# IBM面试题

## Common questions:

### 1. Tell me about yourself

I have done my Bachelor's in Computer Science and Engineering. I have focused my academic study on contemporary aspects of technology, such as data mining and machine learning. Java and C are two programming languages that I am proficient in.

### 2. Why should I hire you?

Being a fresher, I do not have any prior work experience. If you employ me, I'll be able to learn new skills and accomplish my ambitions. This is only going to be achievable if you employ me, and if you do, I promise to put in my best effort for this business.

### 3. What are your strong and weak points?

My strong points are that I am trustworthy, open-minded, and responsible, and I maintain a good perspective. My weak point is that I do not rest till I have completed my task.

### 4. Why are you interested in working for our company?

For me, it's a significant privilege to work for an esteemed worldwide firm like yours. It has a worldwide presence, with several locations. Your company's morale is high, which reflects well on its employees. It's a great opportunity for me to grow as a performer and learn new skills and techniques.

### 5. What distinguishes Confidence from Overconfidence?

Confidence suggests that I will win, whereas overconfidence says that I will always win.

### 6. What distinguishes Hard work from Smart work?

Because you can't become better at anything if you don't practise it, putting in plenty of hard work is an absolute need in your life.

## 7. What do you think about working overnight and at weekends?

If the firm needs my services, I am available to work whenever they need me.

## 8. Are you Capable of Working Under Pressure?

Indeed, it is a privilege for me to serve in a well-respected business such as yours since yours is an outstanding worldwide corporation. It has a significant number of locations all around the globe. Your company provides a very satisfying work environment for its employees. It's a good opportunity for me to showcase my skills while also expanding my knowledge and enhancing my expertise.

## 9. What are your aims?

My immediate objective is to find a position with your firm, and my long-term objective is to achieve success in all aspects of this organization's operations.

## 10. Would you lie for the company?

It's OK with me to lie if it's for the sake of the firm, but it shouldn't come at the expense of other people.

## 11. What are your pay expectations?

I may anticipate receiving anything that will allow me to cover my expenses.

## 12. In five years, where do you see yourself?

I can foresee myself progressing within this firm and achieving a position in which I become an invaluable contributor to this organization at some point in the future.

## 13. From one to 10, how would you evaluate me as an interviewer?

You have more knowledge, more skill, and more life experience than I have. I don't feel qualified to pass judgment on you.

## 14. Who has been your inspiration, and why?

My father is a source of motivation for me since he has consistently shown how one can go from having nothing to having everything. He has always been there to direct me in the right direction so that I may achieve success in all aspects of my life.

## Tech questions:

### 1. In the context of the C++ programming language, explain the major concepts of Object-Oriented Programming.

There are four major concepts of Object-oriented Programming in C++. They include the following:

- **Inheritance**

The ability of a class to derive features and traits from another class is referred to as inheritance. Inheritance is one of the most important features of Object-Oriented Programming. Subclasses and derived classes are classes that inherit properties from another class. A superclass or base class is a class whose properties and member functions are inherited by other classes. The concept of "reusability" is supported through inheritance.

Consider a group. The vehicle has all of the basic features that a vehicle must have. This covers things like speeding up, braking, shifting gears, and so forth. Assume that the classes Car, Bus, Truck, and so on exist. Because all of these classes must essentially possess all of the features of the class Vehicle, they can all be considered subclasses of it.

- **Encapsulation**

In Object-Oriented Programming, encapsulation is defined as the binding of data and the functions that alter it. Encapsulation means that a class's variables or data are hidden from other classes and can only be accessed through member functions of the class in which they are declared.

Consider the following situation:

There are various divisions in a corporation, such as accounts, finance, sales, and so on. The finance department's job is to maintain track of all financial information

and transactions. Similarly, it is the responsibility of the sales department to maintain track of all sales-related activity.

Let's pretend that a finance department official requests sales statistics for a given month. He cannot directly access the data in the sales area in this circumstance. Because only a sales department official has access to the data, he must be contacted. Encapsulation is depicted here by the process of requesting sales representatives for data. The data from the sales department, as well as the personnel who can affect it, are grouped under the heading "sales section."

