Technical

1. What is the output of this C program?

```
#include <stdio.h>
int main() {
   int a = 2, b = 7, c = 10;
   c = a > b ? a : b;
   c++;
   b = a + b;
   a = c + a;
   printf("%d, %d, %d", a, b, c);
   return 0;
}
```

A) 10, 9, 8

B) 8, 9, 10

C) 10, 9, 7

D) 9, 10, 8

Answer: A) 10, 9, 8

Explanation: 1. `c = a > b ? a : b;` becomes `c = 2 > 7 ? 2 : 7;`, so `c` becomes 7. 2. `c++;` increments `c` to 8. 3. `b = a + b;` becomes `b = 2 + 7;`, so `b` becomes 9. 4. `a = c + a;` becomes `a = 8 + 2;`, so `a` becomes 10. The final values printed are 10, 9, 8.

2. What will the following C program print?

```
#include <stdio.h>
int main() {
   char str[] = "Placement Drive";
   printf("%.5s", str);
   return 0;
}
```

A) Placement Drive

B) Place

C) Drive

D) Compilation Error

Answer: B) Place

Explanation: The format specifier `%.5s` tells `printf` to print a string, but at most 5 characters from it. It will print the first 5 characters of the string "Placement Drive".

3. Predict the output of this C code snippet.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int *p;
   p = (int*)calloc(1, sizeof(int));
   if (p)
      printf("Memory allocated: %d", *p);
   else
      printf("Allocation failed");
   free(p);
   return 0;
}
```

A) Memory allocated: 0

B) Memory allocated: (Garbage Value)

C) Allocation failed

D) Compilation Error

Answer: A) Memory allocated: 0

Explanation: The `calloc` function allocates memory and initializes all bits to zero. Therefore, when the integer pointer `p` dereferences this memory, it reads the value 0.

4. What is the output of the C code below?

```
#include <stdio.h>
int main() {
   const char *p = "12345";
   const char **q = &p;
```

```
    *q = "abcde";
    printf("%s %s", p, *q);
    return 0;
}
```

A) 12345 abcde

B) 12345 12345

C) abcde abcde

D) Compilation Error

Answer: C) abcde abcde

Explanation: `p` is a pointer to the constant string "12345". `q` is a pointer to the pointer `p`. `*q = "abcde";` changes the value of `p` itself, making it point to the new string literal "abcde". Both `p` and `*q` now refer to this new string.

5. What is the output of this C program?

```
#include <stdio.h>
int main() {
    int x = 0xAB; // Hexadecimal literal for 171
    int y = 012;  // Octal literal for 10
    printf("%d", x + y);
    return 0;
}
```

A) 119

B) 181

C) 183

D) Compilation Error

Answer: B) 181

Explanation: `0xAB` is a hexadecimal literal. `A` is 10 and `B` is 11. So, `10 * 16^1 + 11 * 16^0 = 160 + 11 = 171`. `012` is an octal literal. `1 * 8^1 + 2 * 8^0 = 8 + 2 = 10`. The sum is `171 + 10 = 181`.

6. What will be printed by the following C code?

```
#include <stdio.h>
struct test {
```

```
    int x:1;
};
int main() {
    struct test t;
    t.x = 1;
    t.x++;
    printf("%d", t.x);
    return 0;
}
```

A) 2

B) 1

C) 0

D) -1

Answer: D) -1

Explanation: This code uses a bit-field of size 1. A single bit can only represent two values, typically 0 and 1. When `t.x` (which holds the value 1, binary `1`) is incremented, it overflows. For a signed bit-field of size 1, the values it can hold are typically -1 and 0. Incrementing 1 causes it to wrap around. The bit pattern for 1 is `1`. Incrementing it makes it `10`. The lowest bit is `0`. But this is implementation-defined. Let's reconsider. The value `1` is stored. Incrementing it makes it `2` (binary `10`). Since the bit-field is only 1 bit wide, only the lower bit `0` is stored. However, this is for unsigned. For signed bit-fields, the behavior is more complex. The values can be -1 and 0. If `t.x = 1`, this is an overflow itself, storing `1` in a bit pattern that could mean -1. Let's assume the common representation where `1` represents -1. Then incrementing it would yield 0. If `0` is stored, incrementing yields 1. This is confusing. A better interpretation: `t.x=1` overflows a 1-bit signed field. The pattern stored is `1`, which represents -1. Incrementing -1 yields 0. Let's create a less ambiguous bit-field question.

6. What will be printed by the following C code?

