**What is Competitive Programming?**

Competitive programming is a sport where contestants solve algorithmic problems within a time limit using a programming language of their choice.

1. *Problem-solving*
2. *Data structures and algorithm*
3. *Fundamentals of programming*

Pre-Requisites of Competitive Programming…

* *Programming Language Proficiency*
* *Basic Data Structures*
* *Algorithms*
* *Mathematics*
* *Problem-Solving Skills*
* *Debugging Skills*
* *Efficient Coding Techniques*
* *Online Judges and Platforms*
* *Time Management*
* *Practice and Consistency*
* *Community Involvement*
* *Competitive Spirit*

Basic terminologies in competitive programming:

* *Time limit exceed*
* *Memory limit exceed*
* *Runtime error*
* *Compilation error*
* *Optimization*
* *Template*
* *Editorials*
* *Testing system*

**Endl vs Back n \*important**

Endl not only inserts a newline character but also flushes the output buffer. Whereas in case of \n used to insert a line break but it does not flush the output buffer.

Endl ; = ‘\n’ << flush;

Difference is seen in case of bulky programs.

C++ standard streams (cin, cout, cerr)

C standard streams (stdin, stdout, stderr).

#include<bits/stdc++.h>

#include< iostream >

using namespace std;

time\_t start , ending;

int main()

{

auto start = chrono:: high\_resolution\_clock::now();

//code

auto ending = chrono:: high\_resolution\_clock::now();

time\_taken \* = 1e-9 ;

cout << time\_taken << endl;

return 0;

}

NOTE:

int main()

{

std::ios\_base::sync\_with\_stdio(false);

std::cin.tie(NULL);

// code for above general structure

}

is used in C++ programming with the Standard Library's input and output streams to control the synchronization between C++ standard streams (cin, cout, cerr) and C standard streams (stdin, stdout, stderr).

#include <iostream>

int main() {

    std::ios\_base::sync\_with\_stdio(false); // Turn off synchronization

    // Now you can use C++ and C-style I/O operations without synchronization issues

    std::cout << "Hello, ";

    printf("world!\n");

return 0;

}

POINTERS

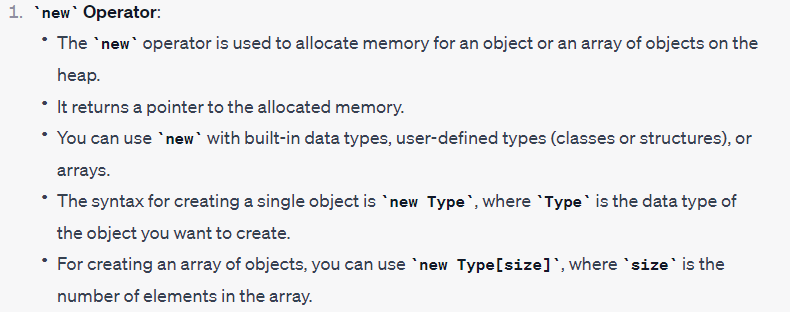
Smart pointer

* Unique pointer
* Shared Pointer



* *CList (common platform)*
* *Codechef*
* *HackerEarth*
* *Leetcode*
* *IOI*
* *ICPC*

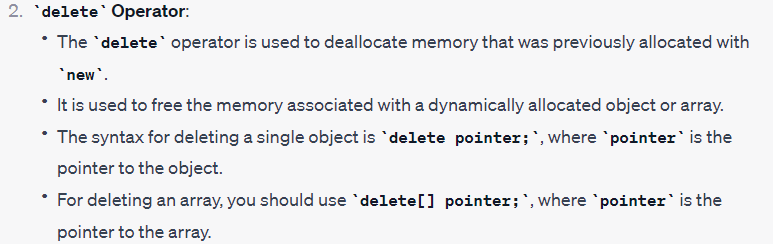
In C++, new and delete are operators used for dynamic memory allocation and deallocation. They are primarily used for allocating memory for objects on the heap, allowing you to create objects whose lifetimes are not limited to the scope of the current function or block.

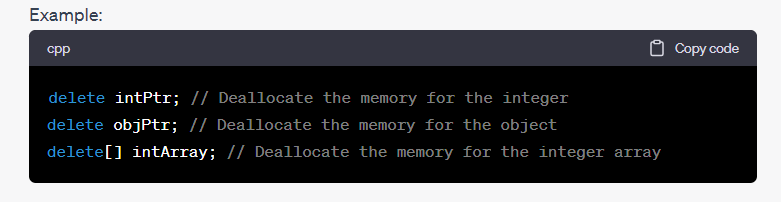


int \*intPtr = new int; // Allocate memory for a single integer

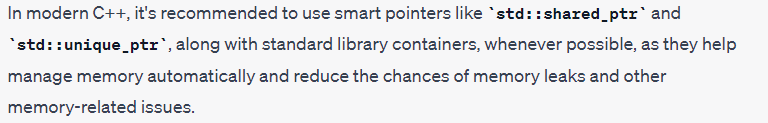
MyClass \*objPtr = new MyClass; // Allocate memory for an object of MyClass

int \*intArray = new int[10]; // Allocate memory for an integer array of size 10





It's important to note that when using dynamic memory allocation, you are responsible for releasing the allocated memory using delete (or delete[] for arrays) to avoid memory leaks. Failure to do so can lead to inefficient memory usage and potentially cause your program to run out of memory over time.