- **Polymorphism**

It describes the existence of numerous versions of anything. In simple words, polymorphism refers to a message's ability to be displayed in numerous ways. A person might, for example, have a range of features at the same time. He is a father, a husband, and a worker all at the same time. As a result, the same person acts differently depending on the situation. This is known as polymorphism.

In C++, there are primarily two types of polymorphism. These are defined as follows:

- Compile Time Polymorphism

This type of polymorphism is achieved by overloading functions and operators.

- Runtime Polymorphism

This type of polymorphism is created by using Function Overriding.

- **Abstraction**

It gives only the most important elements while keeping the rest hidden. Data abstraction is the process of revealing only the most significant aspects of a dataset to the outside world while keeping the implementation details hidden.

Consider the situation of a man driving a car. The man only knows that pressing the accelerators increases the vehicle's speed and that applying the brakes stops it, but he has no understanding of how the speed is increased or how the accelerator, brakes, and other controls are implemented in the car. This is how abstraction is defined.

## 2. In the context of the C++ programming language, explain function overloading and overriding. Distinguish between them.

### Function Overloading

Modifying the signature, i.e., the number of parameters, the data type of the parameters, and the return type, allows for multiple definitions of the function.

### Function Overriding

Overriding a base class function in a derived class with the same signature, that is, the return type and parameters are known as function overriding. Only derived classes allow for this.

Major differences to be noted between these two:

- When one class inherits from another, functionalities are overridden. In the same class, overloading can occur.
- Overloaded functions must have a unique function signature, which means they must have a different number or type of parameters. When overriding, function signatures must be the same.
- Overloaded functions are in the same scope as overridden functions, whereas overridden functions are in different scopes.

The examples demonstrate both of these concepts well.

### Program Demonstrating Function Overloading in C++.

```
using namespace std;

void overloadedMethod(int x)

{

cout << "Inside Overloaded Method 1" << endl;

}

void overloadedMethod(float x)

{

cout << "Inside Overloaded Method 2" << endl;
```

```
}
void overloadedMethod(int x1, float x2)
{
cout << "Inside Overloaded Method 3" << endl;
}
int main()
{
int x = 5;
float y = 5.5;
overloadedMethod(x);
overloadedMethod(y);
overloadedMethod(x, y);
return 0;
}
```

## Output

```
Inside Overloaded Method 1
Inside Overloaded Method 2
Inside Overloaded Method 3
```

## Program Demonstrating Function Overriding in C++.

```
class Test
{
public:
virtual void print(){ cout << "Testing Function"; }
};
class Sample: public Test
{
```

```
public:
void print(){ cout << "Inside a Sample Function";}
};
int main()
{
Test obj = new Sample();
obj.print();
return 0;
}
```

## Output

Inside a Sample Function

## 3. What functions does an operating system perform?

An operating system's functions are as follows:

- **User interface**

Operating systems serve as a conduit between computer hardware and users. It allows the user to access the hardware in a structured manner.

- **Maintains system functionality**

Monitors overall system health to assist in increasing performance. Keep track of the time between service requests and system responses to gain a comprehensive view of the system's health. This will help with performance by providing important information for debugging.

- **Security**

Password protection and other security features are used by the operating system to protect user data. It also prevents unauthorized access to programs and user data.

- **Error detection**

The operating system continuously monitors the system in order to detect errors and keep the machine from falling.

- **Memory Management**

The operating system is in charge of managing primary memory, also known as main memory. The main memory is made up of a large number of bytes or words, each with its own address. Main memory is fast storage that the CPU has direct access to. Before a program can be executed, it must first be loaded into the main memory.

An operating system manages memory by performing the following tasks:

- It keeps track of primary memory usage, or which user programs use certain memory bytes. Memory addresses that have previously been assigned and those that have yet to be used.
- In multiprogramming, the OS sets the order in which processes are allowed memory access and for how long.
- When a process requests memory, it is allocated, and memory is released when the process exits or performs an I/O activity.
- The operating system determines the order in which processes access the processor and the amount of processing time each process has in a multiprogramming environment.
- **Device Management**

Through drivers, an operating system (OS) regulates device connectivity. It keeps track of all of the system's connected gadgets. The Input/Output controller is a program in charge of all devices. Determines which processes and for how long are permitted access to a device. Devices are distributed in an effective and efficient manner. When a device is no longer needed, it is deallocated.

