. They can be used to iterate over array elements, access individual characters in strings, and perform various operations efficiently.

    5. **Implementing Complex Data Structures**: Pointers are essential for implementing complex data structures such as linked lists, trees, graphs, and dynamic arrays. These data structures rely heavily on pointers to establish relationships between elements and manage memory dynamically.

    6. **Reducing Code Size**: Pointers can help reduce code size by allowing the reuse of memory addresses and facilitating more concise code.

8. (a) What is file? Discuss the file modes with an example of each.

File: In C programming, a file is a named collection of data stored on a secondary storage device (like a hard disk) that persists even after the program terminates.

File Modes in C:

1. Reading Mode ("r"):

- Opens an existing file for reading only.
- The file pointer is positioned at the beginning of the file.
- Attempting to write to the file will result in an error.

2. Writing Mode ("w"):

- Opens a new file for writing.
- If the file exists, it will be truncated (cleared) before writing.

- The file pointer is positioned at the beginning of the file.

3. Appending Mode ("a"):

- Opens an existing file for appending.
- If the file doesn't exist, it will be created.
- The file pointer is positioned at the end of the file, allowing you to add new content without overwriting existing data.

4. Reading and Writing Modes ("r+", "w+", "a+"):

- These modes combine reading and writing capabilities.
- "r+": Opens an existing file for both reading and writing. The file pointer starts at the beginning.
- "w+": Opens a new file for reading and writing. If the file exists, it's truncated. The file pointer starts at the beginning.
- "a+": Opens an existing file for reading and appending. If the file doesn't exist, it's created. The file pointer starts at the end of the file.

```
FILE *fp = fopen("data.txt", "r"); //open for read
if (fp == NULL) {
  printf("Error opening file!\n");
  return 1;
}
char ch;
while ((ch = fgetc(fp)) != EOF) { // Read character by character until end-of-file (EOF)
  printf("%c", ch);
}
fclose(fp);
```

```
FILE *fp = fopen("output.txt", "w"); //open for w
fprintf(fp, "This is some text written to the file.\n");
fclose(fp);
```

```
FILE *fp = fopen("log.txt", "a"); //open for append mode
fprintf(fp, "Adding new data to the log file.\n");
fclose(fp);
```

```
FILE *fp = fopen("data.txt", "r+");
char buffer[100];
fscanf(fp, "%s", buffer); // Read existing content (if any)
fprintf(fp, "Modified data\n");  // Write new content
```

8 (b) Explain the operation of the following file handling functions:
i) putc() ii) fprintf() iii) getw() iv) fseek() v) rewind()

i) putc()

Operation: Writes a character to the specified file stream.
Syntax: int putc(int char, FILE *stream)
Example:

```
FILE *file = fopen("example.txt", "w");
if (file != NULL) {
    putc('A', file);
    fclose(file);
}
```

ii) fprintf():

Operation: Writes formatted output to the specified file stream.
Returns the number of characters written on success, or a negative value if an error occurs.
Example:

```
FILE *file = fopen("example.txt", "w");
if (file != NULL) {
    fprintf(file, "Hello, %s!", "world");
    fclose(file);
}
```

iii) getw()

Operation: Reads a word (integer) from a file stream.
Syntax: int getw(FILE *stream);
Example:

```
FILE *file = fopen("example.bin", "r");
if (file != NULL) {
   int num = getw(file);
   printf("Read number: %d\n", num);
   fclose(file);}
```

iv) **fseek()**

Operation: Moves the file position indicator to a specified location within a file.

Syntax: int fseek(FILE *stream, long offset, int origin);

Example :

```
FILE *file = fopen("example.txt", "r");
if (file != NULL) {
   fseek(file, 5, SEEK_SET); // Move to the 6th character in the file
   char ch = getc(file);     // Read the character at the new position
   printf("Character at position 6: %c\n", ch);
   fclose(file);
}
```

v) **rewind()**

Operation: Sets the file position indicator to the beginning of a file.

