LEARN DSA WITH C++

WEEK :: 07

LEARN DSA WITH C++

PDF

COURSE INSTRUCTOR BY
ROHIT NEGI

MADE BY-
PRADUM SINGHA

Check Profile

My profile

LEARN DSA WITH C++

WEEK :: 07 DAY: 01 DATE: 29-05-2023

OBJECT ORIENTED PROGRAMMING

OOPs meaning :: **whatever works around the object it is called oops.**
[whatever has worked just done without knowing the process (beck-end).]

OBJECTS :: **Instance of a Class.**

Exemple :: **Student, Camera, Mobile, TV, ...** **e.t.c**

How to use:-

Student property :: **Name, Roll No, Branch,**

```
Int main()
{
  int Roll No;
  String name, Branch;
  return o;
}
```

Class :: **It is a user defined data type.**
:: **BluePrint of an Object.**

Roll_No- Name- Branch -	Student

Create Class::

```
Class Student {
      int Roll-No;
      String Name;
      String Branch;
};

int main( )
{
   Student obj;
   obj.Roll_No=7;
   obj.Name = Pradum;
```

```
  obj.Branch = ECE;
return  0;
}
```

#Code

```
#include<iostream>
using namespace std;

class Student
{
    public:
    string name;
    int Roll_No;
    string Branch;
};
int main()
{
    Student object;
    object.name = "Pradum";
    object.Roll_No = 7;
    object.Branch = "ECE";
    cout<<object.name<<"  ";

return 0;
};
```

Access Specifier

Private :: **We can't access directly anywhere.**

Public :: **We can access directly anywhere.**

Public Access Specifier

```
#include<iostream>
using namespace std;

class Bank
{
    public:
    string name;
    int balance;

    void check_balance()
    {
        cout<<balance<<endl;
    };

    void withdraw()
    {
        balance-=100;
        cout<<"100 rupees debited"<<endl;
    };
};
int main()
{
    Bank obj;
    obj.name = "Pradum";
    obj.balance = 1000;
    obj.check_balance();
    obj.withdraw();
    obj.check_balance();

return 0;
};
```

Private Access Specifier

```
#include<iostream>
using namespace std;

class Bank
{
    private:
    string name;
    int balance;
    public:

    void setvalue(string person, int amount)
    {
        name = person;
        balance = amount;
    }

    void check_balance()
    {
        cout<<balance<<endl;
    };
    void print_name()
    {
        cout<<name<<endl;
    }

};
int main()
{
    Bank obj;
    obj.setvalue("pradum", 1000);
    obj.check_balance();
    obj.print_name();
return 0;
};
```

Constructor ::

```
#include<iostream>
using namespace std;

class employee
{
    int id;
    int salary;
    public:
    employee()
{
    id = 123;
    salary=100000;
    cout<<"Hello Constructor"<<endl;
}
void print()
{
    cout<<"id ="<<id<<endl<<"salary ="<<salary<<endl;
}
};
int main()
{
    employee pradum;
    pradum.print();
return 0;
};
```

Parameterized constructor

```
#include<iostream>
using namespace std;

class employee
{
    int id;
    int salary;
    public:
    employee(int num, int amount)
{
    id = num;
    salary=amount;
    cout<<"Hello Constructor"<<endl;
}
void print()
{
    cout<<"id ="<<id<<endl<<"salary ="<<salary<<endl;
}
};

int main()
{
    employee pradum(11111, 100000);
    pradum.print();

return 0;
};
```