- **File Management**

To make navigation and usage more effective, a file system is organized into directories. These directories may include additional directories and files. Among other things, the operating system keeps track of where data is stored, user access settings, and the condition of each file.

## 4. In the context of a computer, distinguish between primary and secondary memory.

### Primary/Main Memory

Primary memory is the computer memory that is directly accessible by the CPU. It is made up of DRAM (Dynamic Random Access Memory) and provides a real working space for the processor. It keeps track of the data and instructions that are currently being processed by the processor. RAM is one example (Random Access Memory)

### Secondary Memory

Because the processor does not directly interface with the secondary memory, the contents of the secondary memory must first be transferred to the primary memory before the processor can access it. Hard discs, USB drives, and other similar devices are examples.

The table below summarizes the differences between Secondary and Main/Primary Memory.

| Primary Memory | Secondary Memory |
|---|---|
| • Primary memory storage is just temporary. | • Secondary memory storage is permanent. |
| • Primary memory is immediately accessible to the processor/CPU. | • Secondary memory is not immediately accessible to the processor/CPU. |
| • It can be volatile (needs the power to keep the information stored) or non-volatile. | • It is always non-volatile (needs no electricity to keep the information stored) in nature. |
| • The memory devices used for main memory are semiconductor memories. | • Secondary memory devices include magnetic and optical memories. |
| • The cost of primary memory is higher than that of secondary memory. | • Secondary memory devices are less expensive as compared to primary memory devices. |

## 5. In the context of an operating system, what is meant by threads and processes?

## Process

A process is any program that is currently being executed. A process control block is in charge of controlling any process. The Process Control Block stores process priority, process id, process state, CPU, register, and other data (PCB). When one process spawns another, a new process is produced. A process takes longer to complete and is isolated, meaning it does not share a memory with other processes.

## Thread

A thread is a segment of a process, which means that a process can have multiple threads that are all contained within it. There are three states for a thread: running, ready, and blocked. Threads are faster than processes at terminating, yet they do not isolate like processes.

The table summarizes these two concepts briefly and compares them.

| Process | Thread |
|---|---|
| • It takes longer to create a process. | • A thread takes less time to create. |
| • It takes longer to change contexts from one procedure to another. | • It takes less time to switch between contexts from one thread to another. |
| • The method is inefficient in terms of intercommunication. | • Thread is more efficient in terms of intercommunication. |
| • Different processes make use of different memory locations. They don't have the same recollection. | • Memory is shared among threads in the same process. |
| • Each process has its own Process Control Block, Stack, and Address Space. | • All threads share the process control block in the same process, but the thread control blocks, stack, and address space are all unique. |
| • When one process is blocked, it has no effect on the other processes that are still operating. | • When one thread in a process is blocked, all other threads' processes are also blocked. |
| • Process switching is done using the operating system interface. | • Thread switching does not require a kernel interrupt or the use of an operating system. |

## 6. Explain Database Management System? What are the benefits of this approach over traditional file systems?

A database management system (or DBMS) is essentially just a computerized data storage system. Users of the system are given the ability to execute a variety of operations on it, including data manipulation and database structure maintenance.

The advantages of a database management system over traditional file systems are as follows:

- **Better Data Management**

Users can access better-managed data thanks to database management. As a result, end-users will be able to immediately review their information and respond to any changes.

- **Data Security**

As the number of users grows, so does the rate at which data is moved or shared, increasing the risk of data security. It's commonly utilized in the business world, where companies invest a lot of money, time, and effort to ensure data protection and proper use. By providing a stronger platform for data privacy and security standards, a Database Management System (DBMS) assists businesses in improving data security.

