Technical

1. What is the output of this C program?

```
#include <stdio.h>

int main() {

    int a = 1, b = 1, c;

    c = a++ + b;

    printf("%d, %d", a, b);

    return 0;

}
```

A) 2, 1

B) 1, 2

C) 2, 2

D) 1, 1

Answer: A) 2, 1

Explanation: In the expression `a++ + b`, the post-increment operator `++` is used. The original value of `a` (1) is used in the addition `1 + 1`, so `c` becomes 2. As a side effect of `a++`, the value of `a` is then incremented to 2. The value of `b` remains unchanged.

2. What will the following C program print?

```
#include <stdio.h>

int main() {

    char *p;

    printf("%ld", sizeof(p));

    return 0;

}
```

A) 1

B) 2

C) 4 or 8

D) Compilation Error

Answer: C) 4 or 8

Explanation: The `sizeof` operator is evaluated at compile time. It returns the size in bytes of the given type or variable. In this case, `p` is a character pointer. The size of a pointer depends on the system architecture (32-bit or 64-bit). It will be 4 bytes on a 32-bit system and 8 bytes on a 64-bit system.

3. Predict the output of this C code snippet.

```
#include <stdio.h>

int main() {

   int i = 0;

   for ( ; i < 3; i++) {

      static int count = 10;

      count++;

      printf("%d ", count);

   }

   return 0;

}
```

A) 11 12 13

B) 10 11 12

C) 11 11 11

D) 10 10 10

Answer: A) 11 12 13

Explanation: The variable `count` is declared as `static`. This means it is initialized only once (to 10) when the program starts, and it retains its value between function calls (or in this case, between loop iterations). In each iteration, `count` is incremented and its new value is printed.

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

```
#include <stdio.h>

int main() {

   int x = 10, y = 5;

   if (x = y) {

      printf("%d", x);

   } else {

      printf("0");
```

```
    }
    return 0;
}
```

A) 10

B) 5

C) 0

D) Compilation Error

Answer: B) 5

Explanation: The expression in the `if` statement is an assignment (`x = y`), not a comparison (`x == y`). The value of an assignment expression is the value being assigned. So, `y` (which is 5) is assigned to `x`, and the expression evaluates to 5. Since 5 is a non-zero value, it is treated as true, and the `if` block is executed, printing the new value of `x`.

5. What is the output of this C program?

```
#include <stdio.h>
int main() {
    char s[] = "Hello";
    char *p = s;
    printf("%c", p[p[1]-p[0]]);
    return 0;
}
```

A) H

B) e

C) l

D) o

Answer: C) l

Explanation: `p[1]` is the character 'e' and `p[0]` is the character 'H'. In C, characters are treated as their ASCII integer values. ASCII('e') = 101, ASCII('H') = 72. The expression `p[1]-p[0]` evaluates to `101 - 72 = 29`. This is out of bounds for the string. Let's re-evaluate the premise of such a question. This is likely a trick.

Let's create a more reasonable question.

5. What is the output of this C program?

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

```
int main() {
    char s[] = "String";
    printf("%c", s[4]);
    printf("%c", 4[s]);
    return 0;
}
```

A) ng

B) nn

C) gg

D) Compilation Error

Answer: B) nn

Explanation: The array subscript operator `[]` is commutative. The expression `s[4]` is equivalent to `*(s + 4)`. This is the same as `*(4 + s)`, which can be written as `4[s]`. Both expressions access the character at index 4 of the string, which is 'n'.

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

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

A) 4

B) 8

C) 16

D) 6

Answer: C) 16

Explanation: The expression `x + 1` evaluates to `2 + 1 = 3`. The statement then becomes `x = x << 3;`, which is `x = 2 << 3;`. Left shifting the number 2 (binary `10`) by 3 positions results in `10000` in binary, which is 16 in decimal.

7. Predict the output of this C code.

```
#include <stdio.h>

int main() {

   int i;

   for(i=1; i<=10; i++);

   printf("%d",i);

   return 0;

}
```

A) 10

B) 11

C) 1

D) 12345678910

Answer: B) 11

Explanation: The semicolon after the `for` loop statement `for(i=1; i<=10; i++);` creates an empty loop body. The loop will execute 10 times, incrementing `i` each time. After the 10th iteration, `i` becomes 11. The condition `11 <= 10` is false, so the loop terminates. The `printf` statement then prints the final value of `i`, which is 11.