```
#include <stdio.h>
struct test {
    unsigned int x:2;
};
int main() {
    struct test t;
    t.x = 3; // Binary 11
```

```
    t.x++;
    printf("%d", t.x);
    return 0;
}
```

A) 4

B) 3

C) 1

D) 0

Answer: D) 0

Explanation: The code uses an unsigned bit-field of size 2. A 2-bit field can hold values from 0 to 3. The value `t.x` is initialized to 3 (binary `11`). When it is incremented, it becomes 4 (binary `100`). Since the bit-field can only hold 2 bits, the value wraps around, and the lower 2 bits (`00`) are stored. Thus, `t.x` becomes 0.

7. Predict the output of this C code.

```
#include <stdio.h>
int main() {
    int arr[] = {1,2,3,4,5};
    int *p = arr + 4;
    printf("%d", p[-2]);
    return 0;
}
```

A) 5

B) 4

C) 3

D) Undefined Behavior

Answer: C) 3

Explanation: `p` is initialized to point to the last element of the array (at index 4). The expression `p[-2]` is equivalent to `*(p-2)`. This moves the pointer two elements backward from its current position, making it point to the element at index 2, which has the value 3.

8. What is the output of the following program?

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

```
int main() {
   int i = 5;
   do {
      printf("Loop ");
   } while (i == 0);
   return 0;
}
```

A) Loop

B) No output

C) Infinite loop

D) Compilation Error

Answer: A) Loop

Explanation: A `do-while` loop always executes its body at least once before checking the condition. The `printf` statement runs, printing "Loop ". Then the condition `i == 0` (which is `5 == 0`) is checked. It is false, and the loop terminates.

9. What is the output of this C program?

```
#include <stdio.h>
int main() {
   int i;
   for (i=0; i-5; i++) {
      printf("%d ", i);
   }
   return 0;
}
```

A) 0 1 2 3 4

B) 1 2 3 4 5

C) 0 1 2 3 4 5

D) Infinite loop

Answer: A) 0 1 2 3 4

Explanation: The loop's condition is `i-5`. In C, a zero value is false and a non-zero value is true. The loop continues as long as `i-5` is not zero. When `i` is 0, 1, 2, 3, and 4, the condition is non-zero (true). When `i` becomes 5, the condition `5-5` is 0 (false), and the loop terminates.

10. What does this C function call demonstrate?

```
#include <stdio.h>
#include <string.h>
int main() {
   char dest[10] = "Hello";
   char *src = "World12345";
   strncat(dest, src, 3);
   printf("%s", dest);
   return 0;
}
```

A) Buffer overflow

B) Safe string concatenation

C) String comparison

D) String tokenization

Answer: B) Safe string concatenation

Explanation: `strncat` is a safer version of `strcat`. It concatenates at most `n` characters (in this case, 3) from the source string to the destination string and adds a null terminator. It prevents buffer overflows by limiting the number of characters copied. The output will be "HelloWor".

11. What is the output of the following C++ program?

```
#include <iostream>
struct A {
   virtual void func() { std::cout << "A"; }
};
struct B : A {
   void func() { std::cout << "B"; }
};
int main() {
```

```
    B b;
    A &a = b;
    a.A::func();
    return 0;
}
```

A) A

B) B

C) Compilation Error

D) No output

Answer: A) A

Explanation: Even though `func` is virtual and `a` refers to a `B` object, the call `a.A::func()` uses the scope resolution operator `::` to explicitly call the version of `func` defined in class `A`. This bypasses the virtual dispatch mechanism.

12. Predict the output of this C++ program.

```
#include <iostream>
class Test {
public:
    int *p;
    Test() { p = new int(10); }
    ~Test() { delete p; }
};
int main() {
    Test t1;
    Test t2 = t1;
    return 0;
}
```

A) No error

B) Compilation Error

C) Runtime error (double free)

D) Memory leak

Answer: C) Runtime error (double free)

Explanation: The code violates the Rule of Three. When `t2` is initialized from `t1`, the default copy constructor is used, which performs a shallow copy. Both `t1.p` and `t2.p` now point to the same integer on the heap. When `main` exits, the destructors for `t1` and `t2` are called. The first destructor frees the memory. The second destructor attempts to free the same memory again, leading to a double free error and a crash.

13. What is the output of this code snippet?

```
#include <iostream>
#include <vector>
int main() {
   std::vector<int> v(3, 10);
   v.resize(5);
   for (int x : v) {
      std::cout << x << " ";
   }
   return 0;
}
```

A) 10 10 10

B) 10 10 10 0 0

C) 10 10 10 10 10

D) An exception is thrown

Answer: B) 10 10 10 0 0

Explanation: `v(3, 10)` initializes the vector with three elements, all with the value 10. `v.resize(5)` increases the size of the vector to 5. The new elements are value-initialized, which for integers means they are set to 0.

14. What does the following C++ program print?