- **Reduced Data Inconsistency**

When different versions of the same data appear in different places in a database management system, data inconsistency is minimized. When a student's name is saved as "William Shakespeare" on the school's main computer, but as "W. Shakespeare" on the teacher's registered system, data discrepancy occurs.

- **Faster data access**

A database management system (DBMS) helps to produce quick responses to database queries, allowing for faster and more accurate data access. End users, for example, will have better data access when working with large amounts of sales data, allowing for a faster sales cycle.

## 7. What do you know about transaction ACID properties in relation to Database Management Systems?

Every transaction in a SQL Database must adhere to a set of guidelines. ACID attributes relate to these characteristics.

These are defined as follows:

- **A stands for Atomicity.**

This signifies that the entire transaction occurs at once or not at all. There is no intermediate ground, which means that there are no steps to transactions. Each transaction is viewed as a single entity that is either completed or not. It includes the following two steps.

## Abort:

Any database updates are lost if a transaction aborts.

## Commit:

When a transaction commits, the changes contained within it become visible.

Atomicity is often known as the "all or nothing rule."

Consider transferring money from one bank account to another. Either the transaction must be finished completely, or it will fail. A halfway transaction, such as money being deducted from one account but not credited to the other, is not possible.

- **C for Consistency.**

To ensure that the database is consistent, integrity constraints must be satisfied both before and after the transaction. It relates to the accuracy of a database.

For example, if a bank transaction is made from account A with X money to account B with Y money, the total amount of money must remain the same, i.e., X + Y.

- **I for Isolation.**

This property ensures that several transactions can run concurrently without producing database state problems. Non-interfering transactions are carried out. Changes performed in one transaction aren't visible to other transactions until the change for that transaction is put to memory or committed. This feature ensures that concurrently running transactions provide the same state as if they were run sequentially in some order.

A bank, for example, may have numerous ATMs throughout a country. The ATMs can all be used at the same time. They act as if they are the only transaction taking place on the bank's database, thus isolating them.

- **D for Durability.**

This property ensures that once a transaction has completed execution, the database updates and modifications are saved and written to memory and that they survive system failure. These changes are now permanently stored in non-volatile memory. As a result, the results of the transaction are never lost.

For example, a backup database must be kept so that we can recover data from the backup database if the primary database fails.

## 8. In the C programming language context, explain the differences between struct and union.

### struct

A structure is a user-defined data type in C that allows you to mix data items of different types. A structure is used to represent a record.

### Syntax

struct structureName

{

member definition;

member definition;

...

member definition;

};

### Example

struct student

{

int id;

char name[50];

string branch;

};

## union

A union is a special data type in C that allows you to store many data types in the same memory space. Although a union can have many members, only one of them has worth at any given time. Unions are a good way to reuse the same memory space for several tasks.

## Syntax

union unionName

{

member definition;

member definition;

...

member definition;

};

## Example

union student

{

string name;

string branch;

int phone;

};

The differences between a struct and a union are summarized below:

| **struct** | **union** |
| --- | --- |
| • The struct keyword is used to define a | • The keyword union is used to define |

| | |
|---|---|
| structure. | union. |
| • When variables are declared in a structure, the compiler allocates memory to each variable member. | • The size of a structure is determined by the total size of each data element. |
| • The compiler allocates memory to the variable member with the biggest size when a variable is declared in a union. | • The largest data member in a union determines the size of the union. |
| • Changing the value of one variable has no effect on the other variables in the struct. | • Changing the value of one variable member will have an effect on the union's other variables. |
| • Each member of a variable has its own memory space. | • The memory space of members of a variable is shared by the variable with the largest size. |
| • A structure's multiple variables can be initialized at the same time. | • A union's first data member can only be initialized. |
| • All variable members store some value at any point in the program. | • Only one data member stores a value at any one time in the program. |
| • It's used to keep track of the values of various data types. | • It's used to save a single value from a variety of data formats at a time. |
| • Any data member can be accessed and retrieved at the same time. | • Individual data members can be accessed and retrieved at any time. |

## 9. In every OOPs programming language, distinguish between variable/function declaration and definition.

The aim of a variable declaration is to inform the compiler of the variable's name, the kind of value it contains, and, if applicable, the starting value. In other words, a declaration offers information about a variable's attributes. A variable's definition allocates memory space for the variable and defines where it will be kept.