8. What is the output of the following program?

```
#include <stdio.h>

#define func(x) x * x

int main() {

   int i = func(2 + 3);

   printf("%d", i);

   return 0;

}
```

A) 25

B) 11

C) 13

D) 10

Answer: B) 11

Explanation: Macros perform simple text substitution. The line `int i = func(2 + 3);` is expanded by the preprocessor to `int i = 2 + 3 * 2 + 3;`. Due to operator precedence (multiplication before addition), this is calculated as `2 + (3 * 2) + 3`, which is `2 + 6 + 3 = 11`.

9. What is the output of this C program?

```
#include <stdio.h>

#include <string.h>

int main() {

    char str[10] = "Hi";

    printf("%ld %ld", sizeof(str), strlen(str));

    return 0;

}
```

A) 10 2

B) 2 2

C) 3 2

D) 10 3

Answer: A) 10 2

Explanation: `sizeof(str)` is a compile-time operator that returns the total allocated size of the array `str`, which is 10 bytes. `strlen(str)` is a runtime function that counts the number of characters in the string until it reaches the null terminator, which is 2 for "Hi".

10. What does this C code demonstrate?

```
#include <stdio.h>

int main() {

    char buffer[100];

    printf("Enter a line: ");

    fgets(buffer, 100, stdin);

    printf("You wrote: %s", buffer);

    return 0;

}
```

A) Unsafe reading of user input.

B) Reading a single character from the console.

C) Safe reading of a line of text, including spaces.

D) Reading a formatted integer from the user.

Answer: C) Safe reading of a line of text, including spaces.

Explanation: The `fgets` function is the recommended safe way to read a string from a stream. It takes the buffer, the maximum size to read (preventing buffer overflows), and the input stream (`stdin`) as arguments. Unlike `scanf("%s", ...)` it correctly reads entire lines, including spaces.

11. What is the output of the following 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: A) 0

Explanation: `A a = {};` is a form of value initialization (specifically, aggregate initialization in this case). For aggregates, value initialization zero-initializes all members. Therefore, `a.x` is guaranteed to be 0.

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

```
#include <iostream>

class MyClass {

public:

   MyClass() { std::cout << "C"; }

   MyClass(int i) { std::cout << i; }
};

int main() {
    MyClass arr[2];
    MyClass obj(5);
    return 0;
}
```

A) CC5

B) C5C

C) 5CC

D) Compilation Error

Answer: A) CC5

Explanation: `MyClass arr[2];` creates an array of two objects. Since no arguments are provided, the default constructor `MyClass()` is called for each, printing "C" twice. `MyClass obj(5);` creates a single object, calling the constructor `MyClass(int i)` with the argument 5, which prints "5".

13. What is the output of this code snippet?

```
#include <iostream>
#include <string_view>
int main() {
    std::string_view sv = "Hello World";
    sv.remove_prefix(6);
    std::cout << sv;
    return 0;
}
```

A) Hello

B) World

C) Hello World

D) An exception is thrown

Answer: B) World

Explanation: `std::string_view` provides a non-owning, view-only representation of a string. `remove_prefix(6)` modifies the view itself (not the original string data) to start 6 characters later. The view now refers to the substring "World".

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

```
#include <iostream>

struct A {

   ~A() { std::cout << "Destruct"; }

};

int main() {

   A* a = new A();

   // No delete

   return 0;

}
```

A) Destruct

B) No output

C) Compilation Error

D) Runtime Error

Answer: B) No output

Explanation: The object of type `A` is allocated on the heap using `new`. Heap-allocated objects are not automatically destroyed when their pointers go out of scope. Since `delete a;` is never called, the destructor for the object is never invoked, and a memory leak occurs.

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

```
#include <iostream>

#include <vector>

#include <algorithm>

int main() {

   std::vector<int> v = {1, 2, 3};

   std::rotate(v.begin(), v.begin() + 1, v.end());

   for(int x : v) std::cout << x;

   return 0;

}
```

A) 123

B) 321

C) 231

D) 312

Answer: C) 231

Explanation: `std::rotate` performs a left rotation on a range. The element pointed to by the middle iterator (`v.begin() + 1`, which is the element 2) becomes the new first element of the range. The elements before it are moved to the end. So, `{1, 2, 3}` becomes `{2, 3, 1}`.

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

```
#include <iostream>

class A {

public:

   int x;

   A(int i = 0) : x(i) {}

};

int main() {

   A a;

   A b = 10;

   std::cout << b.x;

   return 0;

}
```

A) 0

B) 10

C) A memory address

D) Compilation Error

Answer: B) 10

Explanation: A constructor that can be called with a single argument (like `A(int)`) acts as a conversion constructor by default. The line `A b = 10;` uses this to implicitly convert the integer `10` into an object of type `A` by calling the `A(int)` constructor.