```
#include <iostream>
int main() {
   int x = 10;
   auto l = [&x]() { x = 20; };
   l();
   std::cout << x;
```

```
    return 0;
}
```

A) 10

B) 20

C) 0

D) Compilation Error

Answer: B) 20

Explanation: The lambda expression captures the variable `x` by reference (`[&x]`). This means the lambda holds a reference to the original `x` variable in `main`. When the lambda is called, it modifies the original `x` through this reference, changing its value to 20.

15. What is the output of this C++ program?

```
#include <iostream>
struct S {
    S() { std::cout << "S"; }
    ~S() { std::cout << "~S"; }
};
int main() {
    S *s = (S*) ::operator new(sizeof(S));
    new (s) S();
    s->~S();
    ::operator delete(s);
    return 0;
}
```

A) S~S

B) S

C) ~S

D) Compilation Error

Answer: A) S~S

Explanation: This code demonstrates manual memory management using placement new. `::operator new` allocates raw memory. `new (s) S()` constructs an object in that memory, calling the constructor ("S"). `s->~S()` explicitly calls the destructor ("~S"). `::operator delete(s)` deallocates the raw memory.

16. What is the output of this C++ program?

```
#include <iostream>
struct A {
   int x;
   static int y;
};
int A::y = 10;
int main() {
   std::cout << sizeof(A);
   return 0;
}
```

A) 4

B) 8

C) 0

D) Compilation Error

Answer: A) 4

Explanation: `static` member variables are not part of the object's memory layout; they are stored separately and shared by all instances of the class. The `sizeof` an object only includes its non-static member variables. In this case, that's only the integer `x`, which is typically 4 bytes.

17. Predict the output of the code.

```
#include <iostream>
int main() {
   int n = 5;
   std::cout << (n << 1) << " " << (n >> 1);
   return 0;
}
```

A) 10 2

B) 2.5 10

C) 2 10

D) 10 3

Answer: A) 10 2

Explanation: `<<` is the bitwise left shift operator, and `>>` is the bitwise right shift operator. Left shifting by 1 is equivalent to multiplying by 2 (`5 * 2 = 10`). Right shifting by 1 is equivalent to integer division by 2 (`5 / 2 = 2`).

18. What is the output of the following C++11 code?

```
#include <iostream>
int main() {
   int arr[] {1, 2, 3};
   std::cout << sizeof(arr) / sizeof(arr[0]);
   return 0;
}
```

A) 3

B) 12

C) 1

D) Compilation Error

Answer: A) 3

Explanation: This is the standard idiom in C/C++ to calculate the number of elements in a statically allocated array. `sizeof(arr)` gives the total size of the array in bytes (e.g., 12 bytes if an int is 4 bytes). `sizeof(arr[0])` gives the size of a single element (4 bytes). Dividing the total size by the element size gives the number of elements.

19. What is the output of the program?

```
#include <iostream>
class A {
public:
   A& operator=(const A&) = delete;
   A() = default;
};
int main() {
   A a1, a2;
   a1 = a2;
   return 0;
```

}

A) No output

B) A single object is created

C) Compilation Error

D) Runtime Error

Answer: C) Compilation Error

Explanation: The line `A& operator=(const A&) = delete;` explicitly deletes the copy assignment operator for the class `A`. This means objects of type `A` cannot be assigned to one another. The line `a1 = a2;` attempts to use this deleted operator, which results in a compile-time error.

20. What is the result of running this C++ code?

```
#include <iostream>
#include <memory>
void func(std::shared_ptr<int> sp) {
   std::cout << "In func: " << sp.use_count() << " ";
}
int main() {
   auto p1 = std::make_shared<int>(10);
   std::cout << "Before: " << p1.use_count() << " ";
   func(p1);
   std::cout << "After: " << p1.use_count() << " ";
   return 0;
}
```

A) Before: 1 In func: 1 After: 1

B) Before: 1 In func: 2 After: 2

C) Before: 1 In func: 2 After: 1

D) Before: 1 In func: 1 After: 0

Answer: C) Before: 1 In func: 2 After: 1

Explanation: `std::shared_ptr` uses a reference count to track how many pointers share ownership. `p1` is created, count is 1. When `p1` is passed by value to `func`, a copy is made, and the reference count is incremented to 2. When `func` exits, its parameter `sp` is destroyed, and the count is decremented back to 1.

21. What is the output of this Java program?

```
public class Main {
   public static void main(String[] args) {
      String s = "a" + 'b' + 3;
      System.out.println(s);
   }
}
```

A) ab3

B) 100

C) 198

D) Compilation Error

Answer: A) ab3

Explanation: The expression is evaluated left-to-right. `"a" + 'b'` is string concatenation, resulting in the string "ab". Then, `"ab" + 3` is also string concatenation, resulting in "ab3".

22. What does the following Java program print?