The differences between definition and declaration are illustrated in the table below:

| **Definition** | **Declaration** |
|---|---|
| A variable or function can only be defined once. | A variable or function can be declared an unlimited number of times. |

| | |
|---|---|
| During the defining process, memory is allocated. | During declaration, no memory is allocated. |
| Example<br>void my_func()<br>{<br>  cout << “something printed” << “\n”;<br>}<br>The compiler allocates memory for the void function "my_func" defined in the preceding code as soon as it detects it. | Example<br>void my_func();<br>The code above declares the void function "my_func." |

## 10. What do you mean by arrays? What are some examples of array applications in real life?

An array is a collection of things that are stored in contiguous memory locations. The goal is to put things of the same type together. By simply adding an offset to a base value, such as the memory location of the array's first element, it is simple to calculate the position of each element.

The array's real-world applications are as follows:

- As a simple application, arrays can be used to store data in a tabular format. If we wish to save our contacts on our phones, for example, the program will simply generate an array containing all of our contacts.
- The leaderboard of a game can be easily arranged by recording the score in arrays and sorting them in descending order to view each player's position in the game.
- A simple question paper is made up of a series of numbered questions, each with its own set of marks.
- 2D arrays, often known as matrices, are employed in image processing.
- It's also utilized in speech recognition, where an array represents each spoken signal.

## 11. In the context of operating systems, how do you define a deadlock? What are the prerequisites for a deadlock?

Consider two trains approaching one other on the same track with just one track: once they are in front of each other, neither train can move. A similar situation occurs in operating systems when two or more processes share resources while waiting for resources held by other processes. Assume two trucks are attempting to cross a one-way bridge from opposite ends. None of the trucks are ready to return, and none of them can pass the bridge. A deadlock has been reached in this case.

The following are the prerequisites for a deadlock:

### Mutual exclusion

Occurs when one or more resources are not shared. That is, the resource can only be used one process at a time.

### Hold and Wait

A process is defined as retaining at least one resource while waiting for further resources.

### No Preemption

A resource can only be gotten from a process if the process releases it. There can't be any forced resource snatching.

### Circular Wait

A collection of processes is waiting in a cyclic pattern for each other.

## 12. In sorting algorithms, distinguish between merge sort and quick sort.

The distinctions between rapid sort and merge are as follows:

| Quick Sort | Merge Sort |
| --- | --- |
| • The array is partitioned into any ratio in a quick sort. There is no need to partition the array of components into equal pieces while doing a quick sort. | • The merge sort divides the array into only two halves (i.e., n/2). |
| • Quick sort's worst-case complexity is O(n^2). | • The worst and average instances in merge sort have the same complexity O |

| | |
|---|---|
| | (n log n). |
| • Large datasets, on the other hand, do not work well with quick sort. | • Merge sorting can be used for any type of data set, no matter how huge (either large or small). |
| • The quick sort is in place because it requires no additional storage. | • Because the auxiliary arrays demand additional memory space, merge sort is not implemented. |
| • In this case, quick sort is unreliable (two elements with the same value may appear in the sorted array in a different order than in the unsorted input array). It could, however, be made stable with a few code changes. | • Because two elements with the same value appear in the sorted output array in the same order as they did in the unsorted input array, merge sort is stable. |
| • For arrays, quick sort is desirable. | • For linked lists, merge sort is desirable. |
| • Quicksort is faster than merge sort because of its cache locality (in many cases, like in a virtual memory environment). | • Merge sort has a bad reference locality. |

## 13. In the context of OOPs programming, what do you mean by an entry controlled loop?

If the condition or expression at the point of entry becomes true, control is passed to the body of the loop. This type of loop is known as an "entry control loop" since it governs loop entry.

Entry controlled loops include while loops and for loops.

## 14. What are your thoughts on procedural programming? What is the difference between it and object-oriented programming?