17. Predict the output of the code.

```
#include <iostream>

int main() {
    char s[] = "abc";
    std::cout << sizeof(s);
    return 0;
}
```

A) 3

B) 4

C) 8 (size of a pointer)

D) Implementation-defined

Answer: B) 4

Explanation: `s` is an array of characters initialized from a string literal. The compiler sizes the array to fit the string, which includes the null terminator `\0`. So, the array contains {'a', 'b', 'c', '\0'}, and its size is 4 bytes.

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

```
#include <iostream>
#include <tuple>
int main() {
    auto my_tuple = std::make_tuple("hello", 10);
    std::cout << std::get<0>(my_tuple);
    return 0;
}
```

A) hello

B) 10

C) An exception is thrown

D) Compilation Error

Answer: A) hello

Explanation: `std::tuple` is a fixed-size collection of heterogeneous values. `std::get<N>(tuple)` is used to access the element at the Nth index (0-based) at compile time. `std::get<0>` correctly accesses the first element, which is the string literal "hello".

19. What is the output of the program?

```
#include <iostream>
```

```
int main() {
    int i = 1;
    if (i++ == 1)
        std::cout << "A";
    if (i++ == 2)
        std::cout << "B";
    if (i++ == 3)
        std::cout << "C";
    return 0;
}
```

A) A

B) AB

C) ABC

D) AC

Answer: B) AB

Explanation: 1. `if (i++ == 1)`: The value of `i` (1) is used in the comparison, which is true. "A" is printed. As a side effect, `i` becomes 2. 2. `if (i++ == 2)`: The value of `i` (2) is used in the comparison, which is true. "B" is printed. As a side effect, `i` becomes 3. 3. `if (i++ == 3)`: The value of `i` (3) is used in the comparison, which is true. "C" is printed. Oh wait, my logic is flawed. Let me re-trace.

Re-trace:

1. i is 1. `i++ == 1` is true. Print "A". `i` becomes 2.

2. i is 2. `i++ == 2` is true. Print "B". `i` becomes 3.

3. i is 3. `i++ == 3` is true. Print "C". `i` becomes 4.

The output should be ABC.

Let's rethink the question for clarity. A slightly different structure would be better.

19. What is the output of the program?

```
#include <iostream>
int main() {
    int i = 1;
    if (++i == 2)
```

```
        std::cout << "A";
    if (i++ == 2)
        std::cout << "B";
    if (i == 3)
        std::cout << "C";
    return 0;
}
```

A) A

B) B

C) C

D) ABC

Answer: D) ABC

Explanation: 1. `if (++i == 2)`: `i` is pre-incremented to 2. The comparison `2 == 2` is true. "A" is printed. `i` is now 2. 2. `if (i++ == 2)`: The current value of `i` (2) is used for the comparison, which is true. "B" is printed. As a side effect, `i` is incremented to 3. 3. `if (i == 3)`: The condition `3 == 3` is true. "C" is printed.

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

```
#include <iostream>
struct MyStruct {
    MyStruct() { std::cout << "C"; }
    MyStruct(const MyStruct&) = delete;
};
MyStruct func() {
    return MyStruct();
}
int main() {
    MyStruct s = func();
    return 0;
}
```

A) C

B) No output

C) Runtime Error

D) Compilation Error

Answer: A) C

Explanation: The copy constructor is deleted, which would normally prevent returning the object by value. However, since C++17, Guaranteed Copy Elision is mandatory in this specific scenario (returning a temporary). The compiler is required to elide the copy and construct the object directly in the destination `s`, so the deleted copy constructor is never needed.

21. What is the output of this Java program?

```
public class Main {
    public static void main(String[] args) {
        short s = 10;
        s = s + 5;
        System.out.println(s);
    }
}
```

A) 15

B) 10

C) An exception is thrown

D) Compilation Error

Answer: D) Compilation Error

Explanation: In the expression `s + 5`, the `short` `s` is automatically promoted to an `int`. The result of the addition is an `int`. The code then tries to assign this `int` result back to the `short` variable `s`, which is a narrowing conversion and is not allowed without an explicit cast (`s = (short)(s + 5);`).

22. What does the following Java program print?

```
public class Main {
    private static void test(int i) { System.out.print("I"); }
    private static void test(Integer i) { System.out.print("W"); }
    public static void main(String[] args) {
        int val = 10;
        test(val);
    }
}
```

A) I

B) W

C) It is ambiguous, Compilation Error

D) `NullPointerException`

Answer: A) I

Explanation: Java's overload resolution prefers widening over boxing/unboxing. The call `test(val)` with an `int` argument is a perfect match for the `test(int i)` method. It does not need to box the `int` to an `Integer` to call the other version.