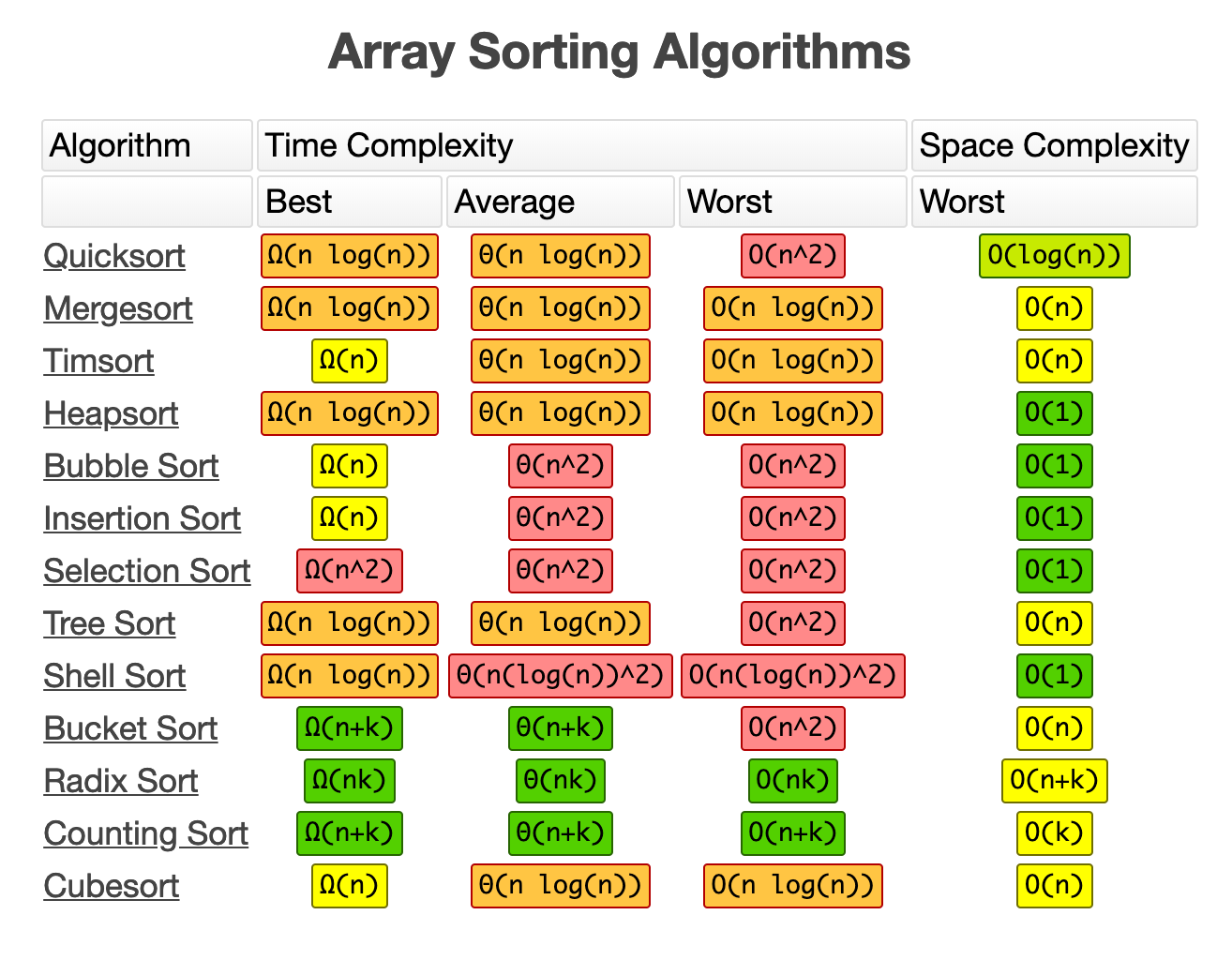
**Arrow operator**

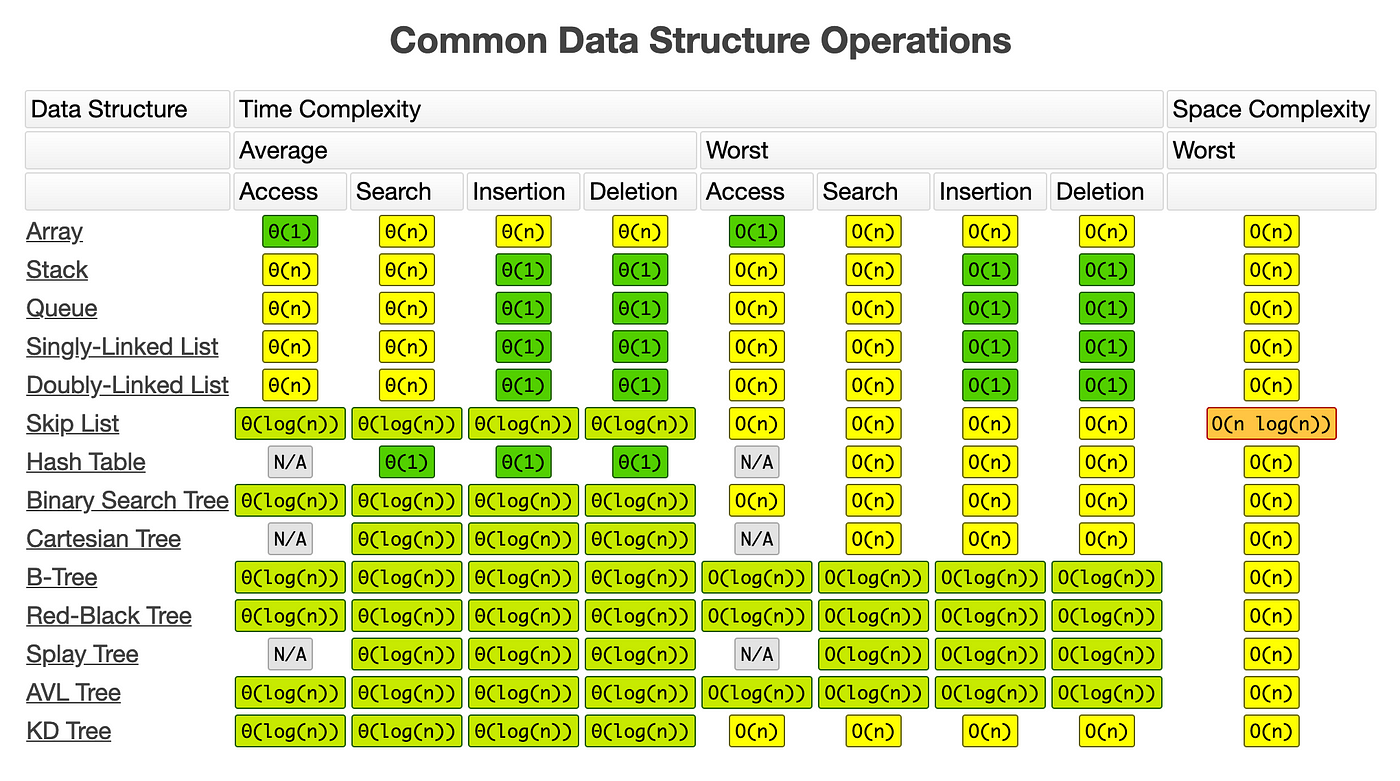
the arrow operator (->) is used to access members of an object that is pointed to by a pointer.