```
public class Main {
   enum Color { RED, GREEN, BLUE }
   public static void main(String[] args) {
      System.out.println(Color.RED.ordinal());
   }
}
```

A) 0

B) 1

C) RED

D) An exception is thrown

Answer: A) 0

Explanation: The `ordinal()` method of an enum constant returns its zero-based position in the enum declaration. `RED` is the first constant, so its ordinal is 0.

23. What is the output of this Java code?

```
import java.util.Arrays;
```

```
public class Main {
    public static void main(String[] args) {
        int[] arr1 = {1, 2, 3};
        int[] arr2 = {1, 2, 3};
        System.out.println(arr1.equals(arr2));
    }
}
```

A) true

B) false

C) Compilation Error

D) No output

Answer: B) false

Explanation: For arrays, the `equals()` method is inherited from the `Object` class, and it checks for reference equality (`==`), not content equality. Since `arr1` and `arr2` are two different objects in memory, `equals()` returns `false`. To compare array contents, one must use `Arrays.equals(arr1, arr2)`.

24. What is the result of running this Java program?

```
public class Main {
    final int x;
    public Main() {
        x = 10;
    }
    public static void main(String[] args) {
        System.out.println("Done");
    }
}
```

A) Done

B) 10

C) Compilation Error

D) An exception is thrown

Answer: A) Done

Explanation: A `final` instance variable can be initialized either at the point of declaration or within the constructor. This code correctly initializes the final variable `x` in the constructor. The code compiles and runs, printing "Done".

25. What does this Java code print?

```
public class Main {
   public static void main(String[] args) {
      int i = 0;
      int j = (i < 1) ? 10 : 20.0;
      System.out.println(j);
   }
}
```

A) 10

B) 10.0

C) 20

D) Compilation Error

Answer: D) Compilation Error

Explanation: The ternary operator in Java has a rule that if one of the potential result expressions is a `double` (like `20.0`) and the other is an `int` (`10`), the type of the whole expression will be `double`. The code then tries to assign this `double` result to an `int` variable `j`, which is a narrowing conversion that requires an explicit cast `(int)`.

26. What will be the output of the following Java program?

```
import java.util.ArrayDeque;
public class Main {
   public static void main(String[] args) {
      ArrayDeque<Integer> stack = new ArrayDeque<>();
      stack.push(1);
      stack.push(2);
      System.out.println(stack.peek());
   }
}
```

A) 1

B) 2

C) null

D) An exception is thrown

Answer: B) 2

Explanation: `ArrayDeque`'s `push` method adds an element to the front, making it behave like a stack (LIFO). After pushing 1 then 2, the deque is `[2, 1]`. The `peek` method retrieves, but does not remove, the element at the head of the deque, which is 2.

27. What is the output of this program?

```
public class Main {
   public static void main(String[] args) {
      try {
         badMethod();
         System.out.print("A");
      } catch (RuntimeException ex) {
         System.out.print("B");
      } catch (Exception ex1) {
         System.out.print("C");
      } finally {
         System.out.print("D");
      }
      System.out.print("E");
   }
   public static void badMethod() {
      throw new RuntimeException();
   }
}
```

A) BDE

B) CDE

C) ADE

D) BCD

Answer: A) BDE

Explanation: `badMethod` throws a `RuntimeException`. The `try` block is exited. The first `catch` block for `RuntimeException` is a match, so "B" is printed. The `finally` block is always executed, so "D" is printed. Execution then continues after the try-catch-finally block, so "E" is printed.

28. What will be printed by this Java code?

```
public class Main {
   public static void main(String args[]) {
      int x = 5;
      assert x > 10;
      System.out.println("Passed");
   }
}
// Command line: java -ea Main
```

A) Passed

B) No output

C) An `AssertionError` is thrown

D) Compilation Error

Answer: C) An `AssertionError` is thrown

Explanation: The program is run with assertions enabled (`-ea`). The condition `x > 10` is false. This causes an `AssertionError` to be thrown, and the program terminates. The "Passed" message is never printed.

29. What is the output of this code?

```
public class Main {
   public static void main(String[] args) {
      Boolean b1 = new Boolean("true");
      Boolean b2 = new Boolean("tRuE");
      Boolean b3 = new Boolean("false");
      Boolean b4 = new Boolean("something");
      System.out.print(b1 + " " + b2 + " " + b3 + " " + b4);
   }
}
```

A) true true false false

B) true true false true

C) true true true true

D) Compilation Error

Answer: A) true true false false

Explanation: The `Boolean(String s)` constructor (now deprecated) creates a `Boolean` object. It results in `true` if the string argument is, case-insensitively, equal to "true". It results in `false` for any other string, including "false" and "something".

30. What does the following Java code output?