Syntax: void rewind(FILE *stream);

Example:

```
FILE *file = fopen("example.txt", "r");
if (file != NULL) {
   rewind(file);  // Move to the beginning of the file
   char ch = getc(file);
   printf("First character in the file: %c\n", ch);
   fclose(file);
}
```

8 (c) Suppose a file named "Data.txt" contains a series of integer numbers. Write a C program to 15 read these numbers and then write all "ODD" numbers to a file to be called "odd.txt" and all "EVEN" numbers to a file to be called "even.txt"

Ans: **Solution**

(Regular) Examination, 2022

1. **(a) Explain in detail call by value and call by reference with example.**

   Ans: Same as Review 2022 4(a)

1. **(b) Write a function in C/C++ that, when you call it, displays a message telling how many times 10 it has been called; "I have been called 3 times", for instance. Write a main () program that calls this function at least 10 times. [Hint: Use a local static variable].**

   Ans : Solution

1. **(c) Write a C language program using recursive function to enter 4 digit number and find the 15 sum of all digits of the number**

   Ans : Solution

2. **(a) Why some of the functions in a class are defined as Inlined? State the restrictions and advantages oflnline function.**

   Ans: Some functions in a class are defined as inline to reduce the overhead of function calls, improving performance by substituting the function body directly at the call site.

   Limitations of Inline Functions

   - The compiler may not perform inlining in such circumstances as:
   - If a function contains a loop. (for, while and do-while)
   - If a function contains static variables.
   - If a function is recursive.
   - If a function contains a switch or goto statement.

   Inline functions Advantages:

   - Function call overhead doesn't occur.
   - It also saves the overhead of push/pop variables on the stack when a function is called.
   - It also saves the overhead of a return call from a function.

   Inline function Disadvantages:

   - If we use many inline functions, then the binary executable file also becomes large.
   - Inline functions may not be useful for many embedded systems. Because in embedded systems code size is more important than speed.

2. **(b) A point on the two-dimensional plane can be represented by two numbers: an x coordinate and a y coordinate. For example, (4, 5) represents a point. The sum of two points can be defined as a new point whose x coordinate is the sum of the x coordinates of the two points, and whose y coordinate is the sum of the y coordinates.**
Create a class called Point to model a 2D point. The class should have one constructor to initialize these data to 0, and another should initialize them to fixed values. Another member function should display it, in (x, y) format. The final member function addPoint () should add two objects of type Point passed as arguments. A main () function should create two initialized and one uninitialized point objects. Then set the third point equal to the sum of the other two, and display the value of the new point.

Ans: Solution

2 (c) Define conversion function with syntax. For the class referred in question 2(b), define a conversion function so that a Point object could be converted into floating-point data, returning the summation of x and y coordinates.
Ans: A conversion function in C++ is a member function of a class that converts an object of that class to another data type.
Here's the basic syntax:

```
operator TargetType() const {
    // Conversion logic
}
```

Program : Solution

3 (a) What is this pointer? Write down its usefulness using a suitable example.
Ans: Same as review 2023 3(a)

3 (b) Is operator overloading feature of OOP model a kind of polymorphism? Validate your answer.
Ans: Same as review 2023 3(b)

3 (c) Referring to the class mentioned in question 2(b), overload Plus(+) operator replacing the addPoint () function. Also overload the Minus(-) and increment operator(++) in prefix form (assume that++ operator will increase only y-coordinate of the point by 1). Design the overloaded function such a way that all functions should return an object of the same class.
Ans: Solution

4 (a) State how a template function is different from an overloaded function?
Ans : Same as review 2023 4(a)

4 (b) Write down a C++ program that uses a template function that returns the average of all the elements of an array. The arguments to the function should be the array name and the

size of the array (type int). In main (), call the function with arrays of type int, long, double, and char.
Ans: Same as review 2023 4(b)

4 (c) Write an interactive program to compute factorial of a number. The input value must be tested for validity. If it is negative, the user defined function factorial. () should raise an exception. You must use C++ exception handling mechanism to handle the wrong user input.
Ans : Solution

5 (a) How many loop control structures are used in C programming? Describe them properly.
Ans: Same as review 2023 5(a)

5 (b) The numbers in the sequence 1 1 2 3 5 8 13 21. .... are called Fibonacci numbers. Write a program using for loop to calculate and print the first m Fibonacci numbers.
Ans: Solution

5 (c) Write a program using a two-dimensional array to compute and print the following information from the sales .table of a shop given below:

I. Total value of sales by each girl.
II. Total value of each item sold.
III. Grand total of sales of all items by all girls.