**Decreasing rate of growth…**

n! > 2n > n3 > n2 > nlog(n) > log(n!) > n > 2logn > log2n > log(n) > √log(n) > log(logn) > 1

NOTE: To access any element in array constant time is needed because other elements scanning is not required.



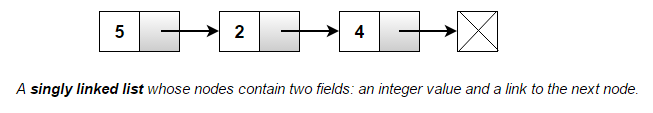


BASICS OF DATA STRUCTURES…

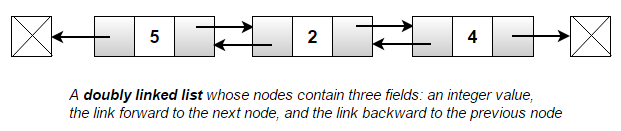
{LINKED LIST}

* *Singly*
* *Doubly*
* *Circular*

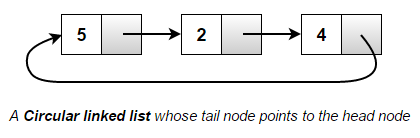
Singly Linked list



Doubly Linked list



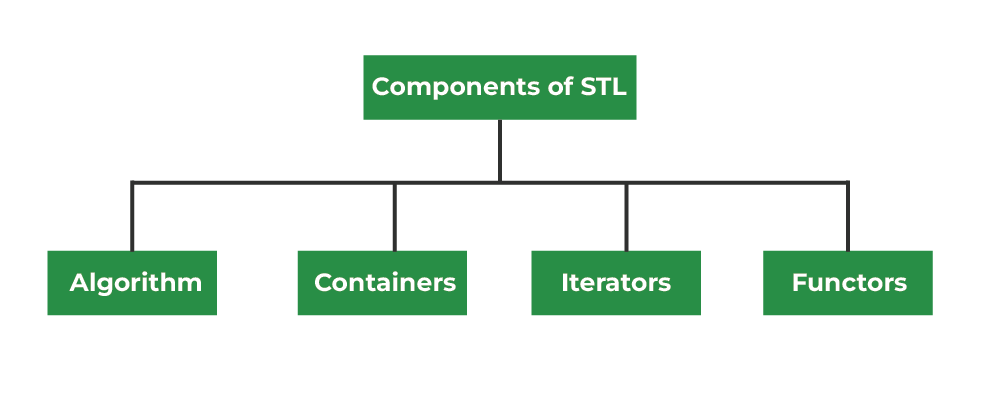
Circular Linked list

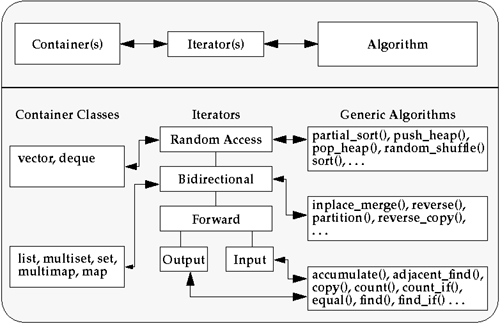


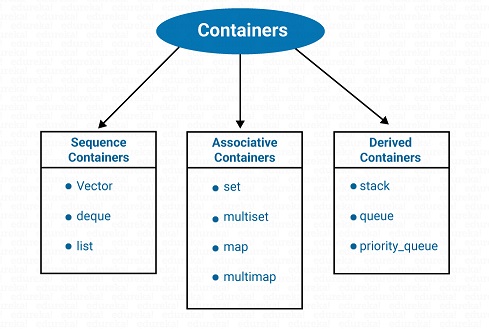
Q1.WAP to insert an element at the end of the Linked List!

#include <bits/stdc++.h>

**Standard Template Library (20.09.2004)**







**Standard array (#include <array>)**

#include <iostream>

#include <array>

int main() {

// Define an array of integers

std::array<int, 5> numbers = {10, 20, 30, 40, 50};

// Calculate the sum of the elements

int sum\_of\_numbers = 0;

for (int num : numbers) {

sum\_of\_numbers += num;

}

// Calculate the average

double average = static\_cast<double>(sum\_of\_numbers) / numbers.size();

// Display the array, sum, and average

std::cout << "Array: [";

for (int i = 0; i < numbers.size(); ++i) {

std::cout << numbers[i];

if (i < numbers.size() - 1) {

std::cout << ", ";

}

}

std::cout << "]" << std::endl;

std::cout << "Sum of numbers: " << sum\_of\_numbers << std::endl;

std::cout << "Average: " << average << std::endl;

return 0;