Using:: this

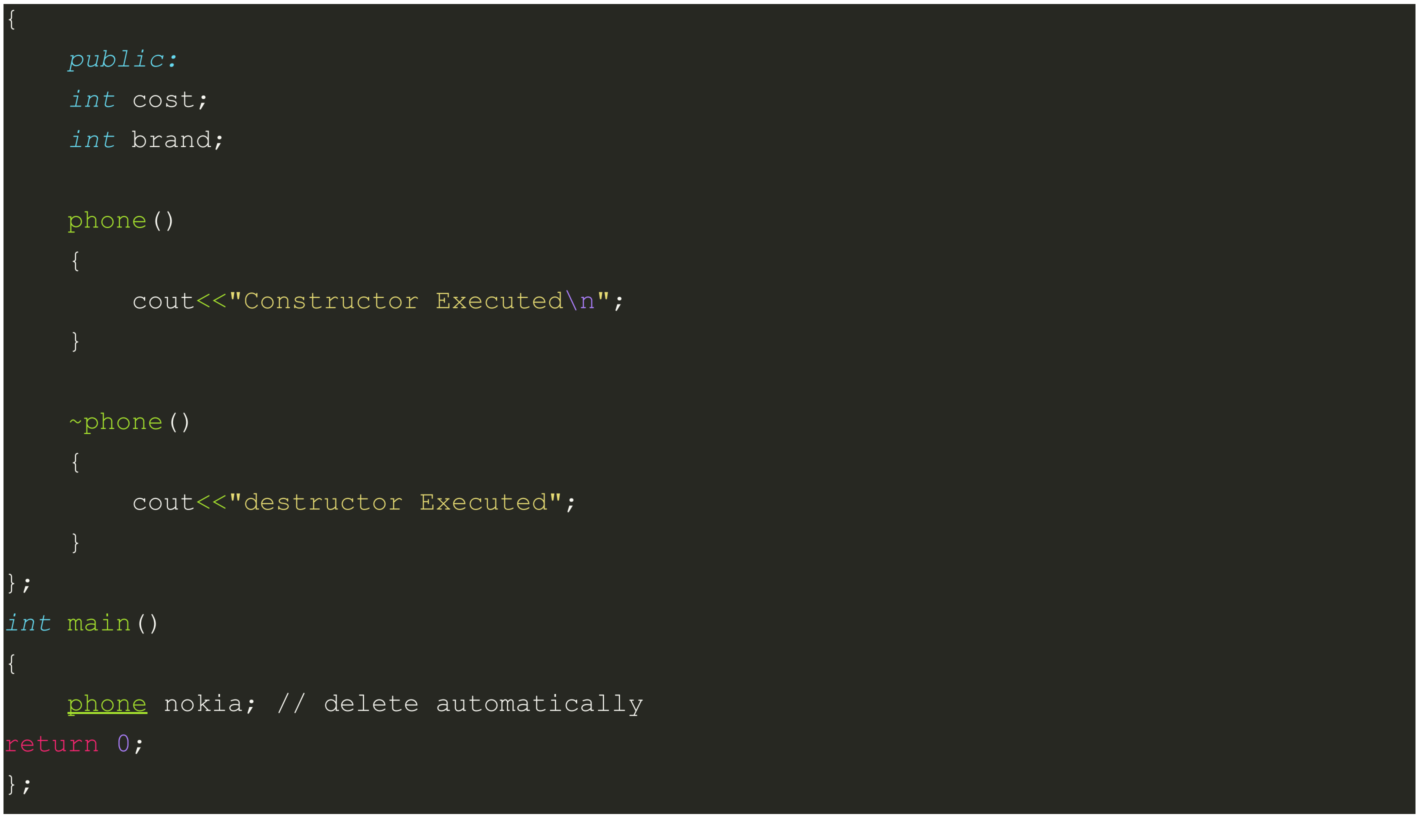
```
#include<iostream>
using namespace std;

class employee
{
    int id;
    int salary;
    public:
    employee(int id, int salary)
{
    this->id = id;    // if you can use same name
    this->salary=salary;
    cout<<"Hello Constructor"<<endl;
}
void print()
{
    cout<<"id ="<<id<<endl<<"salary ="<<salary<<endl;
}
};

int main()
{
    employee pradum(11111, 100000);
    pradum.print();

return 0;
};
```

Using :: pointer

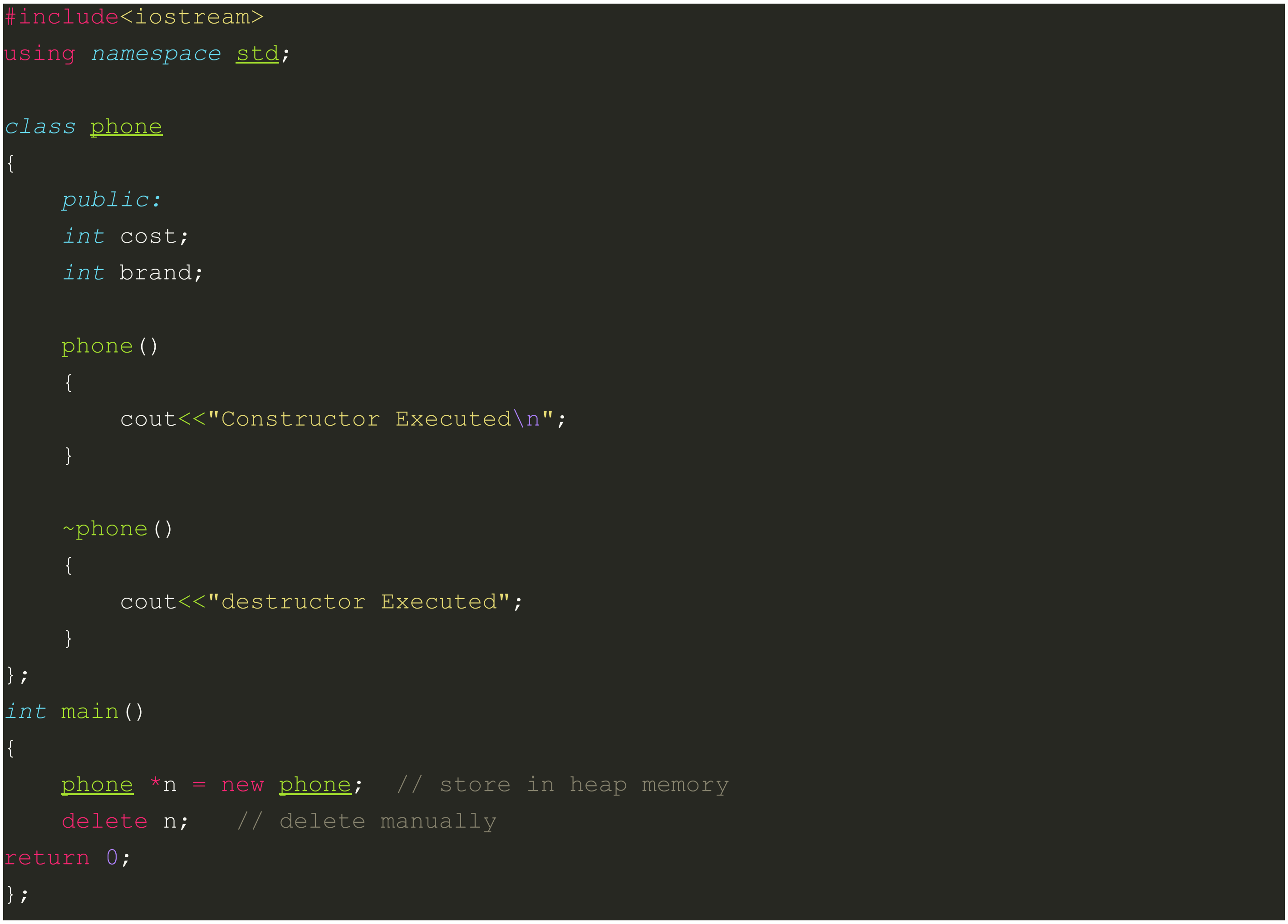
```
#include<iostream>
using namespace std;

class man
{
    public:
    int weight;
    string name;
};
int main()
{
    man *p = new man;
    p->weight= 70;  // (*p).weight = 70;
    p->name = "pradum";  // (*p).name = "pradum";
    cout<<p->weight<<endl<<p->name<<endl;

return 0;
};
```