	Item1	Item2	Item3
Salesgirl #1	310	275	365
Salesgirl #1	210	190	325
Salesgirl #1	405	235	240
	260	300	380

Ans: Solution

6. (a) Explain the significance of inheritance, encapsulation, and polymorphism - the three fundamental attributes of object-oriented programming - using your own words?
Ans: Same as review 2023 6(a)

6 (b) Write a program to copy one string to another using for loop and count the number of characters copied.
Ans: Solution

6 (c) Write a program that demonstrate the utilization of constructor and destructor functions in OOP.
Ans: Same as review 2023 6(b)

7. Same as review 2023 7

8. (a) What is friend function? When does a friend function become compulsory?
Ans: Same as review 2023 8(a)

8(b) The class Distance has two data member feet (int), inches (float). Write a program that uses a friend function to perform the following operations:

```
int main (){
   Distance dist(3, 6.0); //two-arg constructor (3'-6'')
   float sqft;
   sqft = square(dist); //return square of dist
   //display distance and square
   cout << "\nDistance = "; dist.showdist();
   cout << "\nSquare =" << sqft <<" square feet\n";
   return 0;
}
```

Ans: Solution

8(c) Write an OOP program to implement the push, pop, and display operation of a stack by using a class called Stack.
Ans: Solution

(Regular) Examination, 2021

1. **(a) Justify the statement- "C is a mid-level Language".**
   Ans: Same as Review 2023 1(a)

1. **(b) Show the use of 'continue' and 'break' statements with respect to iterative s~ructures in C with suitable example.**

   continue Statement

   The continue statement skips the remaining code inside the current iteration of the loop and proceeds with the next iteration. This can be useful when you want to skip certain iterations based on a condition.

   Example using continue:

```
for (int i = 1; i <= 10; i++) {
    // Skip the current iteration if i is even
    if (i % 2 == 0) {
        continue;
    }
    printf("%d ", i);
}
```

   break Statement

   The break statement terminates the loop immediately and transfers control to the statement following the loop. This is useful when you need to exit the loop early based on a condition.

   Example using break:

```
for (int i = 1; i <= 10; i++) {
    // Break the loop if i is greater than 5
    if (i > 5) {
        break;
    }
    printf("%d ", i);
}
```

1. (c) Write a C program to take an integer number input and find the largest digit in the number and print that number in word with appropriate message.
   Sample Input: Enter an Integer: 5472
   Sample Output: Seven is the largest digit

   Ans: Solution

2. (a) Compare between a 'Structure' and a 'class'. Show with an appropriate example, how do a class achieve data hiding.
   Ans: Same as review 2023 2(a)

2. (b) Define friend function with its properties? Explain with an example, in which situations, friend function is useful.
   Ans: Same as review 2023 8(a)

2. (c) Create a class called Time that has separate int member data for hours, minutes, and seconds. One constructor should initialize this data to 0, and another should initialize it to fixed values. Another member function should display it, in hh:mm:ss format. The • final member function addTime() .should add two objects of type time passed as arguments.
   A main() program should create two initialized time objects and one that isn't initialized. Then it should add the two initialized values together, leaving the result in the third time variable. Finally, it should display the value of this third variable.
   Ans: Solution

3. (a) What is operator overloading? Why is it needed?
   Ans: Operator overloading in C++ is a powerful feature that allows you to redefine the behavior of existing operators (+, -, *, etc.) for user-defined data types (classes and structures). This means you can make these operators work with your custom objects in a way that makes sense for your data type.
   Operator overloading offers several benefits:
   - Improved Readability: Code appears more natural and intuitive, e.g., using + to add complex numbers or concatenate strings, similar to built-in types.
   - Maintainability: Enhances code clarity and consistency, making it easier to understand and maintain.
   - Reusability: Integrates user-defined types with existing code, improving code reusability and seamless operation with overloaded operators.

3. (b) Referring to the class mentioned in Q2(c), overload Plus (+) operator replacing the addTime() function. Also overload increment operator(++) in postfix and prefix form.
   Ans : Solution

3 (c) What do you understand by function inlining? Without using Inline keyword, how a function be inlined?
   Function Inlining : Function inlining is an optimization technique used by compilers to improve the performance of a program. When a function is inlined, the compiler replaces the function call with the actual code of the function.

C++, functions can be inlined implicitly by the compiler during optimization (e.g., -O2 flag) if they're small and defined within the class for member functions. This compiler-driven inlining aims to improve performance without requiring the inline keyword.

4. (a) Demonstrate the exception handling mechanism in C++ with appropriate block diagram.
   Ans: Same as review 2023 4(c)

4. (b) Explain why using a default argument is related to function overloading. Identify the error in the following declaration.

   int f(int a=0, double balance);

   Ans: Using a default argument is related to function overloading because both techniques provide flexibility in function usage by allowing different numbers and types of parameters. Default arguments enable a single function to handle calls with fewer parameters by providing default values, thereby reducing the need for multiple overloaded functions. Function overloading achieves similar flexibility by defining multiple versions of a function with different parameter lists. By combining default arguments with function overloading, developers can cover a wide range of function calls with fewer function definitions.