```
public class Main {
   public static void main(String[] args) {
      int a = -1;
      System.out.println(a >>> 1);
   }
}
```

A) -1

B) 2147483647

C) -2147483648

D) 0

Answer: B) 2147483647

Explanation: `>>>` is the unsigned right shift operator. It shifts the bits to the right and fills the leftmost bit with a 0, regardless of the initial sign bit. -1 in 32-bit two's complement is all 1s (`1111...1111`). Shifting right by one position and filling with a 0 results in `0111...1111`, which is the largest positive `int` value, `Integer.MAX_VALUE`.

31. A table `Customers` has a column `JoinDate`. What does this query do?

```
SELECT * FROM Customers WHERE YEAR(JoinDate) = 2024;
```

A) It finds all customers who joined in the year 2024.

B) It finds all customers whose join date is exactly '2024-01-01'.

C) It extracts the year from every join date and compares it to the integer 2024.

D) It's a valid query that performs A and C.

Answer: D) It's a valid query that performs A and C.

Explanation: The `YEAR()` function is a standard SQL function that extracts the year part from a date/datetime value. The `WHERE` clause then filters the rows, keeping only those where the extracted year is equal to 2024. The effect is to select all customers who joined in 2024.

32. What is the result of the following SQL query?

```
SELECT ROUND(123.456, -1);
```

A) 123.5

B) 123

C) 120

D) 100

Answer: C) 120

Explanation: The `ROUND` function rounds a number. When the second argument (precision) is negative, it rounds to the specified number of places to the left of the decimal point. `-1` means round to the nearest 10. `123.456` is closer to `120` than to `130`.

33. What does the `ON DELETE CASCADE` constraint do in a foreign key relationship?

```
CREATE TABLE Orders (
   OrderID int PRIMARY KEY,
   CustomerID int,
   FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON DELETE CASCADE
);
```

A) It prevents a customer from being deleted if they have orders.

B) When a customer is deleted, all corresponding orders are also automatically deleted.

C) When a customer is deleted, the `CustomerID` in their orders is set to `NULL`.

D) It deletes the `Orders` table when the `Customers` table is deleted.

Answer: B) When a customer is deleted, all corresponding orders are also automatically deleted.

Explanation: `ON DELETE CASCADE` is a referential integrity action. It specifies that if a row in the parent table (`Customers`) is deleted, then all rows in the child table (`Orders`) that have a matching foreign key value should also be deleted automatically.

34. What will be the output of this SQL query?

```
SELECT CONCAT('Hello', NULL, 'World');
```

A) 'HelloWorld'

B) 'Hello World'

C) 'Hello'

D) NULL

Answer: D) NULL

Explanation: In standard SQL, the `CONCAT` function returns `NULL` if any of its arguments are `NULL`.

35. A table `Grades` has columns `Student` and `Grade`. What is the result of this query?

```
SELECT Student,
   RANK() OVER (ORDER BY Grade DESC) as GradeRank
FROM Grades;
```

A) It assigns a unique rank to each student based on their grade.

B) It assigns a rank to each student, with potential gaps in the ranking for ties.

C) It assigns a rank to each student, with no gaps in the ranking for ties.

D) It calculates the average grade for each student.

Answer: B) It assigns a rank to each student, with potential gaps in the ranking for ties.

Explanation: The `RANK()` window function assigns a rank to each row within a partition. If two students have the same grade, they receive the same rank. The next rank assigned will skip the intervening numbers. For example, grades 95, 95, 90 would be ranked 1, 1, 3. `DENSE_RANK()` is the function that does not leave gaps.

36. What is the purpose of this SQL code block?

```
BEGIN TRANSACTION;
INSERT INTO Accounts (UserID, Balance) VALUES (101, 500);
INSERT INTO AuditLog (Action, UserID) VALUES ('New Account', 101);
COMMIT;
```

A) To execute two `INSERT` statements as a single, atomic unit of work.

B) To speed up the execution of the two `INSERT` statements.

C) To log the transaction details to a file.

D) To create a new user account if one does not already exist.

Answer: A) To execute two `INSERT` statements as a single, atomic unit of work.

Explanation: A transaction groups multiple SQL statements together. The `COMMIT` at the end makes all the changes permanent. If an error occurred between the two `INSERT` statements, a `ROLLBACK` could be issued to undo all changes, ensuring that the database is not left in an inconsistent state (e.g., an account exists without an audit log entry).

37. What does the following recursive C++ function do?