Destructor :: Automatic release memory

```
#include<iostream>
using namespace std;

class phone
```



```
{
    public:
    int cost;
    int brand;

    phone()
    {
        cout<<"Constructor Executed\n";
    }

    ~phone()
    {
        cout<<"destructor Executed";
    }
};
int main()
{
    phone nokia; // delete automatically
return 0;
};
```



```
#include<iostream>
using namespace std;

class phone
{
    public:
    int cost;
    int brand;

    phone()
    {
        cout<<"Constructor Executed\n";
    }

    ~phone()
    {
        cout<<"destructor Executed";
    }
};
int main()
{
    phone *n = new phone;  // store in heap memory
    delete n;   // delete manually
return 0;
};
```

We learned about OOPs in detail after class.

Linked List

<table>
<tr><th colspan="4">Memery</th></tr>
<tr><td colspan="2">used</td><td>108</td><td rowspan="4">Free space block :: 108, 168, 204, 224
If we store data this block memory ▭
So, every block contains the address to the next block.</td></tr>
<tr><td>168</td><td>used</td><td>204</td></tr>
<tr><td>used</td><td>224</td><td>used</td></tr>
<tr><td colspan="3">used</td></tr>
</table>

Add another element in array : 6

Arr[5] :

8 || address of 4	4 || address of 3	3 || address of 2	2 || address of 7	7 || address of 6	6||null

Delete element in array : 3

Arr[5] :

8 || address of 4	4 || address of 2	3 || {delete}	2 || address of 7	7 || address of 6	6|| null

How to create :-

int	add		int	add		int	add	
10	200	------------->	31	180	------------->	25	null	
p->108			200			180		

```
int data;
Node *next              // next is a pointer
```

##code::-

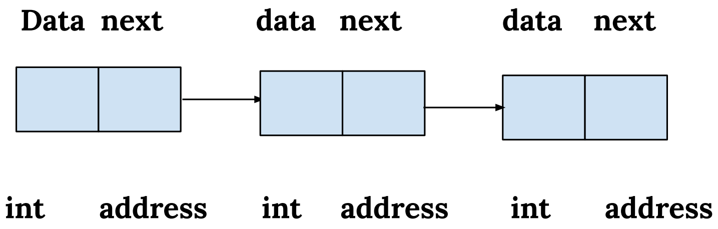
```
#include<iostream>
using namespace std;

class Node
{
    public:
    int data;
    Node *next;
};
int main()
{
    Node *first = new Node;
    first->data = 10;
    cout<<first->data;
return 0;
};
```

LEARN DSA WITH C++

WEEK :: 07 DAY: 02 DATE: 30-05-2023

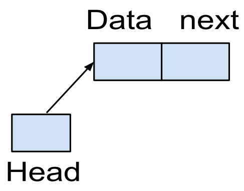
Linked List Part - I

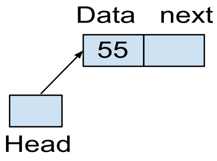
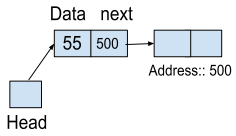
How to define in code ::

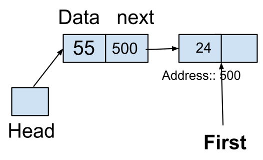


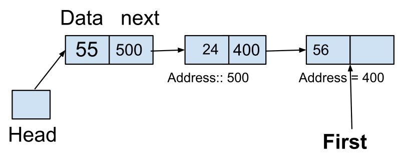
```
class  Node
{
  Public :
    int data;
    Node*  next;
};
```

How to create ::

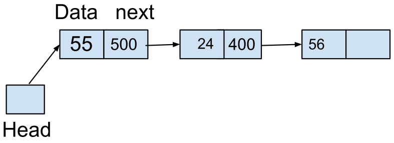
	int main()
	{
	Node * Head = new Node; // Head = Pointer