}

|  |
| --- |
| **STL Array**  Array<int,4> my\_array;  #include<array>  Void fun (int \*arr, int n)  Arr.at(0) = arr[0]  My\_array.front()  My\_array.back()  Void fun(my\_array)  my\_array.fill(0)  my\_array.sort()  Or  sort(my\_array.begin(),my\_array.end())  arr.swap(my\_array)  arr.size() |

|  |
| --- |
| **STL Vector**  vector<int> my\_vector;  #include<vector>  Vector = Dynamic array  V1.push\_back(-1);  Vector <int> V1 (10,0)  //size is ten all keys have value zero  #include<algorithm>  Reverse(myVector.begin(),myVector.end())  V1.size(); |

|  |
| --- |
| **STL Set**  **Properties of STL Set…**   * Associative: Each element in the set is key of itself. Element identified by value not by position. * Ordered: Element must follow a strict order at all times. All inserted elements also follow this. * Set: Value of an element is also its key, which is used to identify it. * Unique Keys: No two element have same values (sames a saying , no same keys). * Dynamic memory: Allocates memory dynamically accordingly to its need.   .insert()  .erase() //constant with pointer, logarithmic with values, linear with range.  .clear()  .size() vs max\_size()  .find() vs .count()  .empty() |

|  |
| --- |
| **STL Multiset**  Same as that of set with following exceptions  Duplicate keys or Duplicate elements  Ordered either (Asc. or Desc.)  Basic functions  .empty()  .size()  .max\_size()  .clear()  .insert(k)  .erase(k)  .count(k)  .find(k)  .lower\_bound(k)  .upper\_bound(k)  In set  2,3,1,3 -> 1,2,3  In Multiset  2,3,1,3 -> 1,2,3,3  DUPLICATES ARE ALLOWED IN MULTISET |

|  |
| --- |
| **STL Map**   * Key, value pairs. * Key must be unique. * Key are sorted in BST like fashion, with each key pointing to/ referring to its corresponding value. * Keys are stored in sorted fashion.   DIFFERENCE BETWEEN  (2D array dictionary Vs Map dictionary)  Operations  -access any key value pair by my\_map[key]  -create new key, value pair by my\_map[new\_key] = new value  -Updating a key value pair value |

|  |
| --- |
| **STL Multimap**  #include<map>  Keys can be DUPLICATE  Key value pair  Keys are still stored in BST like fashion  .emplace ---simple insertion  lower\_bound and upper\_bound  .count(k)  .find(k)  .extract(k) with insert (move(nh))  Use of MULTIMAP   * Fractional knapsack problem |

|  |
| --- |
| **STL Pair**  Pair <T1,T2>  Stores two heterogeneous objects in a tuple-like structure  It is simple tuple of two values |

|  |
| --- |
| **STL Array**  Incomplete |

|  |
| --- |
| **STL Vector1**  #include <bits/stdc++.h>  #include <vector>  using namespace std;  int main()  {      vector<int> my\_vector;      my\_vector.push\_back(-10);      my\_vector.push\_back(2);      my\_vector.push\_back(-1);      my\_vector.push\_back(-5);      sort(my\_vector.begin(),my\_vector.end());      for(int i=0; i< my\_vector.size(); i++)      {          cout << my\_vector[i] << " ";      }      return 0;  } |

|  |
| --- |
| **STL Vector2**  #include <bits/stdc++.h>  #include <vector>  using namespace std;  int main()  {      vector <int> my\_vector {1,3,2,4,5,6,7,8,9,10};  //no need to put = sign.      for(int i=0; i < my\_vector.size() ; i++)      {          cout << my\_vector[i] << " ";      }      cout << endl;      my\_vector.clear();  // this command will clear the above vector.      my\_vector.insert(my\_vector.begin()+1,999);      for(int i=0; i < my\_vector.size();i++)      {          cout << my\_vector[i] << " ";      }      cout << endl;  } |

|  |
| --- |
| **STL Set1**  #include <bits/stdc++.h>  #include <set>  using namespace std;  int main()  {      set <int,less<int>> set1 ={2,1,5,4,3,1};      set <int,greater<int>> set2= {2,1,5,4,3,1};      for(auto it=set1.begin(); it!=set1.end();++it)      {          cout << \*it << " ";      }      cout << endl;      for(auto it=set2.begin(); it!=set2.end(); ++it)      {          cout << \*it << " ";      }      cout << endl;  }  /\*      set1.end();  \*/ |

|  |
| --- |
| **STL Set2** |

|  |
| --- |
| **STL Set3** |

|  |
| --- |
| **STL Multiset1**  #include <bits/stdc++.h>  #include <set>  using namespace std;  int main()  {    multiset <int, greater<>> m\_set {9,6,5,7,8,3,2,1,0,2,8,-4};    //by default ascending order no need to use less<>      cout << "display the multiset" << endl;      for(auto it = m\_set.begin(); it!=m\_set.end(); ++it)      {          cout << \*it << " ";      }      cout << endl;  } |

|  |
| --- |
| **STL Map1**  #include <bits/stdc++.h>  #include <map>  using namespace std ;  int main()  {      map<int,int> my\_map ;      my\_map[0]=4;      my\_map[1]=9;      my\_map[2]=-3;      my\_map[3]=2;      my\_map[4]=0;      my\_map[5]=1;      for(auto it=my\_map.begin(); it!=my\_map.end(); ++it)      {          cout << it->first << " " << it->second << endl;      }      cout << endl;  } |

|  |
| --- |
| **STL Multimap1**  #include<bits/stdc++.h>  #include<map>  using namespace std;  int main()  {    multimap <int,int,greater<>> my\_multimap;      my\_multimap.emplace(0,1);      my\_multimap.emplace(1,2);      my\_multimap.emplace(3,0);      my\_multimap.emplace(1,0);      my\_multimap.emplace(1,0);      for(auto it=my\_multimap.begin(); it!=my\_multimap.end(); ++it)      {          cout << it->first << " " << it->second << endl;      }      multimap <int,int> Knapsack;      Knapsack.emplace(100,80);      Knapsack.emplace(90,70);      Knapsack.emplace(50,10);      for(auto it=Knapsack.begin(); it!=Knapsack.end(); ++it)      {          cout <<  it->first << " " << it->second << endl;      }      cout << my\_multimap.count(1);      //my\_multimap.find(2);  } |