```
#include <iostream>
void print_binary(int n) {
   if (n > 1) {
      print_binary(n / 2);
   }
   std::cout << n % 2;
}
int main() {
   print_binary(10); // 10 is 1010
   return 0;
}
```

A) Prints the decimal number `n`.

B) Prints the hexadecimal representation of `n`.

C) Prints the binary representation of `n`.

D) Prints the sum of the digits of `n`.

Answer: C) Prints the binary representation of `n`.

Explanation: The function recursively calls itself with `n/2` until `n` is 1 or 0. As the recursion unwinds, it prints `n % 2`, which is the remainder when divided by 2 (either 0 or 1). This process effectively prints the bits of the number in the correct order. For 10, it prints "1010".

38. What is the output of this C++ code snippet?

```
#include <iostream>
#include <vector>
#include <algorithm>
int main() {
   std::vector<int> v = {1, 2, 3, 2, 1};
   int n = std::count(v.begin() + 1, v.end() - 1, 2);
   std::cout << n;
   return 0;
}
```

A) 0

B) 1

C) 2

D) 3

Answer: B) 1

Explanation: `std::count` counts the occurrences of a value within a given range. The range is specified by iterators. `v.begin() + 1` points to the second element (2). `v.end() - 1` points to the last element (1). The range is `[v.begin() + 1, v.end() - 1)`, meaning it includes elements from index 1 up to (but not including) index 4. The sub-vector being searched is `{2, 3, 2}`. The value `2` appears once in this range at index 1. Wait, let me re-check. The range is `{2, 3, 2}`. The count of `2` in this range is 2. My mistake. Let's fix this.

Answer: C) 2

Explanation: The range specified is `[v.begin() + 1, v.end() - 1)`. `v.begin() + 1` points to the element at index 1 (value 2). `v.end() - 1` points to the element at index 4 (value 1). The range for `std::count`

is half-open, so it includes the start iterator but excludes the end one. The elements considered are at indices 1, 2, and 3, which are `{2, 3, 2}`. The value `2` appears twice in this range.

39. What is the Big O complexity of Dijkstra's algorithm when implemented with a binary heap (priority queue)?

A) O(V^2)

B) O(E + V)

C) O(E log V)

D) O(V log E)

Answer: C) O(E log V)

Explanation: Dijkstra's algorithm finds the shortest paths from a source node to all other nodes in a weighted graph. When using a priority queue (implemented as a binary heap), the complexity is dominated by the extract-min operations (V times) and decrease-key operations (E times), both of which take O(log V) time. This results in a total time complexity of O(V log V + E log V), which simplifies to O(E log V) for connected graphs (where E \>= V-1).

40. What does this Java code demonstrate?

```
import java.util.stream.Collectors;
import java.util.stream.Stream;
public class Main {
    public static void main(String[] args) {
        String result = Stream.of("A", "B", "C")
                        .collect(Collectors.joining(","));
        System.out.println(result);
    }
}
```

A) Joining elements of a stream into a single string.

B) Collecting stream elements into a List.

C) Filtering elements of a stream.

D) Mapping elements to a different type.

Answer: A) Joining elements of a stream into a single string.

Explanation: This code uses the Java Streams API. `Collectors.joining(",")` is a terminal operation that concatenates the elements of the stream into a single `String`, using the provided delimiter (",") between elements. The output will be the string "A,B,C".

41. What is the output of the following C code?

```
#include <stdio.h>
int main() {
int x = 10;
{
int x = 20;
printf("%d, ", x);
}
printf("%d", x);
return 0;
}
```

A) 10, 10

B) 20, 20

C) 20, 10

D) 10, 20

Answer: C) 20, 10

Explanation: This demonstrates variable shadowing. The inner block `{}` defines a new variable `x` that is only visible within that scope. It "shadows" the outer `x`. The first `printf` prints the value of the inner `x` (20). Once the block is exited, the inner `x` is destroyed, and the outer `x` becomes visible again. The second `printf` prints the value of the outer `x` (10).

42. What is the output of this C++ program?

```
#include <iostream>
struct A {
int x;
};
int main() {
A a;
std::cout << a.x;
```

return 0;

}

A) 0

B) 1

C) A garbage value

D) Compilation Error

Answer: C) A garbage value

Explanation: When an object of a class/struct with a trivial default constructor (like `A`) is created on the stack, its member variables are not automatically initialized. The member `a.x` contains an indeterminate (garbage) value.

43. What is the output of this Java code snippet?

public class Main {

public static void main(String args[]) {

String s1 = "Test";

final String s2 = "Te";

String s3 = s2 + "st";

System.out.println(s1 == s3);

}

}

A) true

B) false

C) Compilation Error

D) An exception is thrown

Answer: A) true

Explanation: The Java compiler performs compile-time constant folding. Since `s2` is a `final` string variable initialized with a literal, it is a constant expression. The concatenation `s2 + "st"` is therefore performed at compile time, resulting in the constant string "Test". This is the same string literal as `s1`, so both `s1` and `s3` point to the same object in the string constant pool.

44. What will this SQL query return?

SELECT 'abc' = 'ABC';

A) true (or 1)

B) false (or 0)

C) An error

D) Depends on the database's collation settings

Answer: D) Depends on the database's collation settings

Explanation: String comparisons in SQL are governed by the collation of the database or column. A case-insensitive collation (like `_ci` in MySQL) would treat 'abc' and 'ABC' as equal, returning true. A case-sensitive collation (like `_cs` or `_bin`) would treat them as different, returning false.

45. What is the time complexity of finding the height of a skewed binary tree with N nodes?