23. What is the output of this Java code?

```
import java.util.ArrayDeque;

import java.util.Deque;

public class Main {

  public static void main(String[] args) {

    Deque<String> d = new ArrayDeque<>();

    d.add("a");

    d.addFirst("b");

    d.addLast("c");

    System.out.println(d);

  }

}
```

A) [a, b, c]

B) [c, a, b]

C) [b, a, c]

D) [a, c, b]

Answer: C) [b, a, c]

Explanation: `d.add("a")` is equivalent to `addLast`, so the deque is `[a]`. `d.addFirst("b")` adds to the front, making it `[b, a]`. `d.addLast("c")` adds to the end, making it `[b, a, c]`.

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

```
public class Main {
    public static void main(String[] args) throws InterruptedException {
        Thread t = new Thread(() -> System.out.print("Run "));
        t.start();
        t.join();
        System.out.print("End");
    }
}
```

A) Run End

B) End Run

C) Either A or B

D) Compilation Error

Answer: A) Run End

Explanation: `t.start()` begins the execution of the new thread. `t.join()` causes the main thread to pause and wait until the thread `t` has finished its execution. Therefore, "Run " is guaranteed to be printed before "End".

25. What does this Java code print?

```
public class Main {
    public static void main(String[] args) {
        int x = 1, y = 1;
        if (x++ < 2 || y++ < 2) {
            System.out.println("x=" + x + ", y=" + y);
        }
    }
}
```

A) x=2, y=1

B) x=2, y=2

C) x=1, y=2

D) x=1, y=1

Answer: A) x=2, y=1

Explanation: This demonstrates short-circuit evaluation with the logical OR `||` operator. The left side `x++ < 2` is evaluated. The current value of `x` (1) is used, so `1 < 2` is true. Because the first part of an OR is true, the entire expression must be true, so the right side `y++ < 2` is never evaluated. As a side effect of the left side, `x` is incremented to 2. `y` remains 1.

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

```
class X {
   X() { System.out.print("X"); }
}
class Y extends X {
   Y() { super(); System.out.print("Y"); }
}
public class Main {
   public static void main(String[] args) {
      new Y();
   }
}
```

A) YX

B) XY

C) X

D) Y

Answer: B) XY

Explanation: When a `Y` object is created, its constructor is called. The first line of any constructor is an implicit or explicit call to `super()`. The `Y` constructor calls `super()`, which executes the `X` constructor, printing "X". After the `X` constructor finishes, the rest of the `Y` constructor is executed, printing "Y".

27. What is the output of this program?

```
public class Main {
   public static void main(String[] args) {
      Integer i1 = Integer.valueOf("127");
      Integer i2 = Integer.valueOf("127");
      System.out.print(i1 == i2);
   }
}
```

A) true

B) false

C) Compilation Error

D) An exception is thrown

Answer: A) true

Explanation: The `Integer.valueOf()` method is required to cache `Integer` objects for values between -128 and 127 (inclusive). When it is called with "127", it returns a reference to the same cached object for both `i1` and `i2`. Since they are the same object, `==` returns true.

28. What will be printed by this Java code?

```
import java.util.stream.Collectors;

import java.util.stream.Stream;

public class Main {

  public static void main(String args[]) {

    System.out.println(Stream.of(1,1,2,3,3).collect(Collectors.toSet()));

  }

}
```

A) [1, 1, 2, 3, 3]

B) [1, 2, 3]

C) [3, 2, 1]

D) The output order is not guaranteed, but will contain 1, 2, 3

Answer: D) The output order is not guaranteed, but will contain 1, 2, 3

Explanation: The `Collectors.toSet()` collector gathers the stream elements into a `Set`. A `Set` does not allow duplicate elements, so the result will contain only the unique values 1, 2, and 3. The specific implementation of `Set` used by the collector is not specified (it's often a `HashSet`), so the iteration order of the resulting set is not guaranteed.

29. What is the output of this code?

```
public class Main {

  public static void main(String[] args) {

    String s1 = "a";

    final String s2 = "a";
```

```
        String s3 = s1 + "";
        String s4 = s2 + "";
        System.out.println(s3 == "a");
        System.out.println(s4 == "a");
    }
}
```

A) true true

B) false false

C) true false

D) false true

Answer: D) false true

Explanation: `s1` is a non-final variable. `s3 = s1 + ""` is a runtime concatenation, creating a new string object on the heap. So `s3 == "a"` is false. `s2` is a `final` variable initialized with a constant. `s4 = s2 + ""` is a compile-time constant expression. The compiler computes the result as "a" and uses the existing literal from the string pool. So `s4 == "a"` is true.