**Pre-requisite / Skillset for C++ Programming Internship**

1. Intern must have basic knowledge on Object Oriented Programming concepts using the C++ language.

2. The principles of data abstraction, inheritance and polymorphism should be known.

3. Should be aware of virtual functions and polymorphism.

4. Adequate knowledge on formatted I/O and unformatted I/O is needed.

5. The concept of exception handling should be known.

6. Should be capable of describing the procedural and object oriented paradigm with concepts of streams, classes, functions, data and objects.

7. Should have knowledge on dynamic memory management techniques using pointers, constructors, destructors, etc.

8. Should have adequate knowledge on concepts like function overloading, operator overloading, virtual functions and polymorphism.

9. Ability to classify inheritance with the understanding of early and late binding, usage of exception handling, generic programming.

10. Should be able to demonstrate the use of various OOPs concepts with the help of programs.

|  |
| --- |
| // 1. Creating list node on case of singly linked list  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  #include <iostream>  // Define a struct for the node  struct Node {      int data;      Node\* next;  };  int main() {      // Creating nodes      Node\* node1 = new Node;      Node\* node2 = new Node;      Node\* node3 = new Node;      // Assigning data values      node1->data = 10;      node2->data = 20;      node3->data = 30;      // Linking nodes to create a simple linked list      node1->next = node2;      node2->next = node3;      node3->next = nullptr; // Mark the end of the list      // Accessing data in the nodes      std::cout << "Data in node1: " << node1->data << std::endl;      std::cout << "Data in node2: " << node2->data << std::endl;      std::cout << "Data in node3: " << node3->data << std::endl;      // Deallocate memory (important to prevent memory leaks)      delete node1;      delete node2;      delete node3;      return 0;} |
| // 2. Creating list node in case of Doubly linked list  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  #include <iostream>  struct Node {      int data;      Node\* prev;      Node\* next;  };  int main() {      // Creating nodes      Node\* node1 = new Node;      Node\* node2 = new Node;      Node\* node3 = new Node;      // Assigning data values      node1->data = 10;      node2->data = 20;      node3->data = 30;      // Linking nodes to create a doubly linked list      node1->prev = nullptr;      node1->next = node2;      node2->prev = node1;      node2->next = node3;      node3->prev = node2;      node3->next = nullptr;      // Accessing data in the nodes      std::cout << "Data in node1: " << node1->data << std::endl;      std::cout << "Data in node2: " << node2->data << std::endl;      std::cout << "Data in node3: " << node3->data << std::endl;      // Deallocate memory (important to prevent memory leaks)      delete node1;      delete node2;      delete node3;      return 0;  } |

|  |
| --- |
| // 3. Traversal in Singly linked list  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  #include <iostream>  struct Node {      int data;      Node\* next;  };  void traverseLinkedList(Node\* head) {      Node\* current = head;  // Start from the head of the list      while (current != nullptr) {          // Process the current node (e.g., print its data)          std::cout << current->data << " ";          // Move to the next node          current = current->next;      }      std::cout << std::endl;  }  int main() {      // Create a linked list with three nodes      Node\* node1 = new Node;      Node\* node2 = new Node;      Node\* node3 = new Node;      node1->data = 10;      node2->data = 20;      node3->data = 30;      node1->next = node2;      node2->next = node3;      node3->next = nullptr;  // Mark the end of the list      // Traverse and print the linked list      traverseLinkedList(node1);      // Clean up (deallocate memory)      delete node1;      delete node2;      delete node3;      return 0;  } |

|  |
| --- |
| // 4. Create an empty linked list  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  #include <iostream>  // Define a structure for a node in the linked list  struct Node {      int data;      Node\* next;  };  // Function to create an empty linked list  Node\* CreateEmptyList() {      return nullptr; // Initialize the head pointer to nullptr  }  int main() {      // Create an empty linked list      Node\* myList = CreateEmptyList();      // Check if the list is empty      if (myList == nullptr) {          std::cout << "The linked list is empty." << std::endl;      } else {          std::cout << "The linked list is not empty." << std::endl;      }      return 0;  } |

|  |
| --- |
| // 5. Insertion at the end  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  #include <iostream>  // Define a structure for a node in the linked list  struct Node {      int data;      Node\* next;  };  // Function to insert a new node at the end of the linked list  Node\* InsertAtEnd(Node\* START, int info) {      Node\* New\_Node = new Node(); // Allocate memory for the new node      New\_Node->data = info;       // Set the data field of the new node      New\_Node->next = nullptr;    // Make it the last node by setting next to nullptr      if (START == nullptr) {          // If the list is empty, make the new node the START          START = New\_Node;      } else {          Node\* Temp = START;          while (Temp->next != nullptr) {              Temp = Temp->next;          }          Temp->next = New\_Node;      }      return START;  }  // Function to display the linked list  void DisplayList(Node\* START) {      Node\* Temp = START;      while (Temp != nullptr) {          std::cout << Temp->data << " ";          Temp = Temp->next;      }      std::cout << std::endl;  }  int main() {      Node\* START = nullptr; // Initialize an empty linked list      // Insert elements at the end      START = InsertAtEnd(START, 10);      START = InsertAtEnd(START, 20);      START = InsertAtEnd(START, 30);      // Display the linked list      std::cout << "Linked List: ";      DisplayList(START);      return 0;  } |