```
int height(Node* node) {
   if (node == nullptr) return -1; // Or 0 for height by nodes
   return 1 + std::max(height(node->left), height(node->right));
}
```

A) O(1)

B) O(log N)

C) O(N)

D) O(N^2)

Answer: C) O(N)

Explanation: A skewed binary tree is one where each node has only one child, resembling a linked list. To find its height, the recursive function must traverse every single node from the root to the single leaf. This requires visiting all N nodes, resulting in a linear time complexity.

46. Predict the output of this C program.

```
   #include <stdio.h>
   int main() {
   char str[5] = "hello";
   printf("%s", str);
```

```
return 0;
}
```

A) hello

B) hell

C) Compilation Error

D) Undefined Behavior

Answer: D) Undefined Behavior

Explanation: The string literal "hello" requires 6 bytes of storage (5 for the characters and 1 for the null terminator). The array `str` is only allocated 5 bytes. This means the null terminator is not stored. When `printf` tries to print the string, it will read past the end of the allocated array looking for a null terminator, resulting in a buffer overflow and undefined behavior.

47. What is the output of the following C++ code?

```
#include <iostream>
int& func() {
int x = 10;
return x;
}
int main() {
int& y = func();
std::cout << y;
return 0;
}
```

A) 10

B) A garbage value

C) Compilation Error

D) A segmentation fault is likely

Answer: D) A segmentation fault is likely

Explanation: The function `func` returns a reference to a local variable `x`. When `func` returns, `x` goes out of scope and its memory on the stack is deallocated. The reference `y` in `main` is now a dangling reference, pointing to invalid memory. Dereferencing this reference (by printing it) is undefined behavior and will very likely cause a crash (segmentation fault). Most compilers will issue a warning for this code.

48. What is the result of running this Java code?

```
import java.util.Vector;
public class Main {
public static void main(String[] args) {
Vector<Integer> v = new Vector<>();
v.add(1);
System.out.println(v.capacity());
}
}
```

A) 1

B) 2

C) 10

D) 16

Answer: C) 10

Explanation: `Vector` is an older, synchronized collection class. When a `Vector` is created with its default constructor, its initial capacity is 10.

49. What does the following SQL query demonstrate?

```
SELECT Name, Salary
FROM (SELECT Name, Salary, Department FROM Employees) AS Sub;
```

A) A self-join

B) A window function

C) A derived table (subquery in the FROM clause)

D) A common table expression

Answer: C) A derived table (subquery in the FROM clause)

Explanation: The inner `SELECT` statement is a subquery. When a subquery is placed in the `FROM` clause, it acts as a temporary, virtual table for the outer query to select from. This is known as a derived table or an inline view.

50. What is the purpose of this DSA code?

```
public int factorial(int n) {
   int result = 1;
   for (int i = 2; i <= n; i++) {
      result *= i;
   }
   return result;
}
```

A) To calculate the nth Fibonacci number.

B) To calculate the factorial of a number iteratively.

C) To calculate a number raised to a power.

D) To find the greatest common divisor.

Answer: B) To calculate the factorial of a number iteratively.

Explanation: This code implements the factorial function (`n!`) using a loop. It initializes a result to 1 and then iteratively multiplies it by all integers from 2 up to `n`.

51. What is the output of this C program?

```
#include <stdio.h>
int main() {
int i = (1, 2, 3);
printf("%d", i);
return 0;
}
```

A) 1

B) 2

C) 3

D) Compilation Error

Answer: C) 3

Explanation: This code uses the comma operator. The comma operator evaluates its first operand, discards the result, then evaluates its second operand and returns that value. The expression `(1, 2, 3)` evaluates 1, discards it, evaluates 2, discards it, then evaluates 3 and the result of the whole expression is 3. This value is assigned to `i`.

52. What is the output of this C++ program?