30. What does the following Java code output?

```
public class Main {
    public static void main(String[] args) {
        System.out.println(10.0 / 0.0);
        System.out.println(-10.0 / 0.0);
        System.out.println(0.0 / 0.0);
    }
}
```

A) Infinity -Infinity NaN

B) Infinity Infinity NaN

C) `ArithmeticException`

D) 0.0 -0.0 0.0

Answer: A) Infinity -Infinity NaN

Explanation: Java's floating-point arithmetic defines special values for division by zero. A positive number divided by zero is positive `Infinity`. A negative number divided by zero is negative `-Infinity`. Zero divided by zero is `NaN` (Not a Number).

31. What is the result of this SQL query?

```
SELECT 'Alpha' WHERE NULL;
```

A) 'Alpha'

B) NULL

C) An empty result set

D) An error

Answer: C) An empty result set

Explanation: The `WHERE` clause requires a boolean expression (true, false, or unknown). The value `NULL` is treated as `UNKNOWN`, which is not true. Since the condition is not met, no rows are selected, and the result is an empty set.

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

```
SELECT Name, Salary,
DENSE_RANK() OVER (ORDER BY Salary DESC) as Rank
FROM Employees;
```

A) It assigns a unique rank to each employee.

B) It assigns a rank to each employee, with gaps for ties.

C) It assigns a rank to each employee, with no gaps for ties.

D) It finds the densest concentration of salaries.

Answer: C) It assigns a rank to each employee, with no gaps for ties.

Explanation: The `DENSE_RANK()` window function is similar to `RANK()`, but it does not leave gaps in the ranking sequence when there are ties. For example, salaries of 100, 90, 90, 80 would be ranked 1, 2, 2, 3.

33. What will this SQL query do?

```
INSERT INTO Customers (ID, Name) VALUES (1, 'Alice')
ON CONFLICT (ID) DO UPDATE SET Name = 'Alice_Updated';
```

A) It will always insert a new customer named 'Alice'.

B) It will insert 'Alice' if ID 1 does not exist, otherwise it will update the name for ID 1.

C) It will update the name for ID 1 only if the current name is 'Alice'.

D) This is not valid standard SQL syntax.

Answer: B) It will insert 'Alice' if ID 1 does not exist, otherwise it will update the name for ID 1.

Explanation: This is an "upsert" operation, common in dialects like PostgreSQL and SQLite. The `ON CONFLICT` clause specifies what to do if the `INSERT` would violate a unique or primary key constraint (on the `ID` column in this case). The `DO UPDATE` action is then performed on the existing conflicting row.

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

```
SELECT REPLACE('ababab', 'ab', 'c');
```

A) cabab

B) c

C) ccc

D) ababab

Answer: C) ccc

Explanation: The `REPLACE` function finds all occurrences of the second argument within the first argument and replaces them with the third argument. All three instances of "ab" are replaced with "c".

35. A table `Logs` has a `LogTime` (datetime) column. What does this query find?

```
SELECT * FROM Logs WHERE LogTime BETWEEN '2024-10-01' AND '2024-10-31 23:59:59';
```

A) All logs from the month of October 2024.

B) All logs from October 1st and October 31st only.

C) All logs from October 2nd to October 30th.

D) It produces a syntax error.

Answer: A) All logs from the month of October 2024.

Explanation: The `BETWEEN` operator is inclusive. This query selects all rows where the `LogTime` is greater than or equal to the start of October 1st and less than or equal to the very end of October 31st, effectively covering the entire month.

36. What is the purpose of this SQL code?

```
GRANT SELECT, INSERT ON Employees TO 'some_user';
```

A) To create a new user named 'some_user'.

B) To give an existing user, 'some_user', permission to read and add data to the Employees table.

C) To copy the Employees table to a user's private schema.

D) To select all data inserted by 'some_user'.

Answer: B) To give an existing user, 'some_user', permission to read and add data to the Employees table.

Explanation: `GRANT` is a Data Control Language (DCL) command used to manage permissions in the database. This statement grants the `SELECT` (read) and `INSERT` (write) privileges on the `Employees` table to the specified user role.