|  |
| --- |
| #include <iostream>  // Function to delete an element from an array using linear search  void deleteElement(int arr[], int &size, int elementToDelete) {      int found = -1; // Variable to keep track of whether the element was found or not      // Linear search to find the element to delete      for (int i = 0; i < size; i++) {          if (arr[i] == elementToDelete) {              found = i; // Element found at index i              break;          }      }      if (found != -1) {          // Shift elements to the left to overwrite the element to be deleted          for (int i = found; i < size - 1; i++) {              arr[i] = arr[i + 1];          }          size--; // Decrease the size of the array          std::cout << "Element " << elementToDelete << " deleted successfully." << std::endl;      } else {          std::cout << "Element " << elementToDelete << " not found in the array." << std::endl;      }  }  int main() {      const int maxSize = 100; // Maximum size of the array      int arr[maxSize];      int size;      std::cout << "Enter the size of the array: ";      std::cin >> size;      if (size <= 0 || size > maxSize) {          std::cout << "Invalid array size." << std::endl;          return 1; // Exit with an error code      }      std::cout << "Enter elements of the array:" << std::endl;      for (int i = 0; i < size; i++) {          std::cin >> arr[i];      }      int elementToDelete;      std::cout << "Enter the element you want to delete: ";      std::cin >> elementToDelete;      deleteElement(arr, size, elementToDelete);      // Print the updated array      std::cout << "Updated array: ";      for (int i = 0; i < size; i++) {          std::cout << arr[i] << " ";      }      std::cout << std::endl;      return 0;  } |

Floyd Cycle Detection Algorithm

Floyd's Cycle Detection Algorithm, also known as the "tortoise and hare" algorithm, is a technique used to detect cycles in linked lists or sequences of elements in a memory-efficient way. It's commonly used to determine if a linked list has a cycle. Here's how it works:

1. Start with two pointers, often called "tortoise" and "hare," both initially pointing to the head of the linked list.
2. Move the "tortoise" pointer one step at a time, and the "hare" pointer two steps at a time. This simulates a race between the two pointers, with the "hare" moving faster.
3. If there is no cycle in the linked list, the "hare" pointer will eventually reach the end of the list (i.e., it will become NULL), and you can conclude that there is no cycle.
4. However, if there is a cycle, the "hare" pointer will eventually catch up to the "tortoise" pointer inside the loop.
5. At this point, you have detected a cycle in the linked list. To find the starting point of the cycle (if needed), you can reset one of the pointers (e.g., "hare") to the head of the list and move both pointers one step at a time until they meet again. The meeting point will be the starting point of the cycle.

Here's a simplified pseudocode for Floyd's Cycle Detection Algorithm:

|  |
| --- |
| function hasCycle(head):      tortoise = head      hare = head      while hare is not null and hare.next is not null:          tortoise = tortoise.next       // Move one step          hare = hare.next.next         // Move two steps          if tortoise == hare:              // Cycle detected              return true      // No cycle found      return false |

This algorithm is efficient and runs in O(n) time, where n is the number of nodes in the linked list. It uses only two pointers and doesn't require additional memory to detect cycles. It's widely used in computer science and algorithms for various applications, such as cycle detection in linked lists and graph algorithms.

|  |
| --- |
| // Balanced Parentheses: Check if a given expression with parentheses is balanced.  #include <iostream>  #include <stack>  #include <string>  bool isBalanced(std::string expression) {      std::stack<char> stack;      for (char &c : expression) {          if (c == '(' || c == '[' || c == '{') {              stack.push(c);          } else if (c == ')' || c == ']' || c == '}') {              if (stack.empty()) return false;              char top = stack.top();              stack.pop();              if ((c == ')' && top != '(') || (c == ']' && top != '[') || (c == '}' && top != '{')) {                  return false;              }          }      }      return stack.empty();  }  int main() {      std::cout << isBalanced("{[()]}") << std::endl;  // true      std::cout << isBalanced("{[(])}") << std::endl;  // false      return 0;  } |

|  |
| --- |
| //Evaluate Postfix Expression: Evaluate a postfix expression using a stack.  #include <iostream>  #include <stack>  #include <string>  int evaluatePostfix(std::string expression) {      std::stack<int> stack;      for (char &c : expression) {          if (isdigit(c)) {              stack.push(c - '0');          } else {              int b = stack.top();              stack.pop();              int a = stack.top();              stack.pop();              switch (c) {                  case '+': stack.push(a + b); break;                  case '-': stack.push(a - b); break;                  case '\*': stack.push(a \* b); break;                  case '/': stack.push(a / b); break;              }          }      }      return stack.top();  }  int main() {      std::cout << evaluatePostfix("23\*5+") << std::endl;  // 11      return 0;  } |

|  |
| --- |
| //Next Greater Element: Find the next greater element for each element in an array.  #include <iostream>  #include <stack>  #include <vector>  std::vector<int> nextGreaterElement(std::vector<int>& nums) {      std::stack<int> stack;      std::vector<int> result(nums.size(), -1);      for (int i = 0; i < nums.size(); ++i) {          while (!stack.empty() && nums[i] > nums[stack.top()]) {              int idx = stack.top();              stack.pop();              result[idx] = nums[i];          }          stack.push(i);      }      return result;  }  int main() {      std::vector<int> nums = {4, 5, 2, 10, 8};      std::vector<int> result = nextGreaterElement(nums);      for (int val : result) {          std::cout << val << " ";      }      std::cout << std::endl;  // 5 10 10 -1 -1      return 0;  } |