```
#include <iostream>

int main() {

bool b = true;

b++;

std::cout<< b;

return 0;

}
```

A) 0

B) 1

C) 2

D) true

Answer: B) 1

Explanation: In C++, `bool` is an integral type. `true` is converted to 1 and `false` to 0. When `b++` is executed, the `bool` value `true` is promoted to the integer 1, incremented to 2. When this integer value 2 is converted back to `bool`, any non-zero value becomes `true`. When this `true` value is printed to `std::cout`, it is displayed as the integer 1.

53. What is the output of this Java code?

```
public class Main {

public static void main(String[] args) {

System.out.println('j' + 'a' + 'v' + 'a');

}

}
```

A) java

B) 418

C) A string containing ASCII values

D) Compilation Error

Answer: B) 418

Explanation: In Java, the `+` operator on `char` types performs integer addition, not string concatenation. The compiler promotes the `char`s to `int`s based on their ASCII/Unicode values and then adds them up. 'j'(106) + 'a'(97) + 'v'(118) + 'a'(97) = 418.

54. What will this SQL query return?

SELECT Department FROM Employees GROUP BY Department HAVING MIN(Salary) > 50000;

A) All departments where at least one employee earns more than 50000.

B) All departments where the average salary is greater than 50000.

C) All departments where every single employee earns more than 50000.

D) All departments where the lowest salary is exactly 50000.

Answer: C) All departments where every single employee earns more than 50000.

Explanation: The query groups employees by department. The `HAVING` clause then filters these groups, keeping only those where the minimum salary (`MIN(Salary)`) within that group is greater than 50000. If the lowest salary is above 50000, it means all salaries in that department must be above 50000.

55. What data structure is most suitable for implementing a task scheduler where tasks have different priorities?

A) Stack

B) Queue

C) Priority Queue

D) Linked List

Answer: C) Priority Queue

Explanation: A priority queue is an abstract data type where each element has an associated priority. The `dequeue` operation removes the element with the highest priority. This is the ideal data structure for scheduling tasks, as the scheduler can always pull the most important task to execute next.

56. Predict the output of this C program.

```
#include <stdio.h>

int main() {

if (sizeof(int) > -1)

printf("True");
```

```
else
printf("False");
return 0;
}
```

A) True

B) False

C) Implementation-defined

D) Compilation Error

Answer: B) False

Explanation: `sizeof` returns a value of an unsigned type (`size_t`). In the comparison `sizeof(int) > -1`, the signed integer `-1` is promoted to a large unsigned integer value. The result of `sizeof(int)` (e.g., 4) will be much smaller than this large unsigned value, so the comparison evaluates to false.

57. What is the output of the following C++ code?

```
#include <iostream>
class C {
public:
C() { std::cout << "C"; }
};
class D {
public:
D() { std::cout << "D"; }
C c;
};
int main() {
D d;
return 0;
}
```

A) D

B) C

C) CD

D) DC

Answer: C) CD

Explanation: When an object of class `D` is created, its member variables are initialized before its own constructor body is executed. The member variable `c` of type `C` is constructed first, printing "C". Then, the body of `D`'s constructor is executed, printing "D".

58. What is the result of running this Java code?

```
public class Main {
public static void main(String[] args) {
int x = 10;
int y = x > 10 ? ++x : --x;
System.out.println(x + "," + y);
}
}
```

A) 10,9

B) 9,9

C) 11,11

D) 10,10

Answer: B) 9,9

Explanation: The condition `x > 10` is false. The `else` part of the ternary operator is executed, which is `--x`. The value of `x` is pre-decremented to 9. This new value (9) is then assigned to `y`. Both `x` and `y` are 9.

59. What does the following SQL statement do?

```
DELETE FROM Employees;
```

A) It deletes the `Employees` table itself.

B) It deletes all rows from the `Employees` table.

C) It produces a syntax error because a `WHERE` clause is missing.

D) It deletes the first row of the `Employees` table.

Answer: B) It deletes all rows from the `Employees` table.

Explanation: A `DELETE` statement without a `WHERE` clause will remove all records from the specified table, but it will not remove the table structure itself.

60. What is the purpose of this DSA code?

```
#include <stack>
#include <string>
// Function to reverse a string using a stack
std::string reverse(std::string s) {
    std::stack<char> st;
    for (char c : s) {
        st.push(c);
    }
    std::string reversed_s = "";
    while (!st.empty()) {
        reversed_s += st.top();
        st.pop();
    }
    return reversed_s;
}
```

A) To check if a string is a palindrome.

B) To sort the characters of a string.

C) To demonstrate the LIFO property of a stack by reversing a string.

D) To count the number of characters in a string.

Answer: C) To demonstrate the LIFO property of a stack by reversing a string.

Explanation: The code iterates through the input string, pushing each character onto a stack. Because a stack operates on a Last-In, First-Out (LIFO) basis, the last character pushed will be the first one popped. By popping all characters from the stack and appending them to a new string, the original string is effectively reversed.