   The error in the declaration int f(int a=0, double balance); is that it provides a default argument (a=0) for the first parameter (a) but not for the second parameter (balance). In C++, once you provide a default argument for a parameter, all subsequent parameters must also have default arguments.

4 (c) Develop an interactive program to compute square root of a number. The input value. must be tested for validity. If it is negative, the user defined function my_sqrt() should raise an exception.

Ans : Solution //problem

5. **(a) Explain the class and object concept in the programming language. How is it different from structured programming?**

   Ans : Class and object are fundamental concepts in object-oriented programming (OOP). Here's a breakdown of each:

- **Class**: A class is a blueprint or template for creating objects. It defines the properties (attributes) and behaviors (methods) that objects of that class will have. Think of a class as a cookie cutter that defines the shape and characteristics of cookies you can bake.
- **Object**: An object is an instance of a class. It is a concrete realization of the class blueprint, with its own unique set of values for the class attributes. Using the cookie analogy, an object is a specific cookie that has been baked using the cookie cutter

Object-oriented programming (OOP) builds with objects (data + behavior) unlike structured programming's focus on functions. OOP promotes data protection and code reuse through inheritance, while structured programming emphasizes code flow and control structures.

5. **(b)Create a class named "Family" which has three data member size, names, ages and four functions display(), avg(). The display function should print the average age of the family,_ the names and ages of the family member.**
   Also write a main function to create two objects of the family class and, find the eldest and youngest family member between two families
   Ans: **Solution** //problem

5 **(c) Write a program to reverse digits of a given a 32-bit signed integer.**
Original integer: 123
Reverse integer: 321
Ans: **Solution**

Q: Difference Between Class and Object:

Class	Object
Class is used as a template for declaring and creating the objects.	An object is an instance of a class.
When a class is created, no memory is allocated.	Objects are allocated memory space whenever they are created.
The class has to be declared first and only once.	An object is created many times as per requirement.
A class can not be manipulated as they are not available in the memory.	Objects can be manipulated.
A class is a logical entity.	An object is a physical entity.

Q: What is Constructor?

A **constructor** is a special method of a class or structure in object-oriented programming that initializes a newly created object of that type. Whenever an object is created, the constructor is called automatically.

Syntax of Constructor:

```
<class-name> (list-of-parameters){
    // constructor definition
}
```

Q: Characteristics of Constructor

- Must have same name as class name.
- Constructor member is scoped public
- The name is case-sensitive
- Constructors do not have a return type.

- The default constructor is implicitly created.
- When creating an object, the constructor gets called automatically.
- Constructor is not implicitly inherited.
- Constructors can be overloaded.
- Constructor can not be declared virtual.

Q: Types of Constructor in C++

Constructors can be classified based on in which situations they are being used. There are 4 types of constructors in C++:

1. **Default Constructor**
2. **Parameterized Constructor**
3. **Copy Constructor**
4. **Move Constructor**

Default Constructor in C++: A default constructor is a constructor that doesn't take any argument. It has no parameters. It is also called a zero-argument constructor.

Parameterized Constructor in C++ : Parameterized Constructors make it possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created.

Copy Constructor in C++: A copy constructor in C++ is a special constructor that creates a new object as a copy of an existing object. The copy constructor typically takes a reference to an object of the same class as its parameter.

```
class MyClass {
public:
  int data;
  // Default constructor
  MyClass() {
    std::cout << "Default constructor called" << std::endl;
  }
  // Parameterized constructor
  MyClass(int d) : data(d) {
    std::cout << "Parameterized constructor called" << std::endl;
  }
  // Copy constructor
  MyClass(const MyClass& other) {
    data = other.data;
    std::cout << "Copy constructor called" << std::endl;
  }
};
```

Q: **What is a destructor?**

Destructor is an instance member function that is invoked automatically whenever an object is going to be destroyed. Meaning, a destructor is the last function that is going to be called before an object is destroyed.

- A destructor is also a special member function like a constructor. Destructor destroys the class objects created by the constructor.
- Destructor has the same name as their class name preceded by a tilde (~) symbol.

The syntax for defining the destructor within the class

```
//~ <class-name>(){}
```

Q: Properties of Destructor

- The following are the main properties of Destructor:
- The destructor function is automatically invoked when the objects are destroyed.
- It cannot be declared static or const.
- The destructor does not have arguments.
- It has no return type not even void.
- An object of a class with a Destructor cannot become a member of the union.
- A destructor should be declared in the public section of the class.
- The programmer cannot access the address of the destructor.

Q: When is the destructor called?

A destructor function is called automatically when the object goes out of scope:

- The function ends
- The program ends
- A block containing local variables ends
- A delete operator is called

Note: destructor can also be called explicitly for an object.