The concept of invoking processes is at the heart of procedural programming, which emerged from structured programming. Procedures, often known as routines, subroutines, or functions, are a collection of instructions that must be followed. Any procedure in a program can be called by other procedures or the program itself at any point during execution.

The contrasts between procedural and object-oriented programming are illustrated in the table below:

| Procedural Programming | Object-Oriented Programming |
| --- | --- |
| • The program is divided down into little modules called functions in procedural programming. | • A program is divided into separate components called objects in object-oriented programming. |
| • Procedural programming employs the top-down approach. | • In object-oriented programming, the bottom-up method is used. |
| • Adding new data and functions is a difficult task. | • Adding extra data and functions is simple. |
| • Procedural programming does not allow for overloading. | • Overloading is feasible in object-oriented programming. |
| • Procedural programming is insecure because it lacks a proper way to hide data. | • Object-oriented programming hides data, making it safer. |
| • The function takes precedence over data in procedural programming. | • In object-oriented programming, data takes precedence over function. |
| • Examples include C, FORTRAN, Pascal, Basic, and other programming languages. | • Examples include C++, Java, Python, C#, and other programming languages. |

## 15. A sorted array of 0s and 1s is provided. The purpose is to find the index of the first '1' in the sorted array. It's possible that the array is entirely made up of 0s or 1s. If the array contains no 1s, print "-1."

### Method

The array is assumed to be sorted. We make use of this array attribute and utilize binary search to locate the first occurrence of 1 in the given array. We begin by looking through the entire array for the center element. When the middle element is 0, it means that our response is on the right side of the middle element. If the middle element is 1, the answer can be this index or the indices left to the current middle if there are more 1s before the current middle.

### Example

### Input

arr = {0, 0, 0, 0}

## Output

- 1

## Input

arr = {0, 0, 0, 0, 1, 1, 1}

## Output

4

## Code

```
#include <bits/stdc++.h>
using namespace std;
int searchIndex(int a[], int left, int right)
{
while (left <= right) {
int mid = left + (right - left) / 2;
if (a[mid] == 1 && (mid == 0 || a[mid - 1] == 0))
return mid;
else if (a[mid] == 1)
left = mid - 1;
else
left = mid + 1;
}
return -1;
}
int main()
{
int a[] = {0, 0, 0, 0};
int n = sizeof(a) / sizeof(a[0]);
```

```
cout << searchIndex(a, 0, n - 1);
return 0;
}
```

### Output

- 1

### Method Explanation

We developed a function called 'searchIndex' in the preceding code to find the first occurrence of the first one in the sorted array. We minimize the search space by adjusting the left and right values to match the middle element's current value. If left is greater than right, we return -1, indicating that there is no 1 in the array.

## 16. Write a program that converts the characters in a string to the opposite case, i.e., converts lowercase characters to uppercase and vice versa.

### Method

We go over the string character by character. We remove 32 from the current character and transform it to uppercase if it is in lowercase. In the same way, if the current character is uppercase, we add 32 and change it to lowercase.

### Example

### Input

“siMPlilEArn”

### Output

“SImpLILeaRN”

### Input

“Ibm”

### Output

“IBM”

## Code

```
#include <iostream>
using namespace std;
void caseChange(string& str)
{
int len = str.length();
for (int i = 0; i < len; i++) {
if (str[i] >= 'a' && str[i] <= 'z')
{
str[i] = str[i] - 32;
}
else if (str[i] >= 'A' && str[i] <= 'Z')
{
str[i] = str[i] + 32;
}
}
}
int main()
{
string str = "siMPlilEArn";
cout << "The Original String is : " << str << "\n";
caseChange(str);
cout << "The Changed String is : " << str << "\n";
return 0;
}
```

## Output

The Original String is : siMPlilEArn

The Changed String is : SImpLILeaRN

## Method Explanation

We define a function named 'changeCase' in the preceding code that takes a string reference as a parameter. We iterate through the string, changing each character's case to the opposing case by adding or removing 32.

## 17. Write a program to figure out how many different ways we can produce a change for N cents, assuming we had an infinite supply of each of the C = C1, C2,...Cm coins. The order in which the coins are placed makes no effect.