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

```
#include <iostream>
#include <string>
std::string convert(int n) {
   if (n == 0) return "";
   return convert(n / 8) + std::to_string(n % 8);
}
int main() {
   std::cout << convert(75);      // 75 = 1*64 + 1*8 + 3
   return 0;
}
```

A) Prints the binary representation of `n`.

B) Prints the octal (base-8) representation of `n`.

C) Prints `n` in reverse.

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

Answer: B) Prints the octal (base-8) representation of `n`.

Explanation: The function recursively divides the number by 8 and concatenates the remainder (`n % 8`). This is the standard algorithm for converting a number from base-10 to any other base (in this case, base-8). `75` in octal is `113`.

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

```
#include <iostream>
#include <string>
int main() {
   std::string s = "Hello";
   s.push_back('!');
   s.pop_back();
   s.pop_back();
   std::cout << s;
   return 0;
}
```

A) Hello

B) Hell

C) Hel

D) Hel!

Answer: C) Hel

Explanation: `s` starts as "Hello". `push_back('!')` appends '!', making it "Hello!". `pop_back()` removes the last character, making it "Hello". The second `pop_back()` removes the 'o', leaving "Hell". Wait, I miscounted.

Re-trace:

1. s = "Hello"
2. s.push_back('!') -> s = "Hello!"
3. s.pop_back() -> s = "Hello"
4. s.pop_back() -> s = "Hell"

   Let me fix my proposed answer.

Correct Answer: B) Hell

Explanation: `s` starts as "Hello". `push_back('!')` appends '\!', making the string "Hello\!". The first `pop_back()` removes the '\!', changing the string back to "Hello". The second `pop_back()` removes the last character 'o', leaving "Hell".

39. What is the time complexity of the following algorithm?

```
// arr is an array of size n
public int findMax(int[] arr) {
    int maxVal = arr[0];
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] > maxVal) {
            maxVal = arr[i];
        }
    }
    return maxVal;
}
```

A) O(1)

B) O(log n)

C) O(n)

D) O(n log n)

Answer: C) O(n)

Explanation: This algorithm finds the maximum value in an array. It must iterate through every element of the array once to ensure it has found the maximum. Therefore, the number of operations is directly proportional to the size of the array, `n`.

40. What does this Java code demonstrate?

```
import java.util.regex.Pattern;
public class Main {
  public static void main(String[] args) {
    boolean matches = Pattern.matches("^[a-zA-Z]+$", "HelloWorld");
    System.out.println(matches);
  }
}
```

A) Checking if a string is a palindrome.

B) Using a regular expression to validate a string.

C) Splitting a string into tokens.

D) Searching for a substring.

Answer: B) Using a regular expression to validate a string.

Explanation: This code uses the `java.util.regex` package for regular expression matching. The pattern `^[a-zA-Z]+$` matches any string that consists of one or more (`+`) uppercase or lowercase letters from the beginning (`^`) to the end (`$`). Since "HelloWorld" fits this pattern, `matches` will be `true`.

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

```
  #include <stdio.h>
  int main() {
  char str[] = "Test";
  str[1] = 'a';
  printf("%s", str);
  return 0;
  }
```

A) Test

B) Tast

C) aest

D) Compilation Error

Answer: B) Tast

Explanation: `str` is declared as a character array on the stack, which is modifiable. `str[1]` accesses the second character of the array ('e') and changes its value to 'a'.

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

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

A) Inside

B) No output

C) Compilation Error

D) A warning is produced, and "Inside" is printed.

Answer: D) A warning is produced, and "Inside" is printed.

Explanation: The semicolon after `if (x == 5);` makes it an `if` statement with an empty body. The following block `{ std::cout << "Inside"; }` is just a regular scope block that is not attached to the `if`. It will always be executed, regardless of the `if` condition. Many compilers will warn about the empty `if` statement.

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

```
public class Main {
public static void main(String[] args) {
String s = "Hello";
System.out.println(s instanceof String);
```

```
   }
   }
```

A) true

B) false

C) Compilation Error

D) 1

Answer: A) true

Explanation: The `instanceof` operator checks if an object is an instance of a particular type. The object referenced by `s` is an instance of the `String` class, so the operator returns `true`.

44. What will this SQL query do?

```
SELECT * FROM Employees LIMIT 5;
```

A) It returns the 5 employees with the highest salaries.

B) It returns the first 5 employees inserted into the table.

C) It returns 5 arbitrary employees from the table.

D) It returns employees with an ID less than or equal to 5.

Answer: C) It returns 5 arbitrary employees from the table.