Q: How to call destructors explicitly?

We can call the destructors explicitly using the following statement:

object_name. ~class_name()

How are destructors different from normal member functions?

- Destructors have the same name as the class preceded by a tilde (~)
- Destructors don't take any argument and don't return anything

Q: Given a pointer to an object, what operator is used to access a member of that object?

In C++, the operator used to access a member of an object through a pointer is the arrow operator (->).

- `ptr->dataMember;`
- `ptr->memberFunction();`

Q: Show the general forms for new and delete. What are some advantages of using them instead of malloc() and free()?

In C++, the new and delete operators are used for dynamic memory allocation and deallocation, respectively. Here are their general forms:

'new' Operator:

pointer_variable = new data_type;

or

pointer_variable = new data_type(initialization_value);

2.

'Delete' Operator:

delete pointer_variable;

Advantages of new and delete over malloc and free:

Type Safety: new ensures the allocated memory is used for the intended data type. This helps prevent errors that might occur with malloc if you cast the pointer to the wrong type.

Constructor and Destructor Calls: new automatically calls the constructor of the object during allocation, allowing for proper object initialization. Similarly, delete calls the destructor before deallocation, malloc and free don't handle these tasks.

Improved Readability: The syntax of new and delete is more intuitive and reflects the object-oriented nature of C++.

Q: What is a reference? What is one advantage of using a reference parameter?

In C++, a reference is an alias, another name for an existing variable. It creates a direct connection to the memory location of the original variable.

Advantages of using References:

Easier to use: References don't need a dereferencing operator to access the value. They can be used like normal variables. The '&' operator is needed only at the time of declaration. Also, members of an object reference can be accessed with the dot operator ('.')

Q: Limitations of References

- Once a reference is created, it cannot be later made to reference another object; it cannot be reset. This is often done with pointers.

- References cannot be NULL. Pointers are often made NULL to indicate that they are not pointing to any valid thing.
- A reference must be initialized when declared. There is no such restriction with pointers.

Q: References vs Pointers

- A pointer can be declared as void but a reference can never be void For example
- The pointer variable has n-levels/multiple levels of indirection i.e. single-pointer, double-pointer, triple-pointer. Whereas, the reference variable has only one/single level of indirection. The following code reveals the mentioned points: Reference variables cannot be updated.
- Reference variable is an internal pointer.
- . Declaration of a Reference variable is preceded with the '&' symbol (but do not read it as "address of").

Q: Common Use References:

- **Modifying Object State:** In object-oriented programming (OOP), functions often need to modify the data or behavior of objects. Reference parameters allow direct access and modification of the original object within the function.
- **Passing Large Objects:** When dealing with large objects, copying them can be inefficient. Passing by reference avoids this overhead.
- Returning Multiple Values (Indirectly): While C++ doesn't directly support returning multiple values from functions, you can use reference parameters to achieve a similar effect.

Q: How do friend operator functions differ from member operator functions?
Friend operator functions and member operator functions are both used to overload operators in C++, but they have different characteristics and usage.

- Member Operator Functions:
  - Member operator functions are declared inside a class definition.
  - They have access to the private and protected members of the class directly.
  - They operate on the object for which they are called, and usually, the left operand of the operator is the object itself..
- Friend Operator Functions:
  - Friend operator functions are declared outside the class definition but inside the class declaration.
  - They are not members of the class but have access to the private and protected members of the class.
  - They do not have a this pointer and must accept parameters for both operands explicitly.

Q: Can the assignment operator be overloaded by using a friend function?

No, the assignment operator (=) cannot be overloaded by using a friend function in C++. There are two main reasons for this:

Member Access: The assignment operator needs direct access to modify the member variables of the object on the left side of the assignment. Friend functions don't have direct access to an object's private members.

Special Treatment: The assignment operator is a fundamental operation and C++ treats it differently. It requires a member function defined within the class itself.

Q:Function Overloading vs Function Overriding in C++

Function Overloading	Function Overriding
Function Overloading provides multiple definitions of the function by changing signature.	Function Overriding is the redefinition of base class function in its derived class with same signature.
An example of compile time polymorphism.	An example of run time polymorphism.
Function signatures should be different.	Function signatures should be the same.
Overloaded functions are in same scope.	Overridden functions are in different scopes.
Overloading is used when the same function has to behave differently depending upon parameters passed to them.	Overriding is needed when derived class function has to do some different job than the base class function.
A function has the ability to load multiple times.	A function can be overridden only a single time.
In function overloading, we don’t need inheritance.	In function overriding, we need an inheritance concept.