|  |
| --- |
|  |

|  |
| --- |
| // Implementing a Queue Using Arrays:  #include <iostream>  const int MAX\_SIZE = 100;  class Queue {  private:      int arr[MAX\_SIZE];      int front, rear;  public:      Queue() {          front = rear = -1;      }      bool isEmpty() {          return front == -1;      }      bool isFull() {          return (rear + 1) % MAX\_SIZE == front;      }      void enqueue(int value) {          if (isFull()) {              std::cout << "Queue is full. Cannot enqueue " << value << std::endl;              return;          }          if (isEmpty()) {              front = rear = 0;          } else {              rear = (rear + 1) % MAX\_SIZE;          }          arr[rear] = value;      }      void dequeue() {          if (isEmpty()) {              std::cout << "Queue is empty. Cannot dequeue." << std::endl;              return;          }          if (front == rear) {              front = rear = -1;          } else {              front = (front + 1) % MAX\_SIZE;          }      }      int getFront() {          if (isEmpty()) {              std::cout << "Queue is empty." << std::endl;              return -1;          }          return arr[front];      }  };  int main() {      Queue q;      q.enqueue(1);      q.enqueue(2);      q.enqueue(3);      std::cout << "Front of the queue: " << q.getFront() << std::endl;      q.dequeue();      std::cout << "Front after dequeue: " << q.getFront() << std::endl;      return 0;  } |

|  |
| --- |
| // Implementing a Queue Using a Linked List:  #include <iostream>  class Node {  public:      int data;      Node\* next;      Node(int value) {          data = value;          next = nullptr;      }  };  class Queue {  private:      Node\* front;      Node\* rear;  public:      Queue() {          front = rear = nullptr;      }      bool isEmpty() {          return front == nullptr;      }      void enqueue(int value) {          Node\* newNode = new Node(value);          if (isEmpty()) {              front = rear = newNode;          } else {              rear->next = newNode;              rear = newNode;          }      }      void dequeue() {          if (isEmpty()) {              std::cout << "Queue is empty. Cannot dequeue." << std::endl;              return;          }          Node\* temp = front;          front = front->next;          delete temp;      }      int getFront() {          if (isEmpty()) {              std::cout << "Queue is empty." << std::endl;              return -1;          }          return front->data;      }  };  int main() {      Queue q;      q.enqueue(1);      q.enqueue(2);      q.enqueue(3);      std::cout << "Front of the queue: " << q.getFront() << std::endl;      q.dequeue();      std::cout << "Front after dequeue: " << q.getFront() << std::endl;      return 0;  } |

1 lifo fifo

2 enqueue(data) dequeue() peek() isEmpty() isFull() resize()

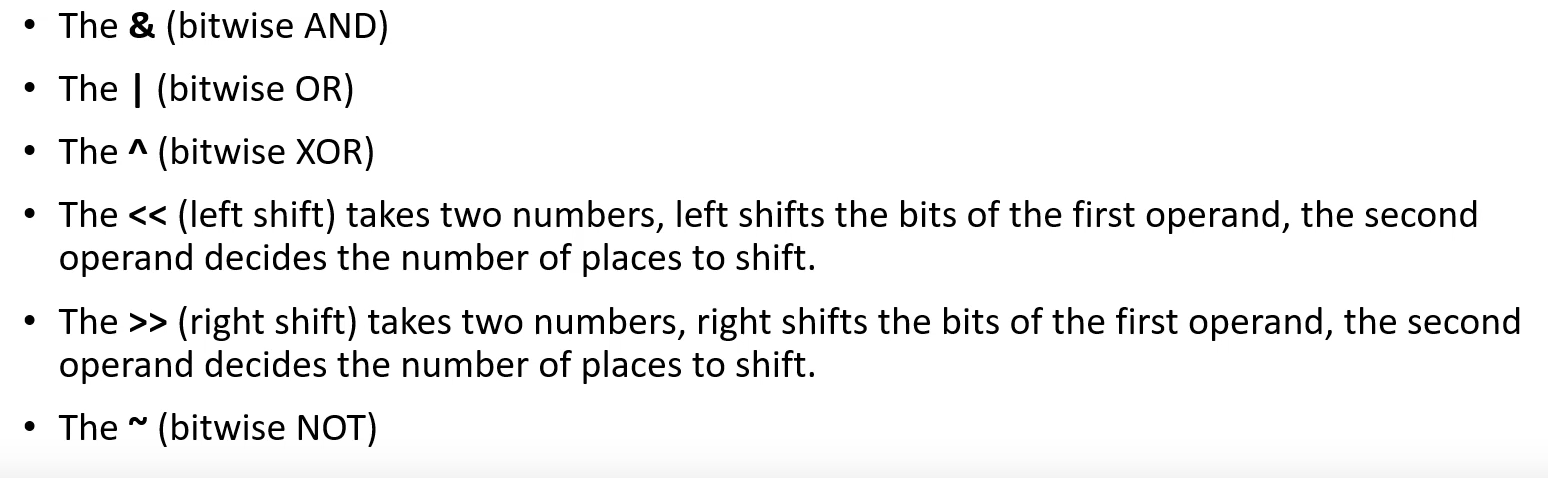
|  |
| --- |
| how can i be good at coding so that i can get placement of 40 lacs per annum in india and what things and skills i need to have to grab highest placement in my college  ChatGPT  Achieving a high-paying job in the field of software development, such as a 40 lakh per annum placement in India, requires a combination of technical skills, practical experience, and soft skills. Here's a roadmap to help you work towards your goal:   1. **Master Programming Languages and Data Structures**:    * Focus on learning and mastering popular programming languages such as Python, Java, C++, or JavaScript.    * Develop a strong understanding of data structures (arrays, linked lists, trees, graphs) and algorithms (sorting, searching, dynamic programming). 2. **Build Strong Problem-Solving Skills**:    * Practice solving algorithmic and coding challenges on platforms like LeetCode, HackerRank, and Codeforces.    * Participate in competitive programming to improve your problem-solving abilities. 3. **Learn Web and Mobile Development**:    * Familiarize yourself with web development technologies (HTML, CSS, JavaScript, front-end and back-end frameworks) and mobile app development (Android or iOS).    * Create personal projects and showcase them in your portfolio. 4. **Database and Backend Development**:    * Learn about databases, both SQL and NoSQL. Understand how to design and optimize database schemas.    * Explore back-end development using technologies like Node.js, Django, Flask, or Spring Boot. 5. **Data Science and Machine Learning** (Optional):    * If you're interested in data science or machine learning, acquire skills in data analysis, machine learning libraries (e.g., TensorFlow, PyTorch), and relevant tools. 6. **Version Control and Collaboration**:    * Learn Git for version control. Understand how to collaborate on projects using platforms like GitHub or GitLab. 7. **Software Development Tools and Practices**:    * Gain proficiency in software development practices such as code reviews, testing, and debugging.    * Familiarize yourself with DevOps tools and methodologies for continuous integration and deployment (CI/CD). 8. **Work on Real Projects**:    * Build a strong portfolio by working on personal or open-source projects. Practical experience is highly valued by employers. 9. **Internships and Co-op Programs**:    * Seek internships or co-op programs with reputable companies. Real-world experience is essential. 10. **Soft Skills and Communication**:     * Develop strong communication and interpersonal skills. Effective communication is vital in team environments.     * Practice problem-solving, critical thinking, and adaptability. 11. **Networking**:     * Attend tech meetups, conferences, and career fairs to network with professionals and recruiters.     * Join online developer communities and forums. 12. **Resume and Interview Preparation**:     * Create a well-structured resume highlighting your skills and projects.     * Prepare for technical interviews. Practice coding challenges and behavioral questions. 13. **Continual Learning**:     * Stay updated with the latest trends and technologies in the field. Technology evolves rapidly, so ongoing learning is essential. 14. **Apply for Top Companies**:     * Apply to reputable tech companies and startups that offer competitive salary packages. 15. **Negotiation Skills**:     * When you receive job offers, be prepared to negotiate your salary and benefits to ensure you receive a competitive package. 16. **Stay Persistent and Positive**:     * Landing a high-paying job can be competitive and challenging. Stay persistent, learn from failures, and maintain a positive attitude. 17. **Consider Higher Education**:     * Pursuing a master's degree in a relevant field (e.g., computer science, data science) can open up additional opportunities.   Remember that achieving a high-paying job in the tech industry takes time and dedication. It's essential to continuously improve your skills, build a strong portfolio, and network with professionals in the field. Be patient and adaptable in your career journey, and you'll be better positioned to secure a lucrative placement in your college. |