## Method

To count the total number of solutions, we can divide all set solutions into two sets.

1) Solutions that do not include the mth coin (or Cm).

2) The solution contains at least one Cm.

If the function solve(C[], m, n) is used to count the number of solutions, it may be written as the sum of solve(C[], m-1, n) and solve(C[], m, n) (C[], m, n-Cm). We'll utilize dynamic programming to save the result for a specific n and m value. This reduces our temporal complexity to O(nm).

## Example

## Input

N = 5, C = {11, 23, 43, 7, 12}

## Output

0

## Input

N = 4, C = { 1, 2, 3}

## Output

4

## Code

```
#include<bits/stdc++.h>
using namespace std;
int solve(int C[], int m, int n)
{
int dp[n + 1][m];
for (int i = 0; i < m; i++)
dp[0][i] = 1;
for (int i = 1; i < n + 1; i++)
{
for (int j = 0; j < m; j++)
{
int x = (i-C[j] >= 0) ? dp[i - C[j]][j] : 0;
int y = (j >= 1) ? dp[i][j - 1] : 0;
dp[i][j] = x + y;
}
}
return dp[n][m - 1];
}
int main()
{
int C[] = {11, 23, 43, 7, 12};
int m = sizeof(C)/sizeof(C[0]);
```

```
int n = 5;
cout << solve(C, m, n);
return 0;
}
```

## Output

0

## Method Explanation

We define a method called 'solve' in the preceding code that returns the number of ways that n may be expressed from a collection of m coins with different values. We make a dp table with the dimensions (n+1) * m. The number of ways that I can be represented by a set of j coins is expressed as dp[i][j]. The recurrence formula dp[i][j] = x + y has been employed here. y = dp[i][j-1], x = dp[i - C[j]][j].

## 18. In the context of the UNIX operating system, what is the purpose of the sudo command?

sudo is defined as Super Users DO. The sudo command is often used in Linux as a prefix to a command that only superusers can run. If you run a command with the prefix "sudo" before it, it will run with elevated privileges, allowing a user with the proper permissions to run a command as another user, such as the superuser. This is the Windows equivalent of "run as administrator."

Each user who may use the sudo command must have an entry in the sudoers file, which is located at "/etc/sudoers." To change or inspect the sudoers file, remember to use the sudo command. For editing the sudoers file, use the "visudo" command.

## 19. What are your thoughts on virtual memory in terms of operating systems?

A storage allocation method that lets you address secondary memory as if it were the main memory is called a Virtual Memory. Program-generated addresses are automatically converted to machine addresses, which are different from the addresses used by the memory system to designate physical storage places.

The quantity of secondary memory available is defined by the number of main storage sites available rather than the actual number of main storage locations, and the capacity of virtual storage is restricted by the computer system's addressing scheme.

## 20. In terms of operating systems, what are the three categories of schedulers? Explain.

Specialized computer programs that control the scheduling of processes in various ways are known as Schedulers. Their main task is to decide which jobs to enter into the system and which processes to conduct.

The three sorts of timetables are as follows:

### Long term Scheduler (LTS)

A task scheduler is another name for a long-term scheduler. The applications admitted for processing into the system are determined by a long-term scheduler. It selects processes from the ready queue and loads them into memory before executing them. The process is loaded into memory for CPU scheduling. The primary purpose of the job scheduler is to deliver a balanced mix of operations, including I/O bound and CPU bound workloads. It also controls the amount of multiprogramming done. The average rate of process formation must be equal to the average rate of process departure from the system if the degree of multiprogramming remains constant.

### Medium Term Schedulers (MTS)

These are used to swap processes in the main memory. It frees up the RAM that the processes have used up. As a result, the degree of multiprogramming is minimized. The medium-term scheduler is in charge of the swapped-out processes.

### Short Term Scheduler (STS)

A CPU scheduler is another name for it. Its main purpose is to increase system performance based on a set of established criteria. It is the shift from the ready to the running stage of a process. The CPU scheduler selects a process from among those that are ready to run and gives it CPU time. Short-term schedulers, often known as dispatchers, choose the next process to run