C

1. **Q: What is debugging? How is a syntax error different from a logical error? Which one is more difficult to handle and why?**
   Ans: Debugging is the process of identifying, analyzing, and removing errors or bugs from a computer program.
   Syntax errors break the language rules, like typos, while logical errors follow the rules but produce wrong results, like using the wrong word in a sentence.
   Logical errors are harder to catch. They run without errors but produce wrong results, making them like hidden bugs that require detective work.

2. **Q: Differentiate between else-if ladder and switch statement.**
   Ans:

if-else Statement	switch Statement
The condition which used in if-else can be any Boolean expression.	The expression in switch is usually an integral type or an enumerated type.
It can handle multiple conditions using else if clauses.	Designed for evaluating a single expression against multiple constants.
if-else doesn't have fall-through behavior by default.	switch can have fall-through behavior (case statements without breaks).
Optional else block for handling cases not explicitly covered.	Optional default case for handling cases not explicitly covered.
Suitable for complex, non-trivial conditions.	More concise for handling a series of conditions based on a single expression.

3. **Q: What is function? How is function defined? Explain the difference between calling function and called function?**

Ans: A function is a self-contained block of code that performs a specific task. It's like a mini-program within your main program.
A function in C++ is defined by specifying the return type, function name, and parameters (if any). The general syntax is:

```
return_type function_name(parameter_list) {
  // body of the function
}
```

Calling Function vs. Called Function

Calling Function:

- This is the function that makes a call to another function. It initiates the execution of the called function.
-

```
int main() {
   int result = add(5, 3); // 'main' is the calling function
   return 0;
  }
```

Called Function:

- This is the function that is invoked by another function (the calling function). It executes its own code and returns control to the calling function after execution.
- Example:

```
int add(int a, int b) {
   return a + b; // 'add' is the called function
  }
```

4. **Q: Differentiate between local variable and global variable? Demonstrate the scope of a static variable with a suitable example.**

Ans:

Local Variables	Global Variables
Limited to the block of code	Accessible throughout the program
Typically within functions or specific blocks	Outside of any function or block
Accessible only within the block where they are declared	Accessible from any part of the program
Created when the block is entered and destroyed when it exits	Retain their value throughout the lifetime of the program

A static variable in C has a local scope but a persistent lifetime. It retains its value between function calls, unlike a normal local variable which is reinitialized every time the function is called.

Here's an example demonstrating the scope and behavior of a static variable:

```
void counterFunction() {
   static int counter = 0;  // static variable initialization
   counter++;
   printf("Counter value: %d\n", counter);
}

int main() {
   counterFunction();  // First call
   counterFunction();  // Second call
   counterFunction();  // Third call
   return 0;
}
```

5. **Q: Define file. What are the steps to follow for file I/O system in C? Write down the usage of the functions with syntax:**
   i) rewind( ) ii) fseek( ) iii) feof( ) iv) ferror( )
   Ans: **Steps for File I/O System in C**
   File Input/Output (I/O) in C involves several steps to handle operations such as reading from and writing to files. Here are the typical steps:
   1. Opening a File: Use fopen() function to open a file and associate it with a stream.
   2. Reading or Writing Data: Use fread() or fwrite() functions to read from or write to the file.
   3. Closing the File: Use fclose() function to close the file stream when done.

iii) **feof**()
- **Purpose**: Checks if the end-of-file indicator for the file stream has been set.
- **Syntax**:
  int feof(FILE *stream);

iv) **ferror**()
- **Purpose**: Checks if an error has occurred for the file stream.
- **Syntax**:
  int ferror(FILE *stream);

6. **Q: What is the necessity to close a file properly?**
   Ans: Closing a file properly in C is necessary to ensure:

1. **Data Integrity**: It ensures all data buffers are flushed to the file, preventing potential data loss or corruption.
2. **Resource Management**: It releases system resources like file descriptors, preventing resource leaks and ensuring efficient use of system resources.
3. **File Locking**: It releases any locks on the file, allowing other processes or threads to access it, thus avoiding conflicts and delays in accessing the file.

7. **Q: What is the difference between structure declaration and structure initialization?**
   Ans: In C, structure declaration defines the blueprint of a structure, specifying the member variables and their data types. It allocates no memory. Structure initialization, on the other hand, allocates memory for a structure variable and assigns initial values to its members, either during declaration or separately.

8. **Q: Explain the effect of structure padding?**
   Ans: Structure padding in C inserts extra empty bytes between members to ensure data alignment. This improves performance for certain data types on some processors by allowing them to be accessed in fewer cycles. However, it can also increase the overall size of the structure, potentially wasting memory.