***Bits and Bitwise operators***

Bit manipulation is a fundamental technique in computer programming that involves manipulating individual bits in a binary representation of data. It's a powerful and efficient way to perform various operations, such as setting, clearing, toggling, and checking the state of specific bits within a binary number or bitset.

Bit manipulation is commonly used in various domains, including competitive programming, low-level system programming, embedded systems, and cryptography.

Bitwise operators…



In left << operator

Eg.

Int a=13;

Int b=a<<n;

It means b = a\* (2)n.

Int b=a>>n;

It means b = a/(2)n .

|  |
| --- |
| // Basic program related bitmanipulation.  #include <iostream>  int main() {      // Bitwise AND (&) example      int a = 5; // binary: 0101      int b = 3; // binary: 0011      int result\_and = a & b; // binary: 0001 (decimal: 1)      std::cout << "Bitwise AND: " << result\_and << std::endl;      // Bitwise OR (|) example      int result\_or = a | b; // binary: 0111 (decimal: 7)      std::cout << "Bitwise OR: " << result\_or << std::endl;      // Bitwise XOR (^) example      int result\_xor = a ^ b; // binary: 0110 (decimal: 6)      std::cout << "Bitwise XOR: " << result\_xor << std::endl;      // Bitwise NOT (~) example      int c = 7; // binary: 0111      int result\_not = ~c; // binary: 1000 (decimal: -8, due to two's complement)      std::cout << "Bitwise NOT: " << result\_not << std::endl;      // Left shift (<<) example      int num = 1; // binary: 0001      int result\_left\_shift = num << 3; // binary: 1000 (decimal: 8)      std::cout << "Left shift: " << result\_left\_shift << std::endl;      // Right shift (>>) example      int num2 = 16; // binary: 10000      int result\_right\_shift = num2 >> 2; // binary: 0010 (decimal: 2)      std::cout << "Right shift: " << result\_right\_shift << std::endl;      return 0;  } |

|  |
| --- |
| // Checking if a Number is Even or Odd using Bitwise AND:  #include <iostream>  int main() {      int num;      std::cout << "Enter an integer: ";      std::cin >> num;      if (num & 1) {          std::cout << num << " is odd." << std::endl;      } else {          std::cout << num << " is even." << std::endl;      }      return 0;  } |

|  |
| --- |
| // Swapping Two Numbers without a Temporary Variable using XOR:  #include <iostream>  int main() {      int a, b;      std::cout << "Enter two integers: ";      std::cin >> a >> b;      std::cout << "Before swapping: a = " << a << ", b = " << b << std::endl;      a = a ^ b;      b = a ^ b;      a = a ^ b;      std::cout << "After swapping: a = " << a << ", b = " << b << std::endl;      return 0;  } |

|  |
| --- |
| // Counting the Number of Set Bits (1s) in an Integer:  #include <iostream>  int countSetBits(int num) {      int count = 0;      while (num > 0) {          count += (num & 1);          num >>= 1;      }      return count;  }  int main() {      int num;      std::cout << "Enter an integer: ";      std::cin >> num;      int setBits = countSetBits(num);      std::cout << "Number of set bits in " << num << " is: " << setBits << std::endl;      return 0;  } |

Linear search

Binary search

**Non-linear data structure**

**A tree can be a graph but a graph can’t be a tree.**