Explanation: The `LIMIT` clause restricts the number of rows returned. However, without an `ORDER BY` clause, the order of rows in a relational database table is not guaranteed. Therefore, the query will return 5 rows, but which 5 rows they are is not predictable.

45. What is the time complexity of a depth-first search (DFS) on a graph with V vertices and E edges?

```
void DFS(int u, Graph& g, vector<bool>& visited) {
   visited[u] = true;
   for (int v : g.adj[u]) {
      if (!visited[v]) {
         DFS(v, g, visited);
      }
   }
```

}

A) O(V^2)

B) O(E log V)

C) O(V * E)

D) O(V + E)

Answer: D) O(V + E)

Explanation: During a DFS traversal of a graph, each vertex is visited exactly once. When at a vertex, the algorithm iterates through all of its adjacent edges. Therefore, the total time complexity is the sum of the time to visit all vertices and the time to traverse all edges, which is O(V + E).

46. Predict the output of this C program.

```
#include <stdio.h>

int main() {

int x = 10 > 5 ? 1 : 0;

printf("%d", x);

return 0;

}
```

A) 10

B) 5

C) 1

D) 0

Answer: C) 1

Explanation: This code uses the ternary conditional operator. The condition `10 > 5` is true. Therefore, the first expression after the `?`, which is `1`, is evaluated and assigned to `x`.

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

```
#include <iostream>

int main() {

std::cout << "Hi\0there";

return 0;

}
```

A) Hi

B) Hithere

C) Hi there

D) Hi\0there

Answer: A) Hi

Explanation: The `<<` operator for C-style strings (`const char*`) stops printing when it encounters the first null character `\0`. Even though the string literal contains more characters, they are ignored by `cout`.

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

```
public class Main {
public static void main(String[] args) {
int x = 5;
x += x++;
System.out.println(x);
}
}
```

A) 10

B) 11

C) 12

D) 6

Answer: A) 10

Explanation: The compound assignment operator `+=` has a specific evaluation order. The left-hand operand (`x`) is evaluated first, its value is 5. The right-hand operand (`x++`) is evaluated next. Its value is 5, and as a side effect, `x` is incremented to 6. The addition `5 + 5` is performed, resulting in 10. This result is then assigned back to `x`.

49. What does the following SQL query demonstrate?

```
SELECT Name FROM Employees
EXCEPT
SELECT Name FROM Managers;
```

A) It returns all employees who are also managers.

B) It returns all employees and all managers.

C) It returns all employees who are not managers.

D) It produces a syntax error.

Answer: C) It returns all employees who are not managers.

Explanation: The `EXCEPT` set operator takes the distinct rows from the first `SELECT` statement and returns the rows that do not appear in the second `SELECT` statement's result set.

50. What is the purpose of this DSA code?

```
public boolean isPalindrome(String s) {
   int left = 0;
   int right = s.length() - 1;
   while (left < right) {
      if (s.charAt(left) != s.charAt(right)) {
         return false;
      }
      left++;
      right--;
   }
   return true;
}
```

A) To reverse a string.

B) To check if a string contains another string.

C) To sort the characters of a string.

D) To check if a string reads the same forwards and backwards.

Answer: D) To check if a string reads the same forwards and backwards.

Explanation: This is the two-pointer algorithm to check for a palindrome. It compares characters from both ends of the string, moving the pointers towards the center. If any pair of characters does not match, it returns false. If the loop completes, the string is a palindrome.

51. What is the output of this C program?

```
#include <stdio.h>

int main() {

float pi = 3.14159;

printf("%0.2f", pi);

return 0;

}
```

A) 3.14

B) 3.1

C) 3.14159

D) 03.14

Answer: A) 3.14

Explanation: The format specifier `%0.2f` controls the output of a float. The `.2` part specifies the precision, meaning it should print exactly 2 digits after the decimal point. The `0` is a flag that would normally mean zero-padding for width, but since no width is specified, it has no effect. The number is rounded to two decimal places.

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

```
#include <iostream>

int main() {

int i = 5;

while(i--) {

if (i == 2) continue;

std::cout << i;

}

return 0;

}
```

A) 54310

B) 4310

C) 43210

D) 430

Answer: B) 4310

Explanation: The `while(i--)` loop checks the value of `i` and then decrements it.

- i=5: condition is true, i becomes 4. print 4.
- i=4: condition is true, i becomes 3. print 3.
- i=3: condition is true, i becomes 2. `if (i == 2)` is true, `continue` skips the print.
- i=2: condition is true, i becomes 1. print 1.
- i=1: condition is true, i becomes 0. print 0.
- i=0: condition is false, loop terminates.

53. What is the output of this Java code?

```
public class Main {
public static void main(String[] args) {
System.out.println(String.join("-", "A", "B", "C"));
}
}
```

A) A,B,C

B) ABC

C) A-B-C

D) ["A", "B", "C"]

Answer: C) A-B-C

Explanation: `String.join(delimiter, elements)` is a static utility method that joins the given elements with the specified delimiter. It creates and returns the string "A-B-C".

54. What will this SQL query return?

```
SELECT 'Value' FROM Employees WHERE Department = 'Sales'
UNION ALL
SELECT 'Value' FROM Employees WHERE Department = 'HR';
```

A) The string 'Value' once.

B) The string 'Value' twice.

C) The string 'Value' for every employee in Sales and HR.

D) An error.

Answer: C) The string 'Value' for every employee in Sales and HR.

Explanation: The first subquery returns the literal 'Value' for each employee in the 'Sales' department. The second subquery does the same for the 'HR' department. `UNION ALL` combines these two result sets without removing duplicates. The final result will be a list of the string 'Value', with the number of rows equal to the total number of employees in both departments.

55. What data structure is commonly implemented using an array and follows the rule that for any element at index `i`, its left child is at `2*i + 1` and its right child is at `2*i + 2`?

A) Binary Search Tree

B) Stack

C) Binary Heap

D) Hash Table

Answer: C) Binary Heap

Explanation: This indexing scheme is the standard way to represent a complete binary tree (the structure required for a heap) within a flat array. It allows for efficient navigation between parent and child nodes without needing explicit pointers.

56. Predict the output of this C program.

```
#include <stdio.h>
int main() {
char a[3] = "ab";
char b[3] = "ab";
if (a == b)
printf("Equal");
else
printf("Not Equal");
return 0;
}
```

A) Equal

B) Not Equal

C) Compilation Error

D) Undefined Behavior

Answer: B) Not Equal

Explanation: In C, `a` and `b` are two distinct arrays, allocated in different memory locations. When an array name is used in an expression like this, it decays to a pointer to its first element. The `==` operator compares these two different memory addresses, which are not equal.

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

```
#include <iostream>
int main() {
const char* str = "Hello";
str = "World";
std::cout << str;
return 0;
}
```

A) Hello

B) World

C) H

D) Compilation Error

Answer: B) World

Explanation: `const char* str` declares `str` as a pointer to a constant character. This means you cannot modify the content the pointer points to (e.g., `str[0] = 'h'` would be an error). However, the pointer itself is not constant. You can change the pointer to make it point to a different memory location, such as the string literal "World".

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

```
public class Main {
public static void main(String[] args) {
System.out.println("Result: " + 10/0);
}
}
```

A) Result: Infinity

B) Result: 0

C) Compilation Error

D) A runtime `ArithmeticException` is thrown.

Answer: D) A runtime `ArithmeticException` is thrown.

Explanation: Unlike floating-point division, integer division by zero in Java is an illegal operation. It throws an `ArithmeticException` at runtime. The string "Result: " will not be printed.

59. What does the following SQL statement do?

```
SELECT Name FROM Employees ORDER BY RAND() LIMIT 1;
```

A) It selects the employee with the name 'RAND()'.

B) It selects a random employee from the table.

C) It sorts employees by name and picks one at random.

D) This is not valid SQL in most dialects.

Answer: B) It selects a random employee from the table.

Explanation: This is a common idiom, especially in MySQL, for selecting a single random row. `ORDER BY RAND()` sorts the entire table in a random order, and `LIMIT 1` then selects the first row from this randomized result. (Note: This can be inefficient on large tables).

60. What is the purpose of this DSA code?

```
#include <vector>
#include <numeric>
int kadane(const std::vector<int>& arr) {
    int max_so_far = 0;
    int max_ending_here = 0;
    for (int x : arr) {
        max_ending_here = max_ending_here + x;
        if (max_ending_here < 0) {
            max_ending_here = 0;
        }
        if (max_so_far < max_ending_here) {
            max_so_far = max_ending_here;
        }
    }
    return max_so_far;
}
```

A) To find the sum of all elements in an array.

B) To find the maximum element in an array.

C) To find the largest sum of a contiguous subarray.

D) To find the longest increasing subsequence.

Answer: C) To find the largest sum of a contiguous subarray.

Explanation: This is Kadane's algorithm. It efficiently solves the maximum subarray problem in linear time, O(n). It iterates through the array, keeping track of the maximum sum of a subarray ending at the current position (`max_ending_here`) and the overall maximum sum found so far (`max_so_far`).