9. **Q: What is a pointer? What is the use of pointer in C. Explain how self-referential structure is related to pointer?**
   Ans: A pointer is a special variable that stores memory addresses. It acts like an arrow pointing to a specific location in memory where another variable or data resides.

   Uses:
   - **Dynamic Memory Allocation**: Pointers enable you to allocate memory at runtime using functions like malloc and calloc. This allows you to create data structures of varying sizes during program execution.
   - **Passing Arguments by Reference:** When you pass an argument to a function using a pointer, you're passing the memory address of the variable, allowing the function to directly modify the original data. This is more efficient than copying large data structures by value.
   - **Working with Arrays:** Arrays in C are essentially pointers that decay to the address of the first element. Pointers are often used to manipulate and iterate through array elements.

10. **Q: What is a preprocessor directive? Distinguish between function and preprocessor directive.**

Ans: Definition: Preprocessor directives are special instructions that are processed by the preprocessor, a separate program that runs before the actual C compiler.

Difference between Preprocessor Directives and Function

Pre-processor directives	Function Templates
There is no type checking	There is full type checking
They are preprocessed	They are compiled
They can work with any data type	They are used with #define preprocessor directives
They can cause an unexpected result	No such unexpected results are obtained.
They don't ensure type safety in instance	They do ensure type safety

11. **Q: Define C tokens. Classify C tokens and explain them with suitable examples**

Ans: A token in C can be defined as the smallest individual element of the C programming language that is meaningful to the compiler. It is the basic component of a C program.

Types of Tokens in C:

1.Keywords:

Definition: These are reserved words with predefined meanings in the C language. You cannot use them as variable names or for other purposes.

Examples: int, float, char, if, else, for, while, do, return, void, etc.

2.Identifiers:

- **Definition:** User-defined names given to variables, functions, arrays, structures, and other user-created entities. They must follow specific naming rules.
- **Examples:** main, calculateArea, studentName, array1, myStructure.

3. Constants:

- **Definition:** Fixed values that cannot be changed during program execution. They can be of various data types like integer, floating-point, character, or string literals.
- **Examples:**
    - **Integer constants:** 10, -25, 0xAB (hexadecimal)
    - **Floating-point constants**: 3.14159, 1.23e-5 (scientific notation)

4.Operators:

- **Definition:** Symbols that perform operations on operands (variables, constants, or expressions).
- **Examples:**
    - **Arithmetic operators:** +, -, *, /, % (modulo)
    - **Relational operators**: <, >, <=, >=, == (equal to), != (not equal to)
    - **Logical operators:** && (AND), || (OR), ! (NOT)
    - **Assignment operators**: =, +=, -=, *=, /=, etc**.**
    - **Bitwise operators:** & (AND), | (OR), ^ (XOR), ~ (NOT), etc.
    - **Increment/decrement operators:** ++, --

5.Special Characters:

- **Definition:** Symbols that have special meanings in the C language, often used for punctuation or control flow.
- **Examples:**
    - **Braces ({}):** Delimit code blocks within functions, loops, and other constructs.
    - **Parentheses (()) :** Used for function calls, expressions, and grouping of operations**.**
    - **Square brackets** ([]): Used for array indexing.
    - **Comma (,):** Separates items in lists (e.g., variable declarations, function arguments).
    - **Semicolon (;):** Terminates statements.

6. Strings:

- **Definition:** A sequence of characters enclosed in double quotes ("). They are treated as a single entity by the C compiler.
- **Examples:** "This is a string", "Hello\nworld" (includes newline character)

12. Q: What do you mean by operator precedence? What is the associativity of the arithmetic operators?

Ans: Operator precedence in C determines the order in which operations are evaluated within an expression. Operators with higher precedence are evaluated first.

Associativity, on the other hand, specifies the direction (left to right or right to left) for evaluating operators of the same precedence. In C, arithmetic operators (+, -, *, /) have left-to-right associativity. So, 2 + 3 * 4 evaluates to 14 (3 * 4 first, then 2 added).

13. **Q: Describe the limitations of using** getchar **and** scanf **functions for reading strings.**

Ans: Using getchar() and scanf() for reading strings in C has limitations:

1. **getchar():** Reads characters one by one, making it inefficient for reading entire strings or handling input with spaces or newlines. It requires additional logic to concatenate characters into a string.
2. **scanf():** Stops reading at whitespace, so it cannot capture strings with spaces or leading/trailing whitespace effectively. It also does not handle input that exceeds the specified buffer size, risking buffer overflow if not managed carefully.