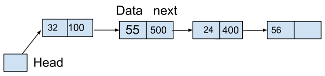
	Head ->Data = 55;
	Head -> next = new Node;

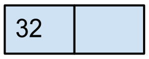
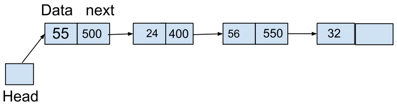

 Data
next
55
500
24
Address:: 500
Head
First
	First = Head;
	First = first->next;
	First->Data=28;

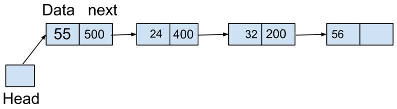

 Data
next
55
500
24
400
56
Address:: 500
Address:: 400
Head
First
	First -> next = new Node();
	First = First -> next;
	First ->data = 56;

Insertion

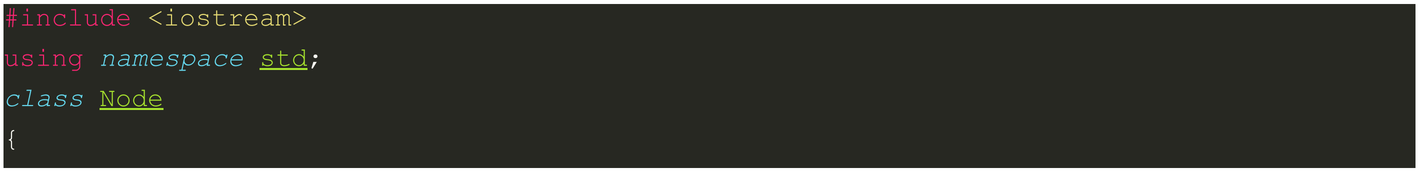


Starting	
	
		
		 Data
		next
		32
		100
		55
		500
		24
		400
		56
		Head
			Node *temp = new node();
			temp -> Data = 32;
			temp -> next = head;
			Head = temp;

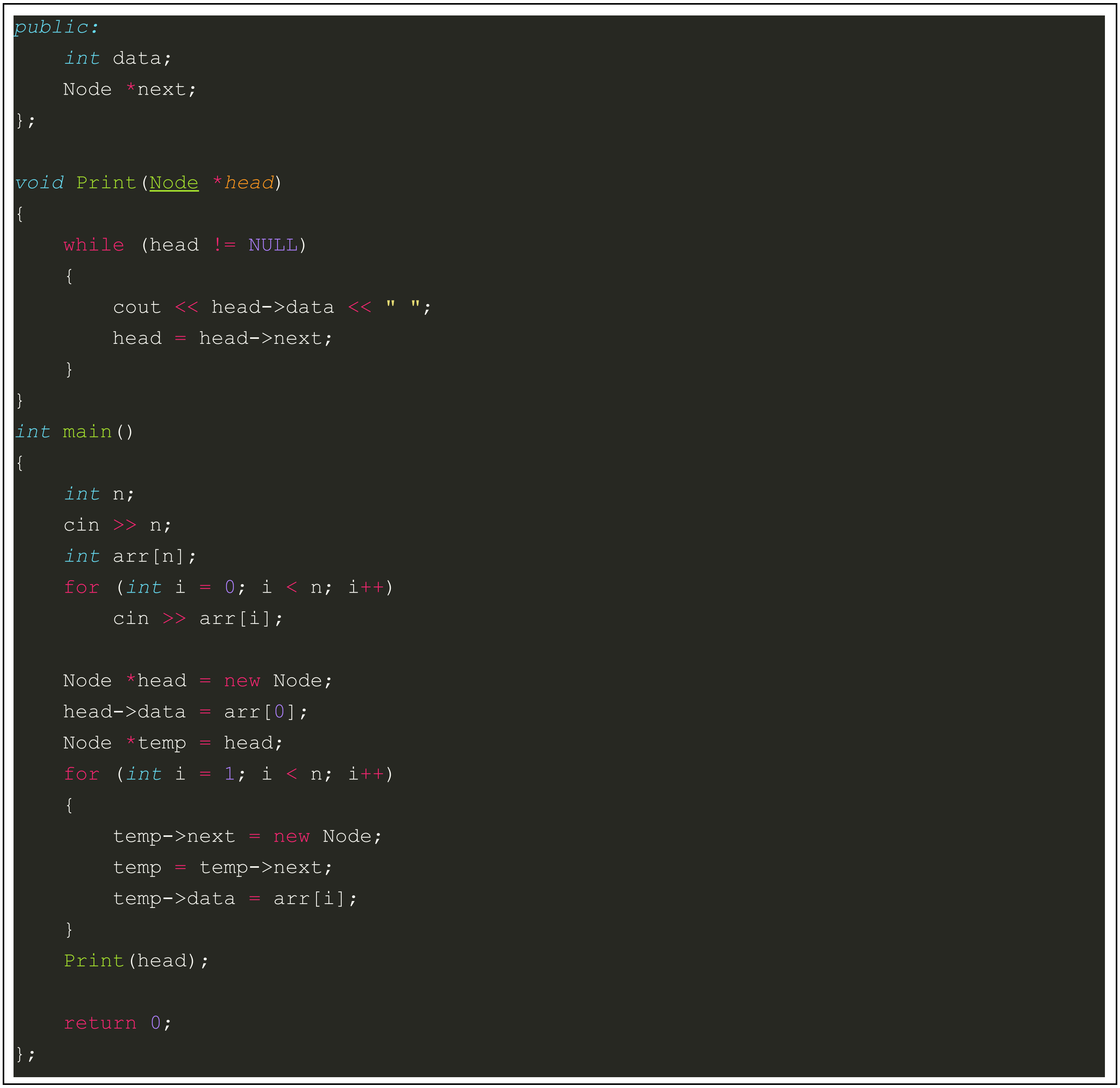
End	
	
		
		 Data
		next
		55
		500
		24
		400
		56
		550
		32
		Head
			while(temp ->next != NULL)
			{
			temp = temp ->next;
			temp -> next = new node();
			temp = temp -> next;
			temp -> data = 32;
			};

Middle	
	
		
		 Data
		next
		55
		500
		24
		400
		32
		200
		56
		Head
			while (temp -> Data != 57)
			{
			temp = temp -> next;
			}
			Node * first = new Node();
			First -> next = temp -> next;
			temp -> next = First;

Create Linked List :-



```
#include <iostream>
using namespace std;
class Node
{
```



```cpp
public:
    int data;
    Node *next;
};

void Print(Node *head)
{
    while (head != NULL)
    {
        cout << head->data << " ";
        head = head->next;
    }
}
int main()
{
    int n;
    cin >> n;
    int arr[n];
    for (int i = 0; i < n; i++)
        cin >> arr[i];

    Node *head = new Node;
    head->data = arr[0];
    Node *temp = head;
    for (int i = 1; i < n; i++)
    {
        temp->next = new Node;
        temp = temp->next;
        temp->data = arr[i];
    }
    Print(head);

    return 0;
};
```

Reverse the linked list :-

Using Recursion :

```
reverse (Node *head)
{
  if(head == NULL);
  return;

  reverse (head -> next);
  Cout << head -> data;
```

Using Loop :

```
Node *prev = NULL;
Node *temp;
while(head -> next)
{
  temp = head -> next;
  head -> next = prev;
  prev = head;
```

};	head = temp;
	};

Add Node before 23 Node : -

```
if(head -> data == 23)            if 23 node have first position
{
  Node *first = new Node;
  First -> data = 7;
  First -> next = head;
  head = first;
};
while (temp -> next -> data != 23)
{
   temp = temp -> next;
};
```

Delete First Node : -

```
First = head;
Head = head -> next;

delete(p);
```

Delete Last Node :

```
while (First -> next -> next)
 {
     First = First -> next;
};
delete(First->next);
First -> next = NULL;
```

Delete Middle Node :-

```
temp = first -> next;
First -> next = First -> next -> next;
delete (temp);
```

Count nodes of linked list << GeeksForGeeks >>

```
class Solution
{
    public:
    //Function to count nodes of a linked list.
    int getCount(struct Node* head){

        //Code here
        int count = 0;
        while(head)
        {
            head = head-> next;
            count++;
        }
        return count;

    }
};
```

Insert in a Sorted List << GeeksForGeeks >>

```
class Solution{
  public:
    // Should return head of the modified linked list
    Node *sortedInsert(struct Node* head, int data) {
        // Code here
        if(head -> data > data)
        {
            struct Node * temp = new Node(data);
            temp -> next = head;
            head = temp;
            return head;
        };

        struct Node * first = head;
        while(first -> next && (first -> next ->data<data))
        {
            first = first -> next;
        };
        struct Node * temp = new Node(data);
        temp-> next = first -> next;
        first ->next = temp;

        return head;
    }
};
```

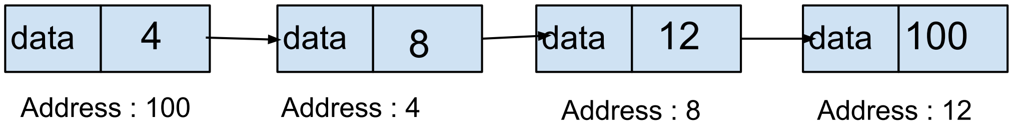
LEARN DSA WITH C++

WEEK :: 07 **DAY: 03** **DATE: 31-05-2023**

Linked List Part - II

# Circular Linked List :

The last node pointed to the first node.



How To Create

```
Node *first = head;
While(first ->next)
{
  first = first -> next
};
first -> next = head;
```

Manipulate Any element	
Array	**Linked List**
We can access it directly. Time Complexity = O(1)	We can't access it directly. We go one by one then we can access it. Time Complexity = O(N)
We can't add elements in the first position directly. Move every element to the next position. Time Complexity = O(N)	We can add directly in the first position. Time Complexity = O(1)
We can't delete directly in the first position. Move every element there after position. Time complexity = O(N)	We can directly change the pointer position. Time Complexity = O(1)
Search any element using binary search Time Complexity = O(logN)	Search any element manually Time Complexity = O(N)

Why does Vector copy all arrays in another memory when adding an extra element?

Arr[3] = {6, 8, 3} ;   add = 1

Memory

6	8	3	used	
	used	used		
used	6	8	3	1

When adding another element in array using vector all arr elem copy because its take free space continuous way.

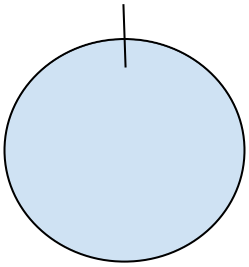
Check If Circular Linked List   << GeeksForgeeks >>

```
bool isCircular(Node *head)
{
   // Your code here
   Node *first = head;
   while(first -> next && first ->next != head)
   {
       first = first ->next;
   };

   if(first -> next ==NULL)
   return 0;
   else
   return 1;
}
```

Intersection Point in Y Shaped Linked Lists   << GeeksforGeeks >>

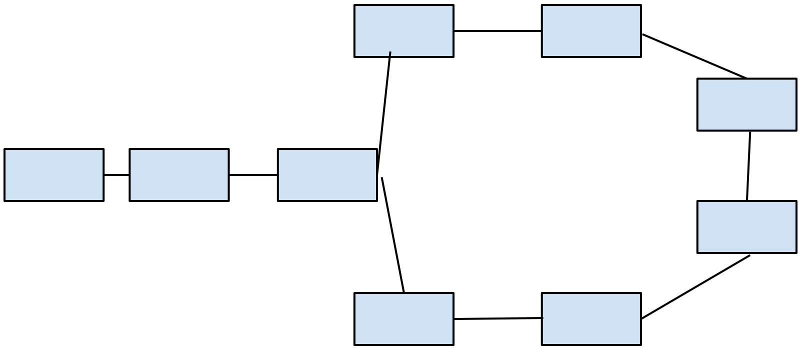
```
int intersectPoint(Node* head1, Node* head2)
{
    // Your Code Here
    int count1 = 0, count2 = 0;
    Node* first = head1;
    Node* second = head2;

    while(first)
    {
        count1++;
        first = first->next;
    }

    while(second)
    {
        count2++;
        second = second->next;
    };
    while(count2>count1)
    {
        count2--;
        head2 = head2->next;
    };
     while(count1>count2)
    {
        count1--;
        head1 = head1->next;
    };
    while(head1 && head2 && head1 != head2)
    {
        head1 = head1->next;
        head2 = head2->next;
    };
    if(head1 && head2)
    return head1 -> data;

    return -1;
}
```

How to find its circular :: there are no clues.

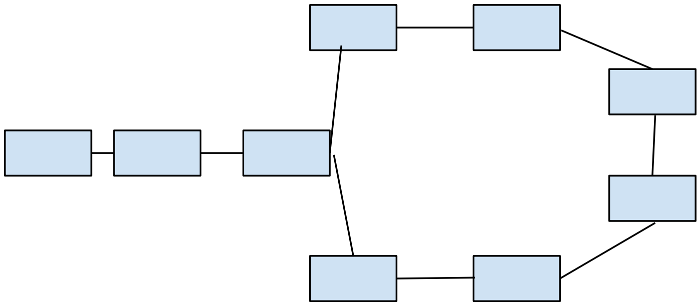


We took two runners. Runner1 & Runner2.
Runner1 runs 10 km/h or Runner2 runs 20 km/h.
If they meet again, we understand this path is circular.

Find loop in linked list ::



Take Two Pointer
Ponter1 takes one step & Poiter2 takes 2 steps.
If they meet any point we understand it has a loop.

Convert into single line linked list ::	
	**Take two pointers.** **Pointer1 takes steps one by one.** **Pointer2 takes step 2 node.** **When they meet each other then one pointer takes steps one by one from the first node or second pointer also takes step one by one from the meeting node. Which node they are meeting now. This node disconnects with other nodes.**

LEARN DSA WITH C++

WEEK :: 07 **DAY: 04** **DATE: 01-06-2023**

Linked List Advance Part - I

Check if Linked List is Palindrome << GeeksforGeek >>

```
class Solution{
  public:
    //Function to check whether the list is palindrome.

    Node* Reverse(Node* curr)
    {
        Node* prev = NULL, *next;
        while(curr)
        {
            next = curr ->next;
            curr -> next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }
    bool isPalindrome(Node *head)
    {
        //Your code here
        if(head -> next == NULL)
        return 1;

        Node* first = head, *second = head;
        int count = 0;
        while(first)
        {
            count++;
            first = first -> next;
        };
        count = (count+1)/2-1;
        while(count--)
        second = second -> next;
        first = second ->next;
        second -> next = NULL;

        first = Reverse(first);

        second = head;
        while(first)
        {
            if(first -> data != second -> data)
            return 0;
            first = first -> next;
            second = second -> next;
        };
        return 1;
    }
}
```

Remove loop in Linked List :: << GeeksForGeek >>

```
class Solution
{
    public:
    //Function to remove a loop in the linked list.
    void removeLoop(Node* head)
    {
        // code here
        // just remove the loop without losing any nodes
        if(head == head -> next)
        head -> next=NULL;

        if(!head -> next)
        return;

        Node *slow = head -> next;
        Node *Fast = head ->next -> next;

        while(Fast && Fast -> next && Fast != slow)
        {
            Fast = Fast -> next -> next;
            slow = slow -> next;
        };
        if(!Fast || !Fast -> next)
        return;

        Fast = head;

        if(Fast == slow)
        {
            while(slow->next != Fast)
            slow = slow-> next;
            slow ->next = NULL;
            return;
        }
        while(Fast -> next != slow -> next)
        {
            Fast = Fast ->next;
            slow = slow -> next;
        }

        slow -> next = NULL;
        return;

    }
};
```

Rearrange a linked list :: << GeeksforGeeks >>

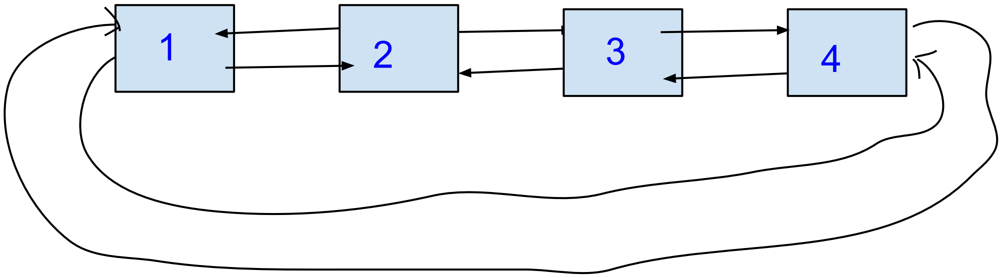
```
class Solution
{
    public:
    void rearrangeEvenOdd(Node *head)
    {
       // Your Code here
       if(!head -> next)
       return;

       Node *first = head, *second = head -> next, *temp = head->next;

       while(second && second ->next)
       {
           first -> next = second ->next;
           first = first->next;
           second -> next = first ->next;
           second = second ->next;
       };
       first ->next = temp;
    }
};
```

Doubly - Linked List ::

Problem Solve : **It helps to go reverse .**
There are all nodes containing two addresses. Go to the front and bake side.

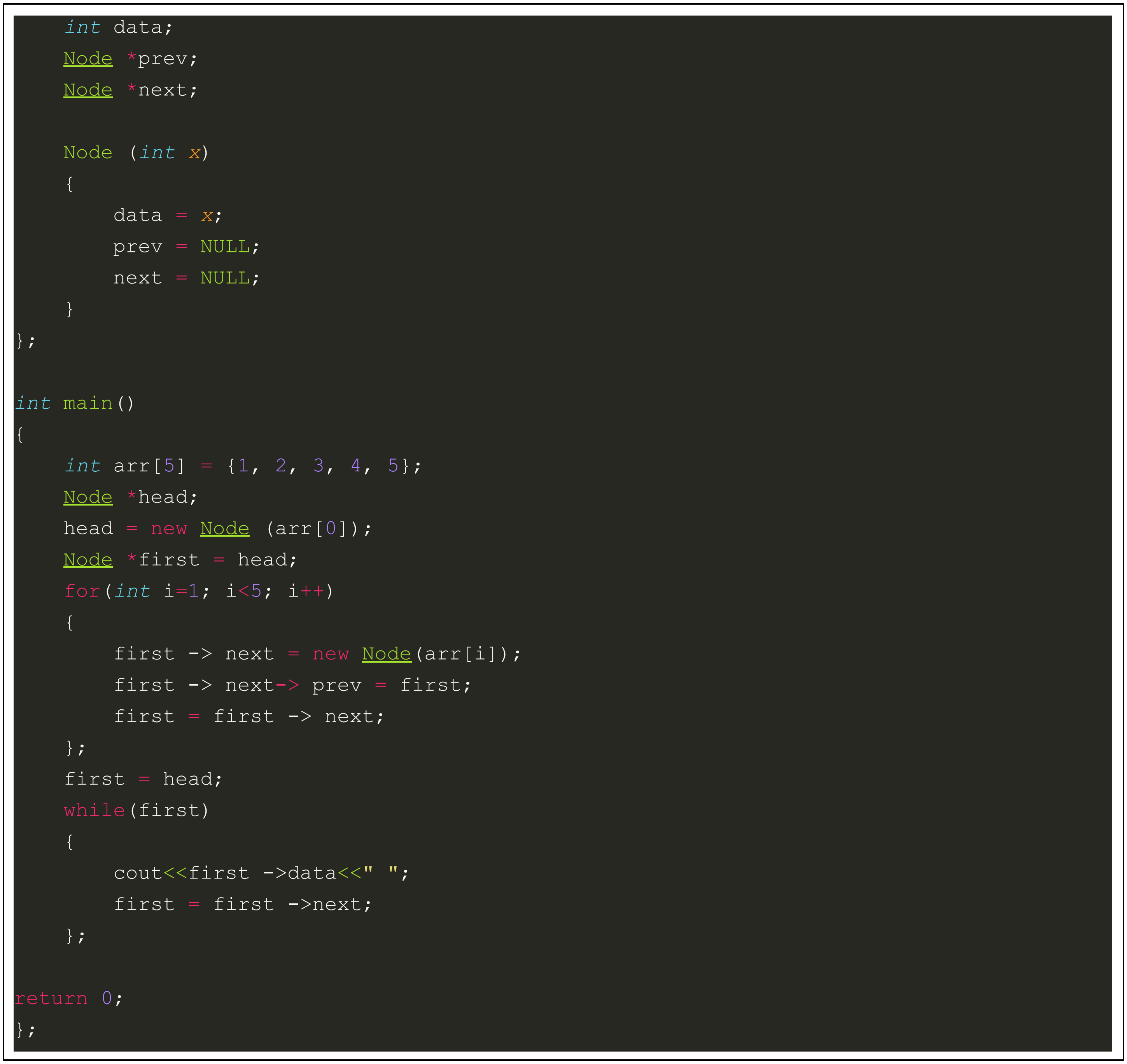
```
class Node
{
    public:
    int data;
    Node * next;
    Node * prev;
};
```

Circulatory Doubly List ::



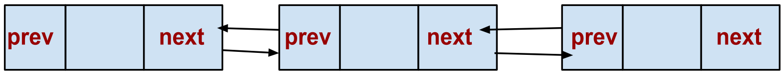
Create Doubly-Linked List ::

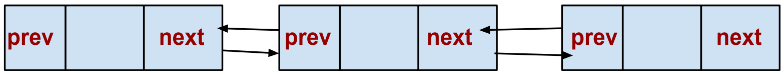
```
#include<iostream>
using namespace std;
class Node
{
    public:
```



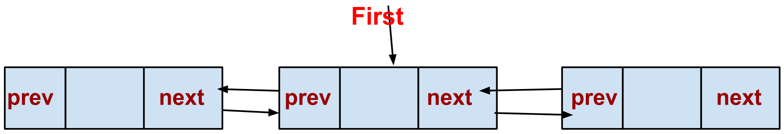
```
    int data;
    Node *prev;
    Node *next;

    Node (int x)
    {
        data = x;
        prev = NULL;
        next = NULL;
    }
};

int main()
{
    int arr[5] = {1, 2, 3, 4, 5};
    Node *head;
    head = new Node (arr[0]);
    Node *first = head;
    for(int i=1; i<5; i++)
    {
        first -> next = new Node(arr[i]);
        first -> next-> prev = first;
        first = first -> next;
    };
    first = head;
    while(first)
    {
        cout<<first ->data<<" ";
        first = first ->next;
    };

return 0;
};
```

Doubly Linked List ::		

Delete first		**Node *head;**
		Node *first;
		Head = head -> next;
		Head ->prev = NULL:
		First -> next = NULL;
		delete (first);

Delete Last		**Node * head;**
		Node *first;
		Node *second;

		while(first != NULL)
		{ first = first ->next; }
		while(second == NULL)
		{second=second->next;}
		First -> next = NULL:
		second -prev =NULL;
		delete (second);

Delete Middle		First -> prev -> next = First ->next;
		First -> next ->prev = first -> prev;

LEARN DSA WITH C++

WEEK :: 07 **DAY**: 05 **DATE**: 02-06-2023

Linked List Advance Part - II

Clone a linked list with next and random pointer:: << GeeksforGeek>>

```
class Node
{
  Public:
     int data;
     Node * next;
     Node * arr;
  Node (int x)
  {
       data = x;
       next = NULL;
       arr   =  NULL;
  }
}
Node * clone = head;
while(clone)
{
  Node * temp = new Node(clone-data);
  temp -> next = clone -> next;
  clone -> next = temp;
  clone = clone -> next -> next;
}
while(clone)
{
   if(clone -> arb)
  clone -> next -> arb = clone ->arb ->next;
  clone = clone -> next - next;
}
header =(first -> next);
while(first)
{
   first -> next = second -> next;
   if(first ->next)
   second -> next = first -> next -.next;
   first =  first -> next;
   second = second ->next;
}
Return header;
```

```
class Solution
{
    public:
    Node *copyList(Node *head)
    {
        //Write your code here
        Node *clone = head;
        Node *temp;

        while(clone)
        {
            Node *temp = new Node (clone->data);
            temp-> next= clone ->next;
            clone ->next = temp;
            clone = temp ->next;
        };

        clone = head;
        while(clone)
        {
            if(clone ->arb)
            clone ->next ->arb = clone ->arb
->next;
            clone = clone ->next ->next;
        };

        Node* ans = head -> next;
        clone = head;
        temp = head -> next;

        while(temp)
        {
           clone ->next = temp ->next;
           clone = temp;
           temp = temp ->next;
        };
        return ans;
    }

};
```

Reverse a sublist of a linked list :: << GeeksForGeek >>

```
class Solution
{
    public:

    Node *Reverse(Node *head, int n)
    {
        Node* prev =NULL, *next;
        while(n--)
        {
            next = head ->next;
            head ->next = prev;
            prev = head;
            head = next;
        };

        return prev;
    }
    Node* reverseBetween(Node* head, int m, int n)
    {
        //code here
        if(m==n)
        return head;
        int count1 = m, count2 = n;
        Node * first = NULL, *second =head;

        while(count2--)
        {
            count1--;
            if(count1 ==1)
            {
                first = second;
            };
            second = second ->next;
        };

        if(!first)
        {
            if(!second)
            return Reverse(head, n);

            else
            {
                Node *temp = head, *ans;
                ans = Reverse(head, n);
                temp ->next = second;
                return ans;
            }
        }
        Node *temp = first -> next;
        first -> next = Reverse(first->next,n-m+1);
        temp -> next = second;

        return head;
    }
};
```