14. **Q: Differentiate -between array and pointer? What are the arithmetic operators that are permitted on pointers? Are the expressions *ptr++ and ++*ptr same? Justify your answer**

Ans: Difference between Array and Pointer

Array:

- Arrays in C are a collection of elements of the same data type stored in contiguous memory locations.
- The size of the array is fixed and specified during declaration.
- Arrays are accessed using indexing (array[index]).

Pointer:

- Pointers in C are variables that store the memory address of another variable or data item.
- Pointers can point to arrays, individual variables, or dynamically allocated memory.
- Pointers can be dereferenced (*ptr) to access the value stored at the memory location they point to.

Arithmetic Operators Permitted on Pointers

In C, the following arithmetic operations are permitted on pointers:

1. **Increment (++) and Decrement (--):**
   - Moves the pointer to point to the next or previous memory location of its type.
2. **Addition (+) and Subtraction (-):**
   - Adds or subtracts an integer value from a pointer, adjusting its address accordingly.

ptr++ vs. ++*ptr:

These expressions are not the same and produce different results:

- ***ptr++:**

1. Dereferences the pointer ptr first, accessing the value at the current memory location it points to.
2. Then, increments the pointer ptr to point to the next memory location (based on the data type).

- **++*ptr:**
  1. Increments the pointer ptr first, moving it to the next memory location.
  2. Dereferences the pointer (now pointing to a new location), accessing the value at the new memory location.

15. **Q: Classify the conditional control statements in C with syntax and example.**
    Ans: if, if-else, if else-f else, switch

16. **Q:** Differentiate between entry-controlled loop and sentinel-controlled loop with appropriate example and flow diagram. Specify their applications in problem solving**.**
    Ans:
    Entry-Controlled Loop

- **Concept:** The loop's condition is checked at the beginning of each iteration. The loop continues as long as the condition remains true.
- **Syntax:**

```
for (initialization; condition; increment/decrement) {
        // code to execute in each iteration
}

while (condition) {
        // code to execute in each iteration
        // increment/decrement (optional)
}
```

- **Example:**

```
int i;
for (i = 0; i < 5; i++) {
      printf("Number: %d\n", i);
}
```

- **Applications:** When you know the exact number of iterations beforehand (e.g., iterating over an array of fixed size, printing a specific number of stars).

Sentinel-Controlled Loop

- **Concept:** The loop continues until a specific value (sentinel) is encountered as input. The condition is checked at the beginning of each iteration.

- **Syntax:**

```
int value;
while (value != sentinel_value) {
 // code to execute in each iteration
 printf("Enter a number (enter %d to stop): ", sentinel_value);
 scanf("%d", &value);
}
```

- **Example:**

```
int num, sum = 0;
printf("Enter numbers (enter 0 to stop):\n");
while (scanf("%d", &num) && num != 0) {
 sum += num;
}
printf("Sum of entered numbers: %d\n", sum);
```

Applications: When you don't know the number of iterations in advance, and the loop should continue until the user provides a specific signal

17. **Q: What do you mean by scope of a variable? What are local and global variables? Give examples.**
    Ans: The scope of a variable in C defines the region of your code where the variable's name has meaning and can be used.
    Local variables is variable that are accessible only within the function or block they're declared in.
    Global variables is variable that are accessible from anywhere in your program (except for nested header files).

18. **Q: Distinguish between an array of structures and an array within a structure. Give an example each**
    Ans:
    Array of Structures:
    - **Concept:** A collection of multiple structures of the same type, arranged in contiguous memory locations. Each element in the array is a complete structure.
    - **Example:**

```
struct Student {
 int id;
 char name[50];
```

```
};

struct Student students[3]; // Array of 3 Student structures

students[0].id = 1;  // Access and modify elements of individual structures
strcpy(students[0].name, "Alice");
students[1].id = 2;
strcpy(students[1].name, "Bob");
```

Array Within a Structure:

- **Concept: An array declared as a member variable within a structure. Each instance of the structure contains the array.**
- **Example:**

```
struct Course {
  int code;
  char title[50];
  int enrolled[100]; // Array within the Course structure to hold student IDs
};
struct Course math; // Create a Course structure
math.code = 101;
strcpy(math.title, "Calculus");
math.enrolled[0] = 123; // Access and modify elements within the array
math.enrolled[1] = 456;
```

19. **Q: What are the functionalities of the following tools in a typical program development environment:**
    i. Preprocessor ii. Linker iii. Interpreter v. Compiler
    Ans:

20. **Q: What is an operator? Classify operators of C programming Language.**
    Ans:

21. **Q: Wbat is an array? Classify it**
    Ans: