**DATA STRUCTURES C++**

**1. what is Data?**

Data means a characters or symbols that are performed by a computer and it can be transmitted and stored in the form of electrical signals

and recorded on magnetic, optical through recorded media

**2. when data becomes information?**

it means the computer performed by a data and produces some known data is called information.

example:-

"MJANO WDAOG" these are the characters and it won't be understandable by using this data

the computer process some operation that we required then it produce some data that is called information.

**3. what is data structure ?**

Data structure is a way to organize the data in a systematical way and maintaining the structure of a data in efficient manner

for example:- by creating multiple variables for same data why can't we create array.

storing a string is equivalent to storing a sequence of characters so array is needed hence the data structure is useful

REAL TIME EXAMPLE OF DATA STRUCTURE

\*\*\*UNDO AND REDO\*\*\*\* operations in google doc and Microsoft word are using this undo and redo features

STACK DATA STRUCTURE is used for undo and redo features

so stack means LAST IN FIRST OUT scenario.

so in the undo and redo feature:-

1. if we undo:- what we enter last thing that will pop and that will push in the redo bucket example "ctr & ctr y";

|  |
| --- |
| ISMANOJ |
| NAME |
| MY |
| HELLOWORLD |
| **UNDOSTACK** |

UNDO

If once pressed CTRLZ -> ISMANOJ will pop and it will push fro REDO STACK

Again undo NAME will pop and it will push to redo stack so on.

**TYPES OF DATA STRUCTURES**

1. Linear Data structure
2. Non linear data structure
3. **Linear Data structure:-**

* Linear data structure means each elements have one precedence and one succeed-er
* The data should be sequential order and efficient
* Example :- Array, queue, stack and Linked list are the example f this data structure.
* Exception:- is first and last element doesn't have precedence and one succeed-er

1. **Non linear data structure**

* Non linear data structure means it may have one precedence and more succeed-er
* Data are not be in sequential order
* Example:- Tree DS and Graph DS

1. **Static Data structure**

* In this data structure the memory should be allocated at compile time and the size is fixed
* Therefore maximum size id fixed
* Advantages is fast data access and disadvantage is slow insertion and deletion
* Example: for array the maximum size is fixed at compile time only.

1. **Dynamic Data structure**

* These data structures the memory should be allocated at runtime
* So there is no memory leak and we can store the elements dynamically in a array
* Here fast insertion and deletion and slower access
* Example:- vectors,list etc

**ASCYMPTOMIC ANALYSIS**

The term asymptotic analysis means comparing the data structures, that means how the data structure are more efficient and faster like that. On the basis of time complexity and memory we are comparing the data structure .

An ideal data structure means it can executes the data faster and less memory consumption.

On What basis we could comparing the data structure ?

On the basis of time complexity and efficiency on operation performed by two data structures.

For example:-1scenario 1 [ “Array has 8 elements but the array can store up to 100 elements in this case I need to add the element at the beginning “]

In the above scenario how I’m going to add the element at beginning is, I need to shift all the elements at right and the I need to add the element at beginning so this is so much hard work because all the elements should be shift to right. But in Linked list there is no necessity shift all the element to right. just we have to link the element at first so linked list is best compare to Array as well time complexity and speed.

**Time complexity:-** how to calculate the time complexity of a two data structure with different size of input?

-> Examining the Exact runtime calculation is not a best solution to find time complexity of data structures because different machine shows different time complexity based on the size of an input.

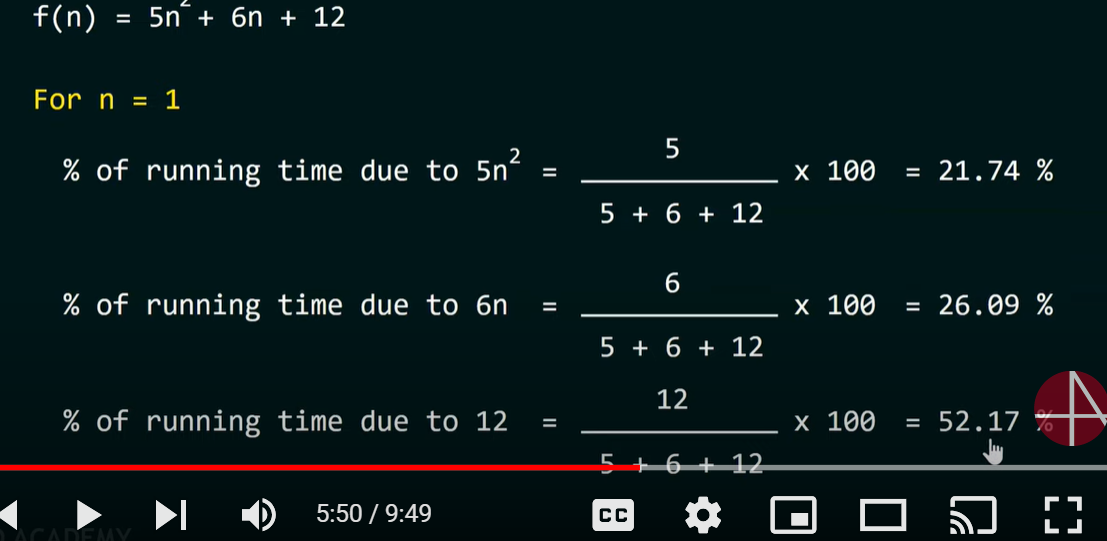
->The running time depends on Size of an input: - because size to make empty slot at front of the array we need to shift array size of times then only we can make empty slot in front of an array

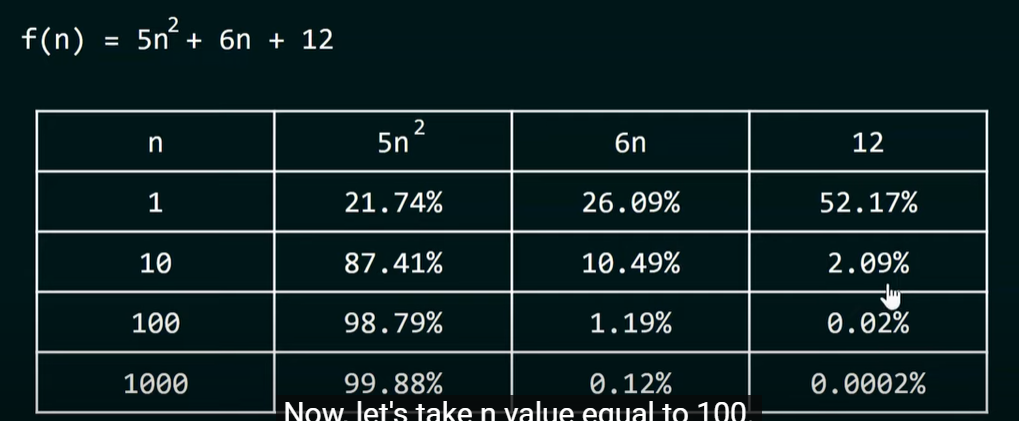
->for example:- we have 3 elements in an array and it’s size is 3 I need to add one element at front, so I need to shift the each elements therefore we shifts 3 times .

-> if we have 10000 size means 10000 times we need to shift the elements to make empty slot of an array, therefore it will take 10000units of time. in this case input element at front not costly shifting to make slot at front is costly

->conclusion time complexity of a data structure is mainly depends on size of the array or size of an inputs.

**TIME COMPLEXITY CALCULATION**





**TIME SAVING CHART IN DSA**

What Is asymptotic complexity?

Approximate measure of time complexity is called asymptotic complexity in this complexity what we are not measuring exact running time complexity in different machine.

Day2:

**VOID POINTRS**

* Void pointer is the pointer doesn't have any associated data type , it can store any of the data without knowing its data type.
* We can initialize the address of the any data type memory, but if we want to access that data we need to typecast it, otherwise, we will get error

**#include<iostream>**

**int main() {**

**void\* ptr;**

**int num = 10;**

**ptr = &num;**

**std::cout <<\*(int\*)ptr;//here we can’t directly dereferenced.first we have to typecast then we have to dereference d the ptr;**

**}**

**What is the use of void pointer then?**

The malloc and calloc are the two builtins in c these are used to allocate the memory at runtime , these two builtins returns the void pointer so the void pointers are essential.

**NULL POINTERS**

* Null pointer are the pointers that does not points to any memory location ,It is also called as an empty
* When any pointer are initialized with the NULL then it is considered as a NULL pointer,
* Any pointers which doesn’t have any initialization the better practice to initialize with NULL
* It is used to check whether the memory is allocated for a pointer dynamically using malloc and calloc

#include<iostream>

int main() {

int\* ptr = NULL;

ptr = (int\*)malloc(2 \* sizeof(int));

if (ptr == NULL)

{

std::cout << "Memory not allocated\n";

}

else {

std::cout << "Memory is allocated succesfully\n";

}

}

**DANGLING POINTERS**

Dangling pointers are the pointers which points non existing memory location or deleted, freed memory locations

This dangling pointers are very effective in the code, if we are using or dereference the pointer which may deal-located or deleted its memory on that time we may program get crashed.

To avoid this issue we need to initialize NULL to the freed or deleted memory pointer

SEGMENTATION FAULT :- Means that trying by read or write the data into illegal memory variables or deal-located on that time error will come that is called as segmentation fault.

WILD POINTERS

Wild pointers are the pointers which is not initialized any memory for it, if we used any where in the code the program may crash

#include<iostream>

int main() {

int\* ptr;

\*ptr = 10;

std::cout << ptr;

}

To avoid the crash better initialization of memory or know variable to a pointer

#include<iostream>

int main() {

int var = 10;

int\* ptr;

ptr = &var;

std::cout << ptr;

}

OR

Explicitly allocate the memory and then initialize the value for it

#include<iostream>

int main() {

int\* ptr=new int();

\*ptr = 10;

std::cout << ptr;

}

**STRUCTURES:-** structure is a user defined datatype and it is used make elements of different types as single type

By default the structure will be a public and there is no access specifiers for the structure

Remove the special characters using pass by refference

std::string removespclchars(std::string removespclchars)

{

int size = removespclchars.length();

std::string result;

for (int i = 0; i < removespclchars.length(); i++)

{

if ((removespclchars[i] >= 'A' && removespclchars[i] <= 'Z') || (removespclchars[i] >= 'a' && removespclchars[i] <= 'z'))

{

result += removespclchars[i];

}

}

return result;

}

int main()

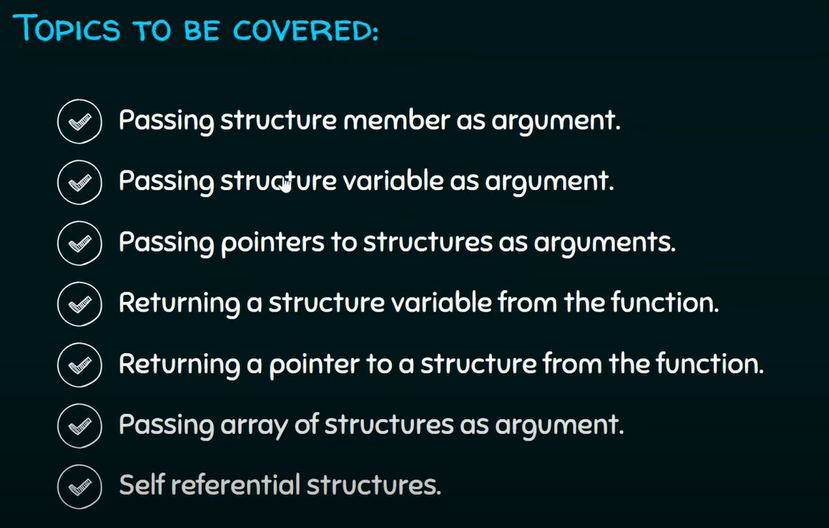
{

std::string mixedchars = "ManJSDC4R23@#42MAmadsk";

std::string letters=removespclchars(mixedchars);

std::cout << letters << "\n";

}



Passing structure member as an argument:-

struct studenthis {

int marks;

char grade;

std::string name;

};

void print(int marks, char grade, std::string name)//

//(int\* marks, char\* grade, std::string name)

{

std::cout << marks << grade << name << "\n";

}

Int main()

{

studenthis obj;

obj.grade = 'A';

obj.marks = 30;

obj.name = "kfnckjrnf";

print(obj.grade, obj.marks, obj.name);//(&obj.grade, &obj.marks, obj.name)

}

Passing structure variable as an argument

* Instead of passing each member we are passing structure object as an argument.

struct studenthis {

int marks;

char grade;

std::string name;

};

Void print(struct studenthis obj)

{

Std::cout<<obj.grade<< obj.marks<< obj.name

}

Int main()

{

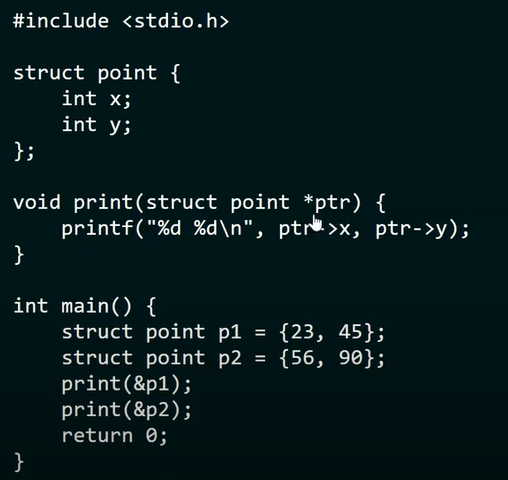
studenthis obj={234,‘A’,”hknjkdkj”};

studenthis obj2={234,‘B’,”hknjkdkj”};

Print(obj); Print(obj2);

}

**Passing pointer to structure as an argument**



Here we are sending the address of the objects as an argument of pointer to structure function

**Returning a structure variable from the function**

struct studenthis {

int marks;

int val;

};

struct studenthis edit(struct studenthis ob)

{

ob.marks++;

return ob;

}

void print(struct studenthis object) {

std::cout << object.marks << " " << object.val << "\n";

}

int main() {

studenthis ob = { 12,13 };

studenthis ob2 = { 100,200 };

ob = edit(ob);

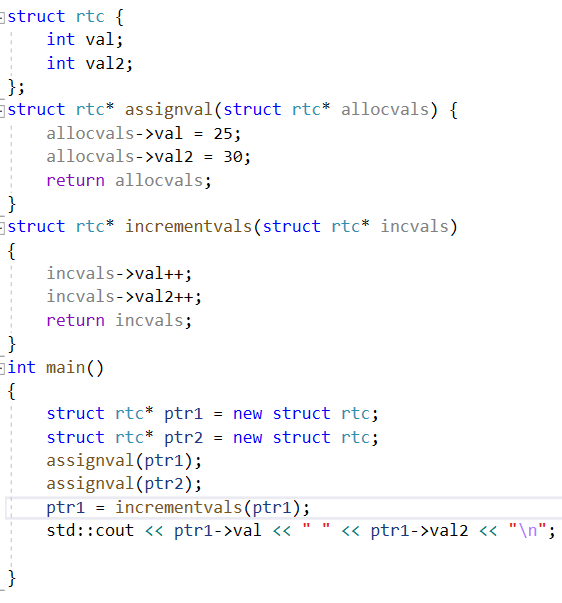
ob2 = edit(ob2);

print(ob);

print(ob2);

}

Returning a pointer to a structure from the function



Refer above program to, return a pointer to a structure from the function , Here we are returning the pointer from the function .

**Pass array of a structure as an argument**

struct rtc {

int val;

char let;

};

void pass(struct rtc arr[]) {

for (auto i = 0; i < 5; i++)

{

std::cout << arr[i].val << arr[i].let << "\n";

}

}

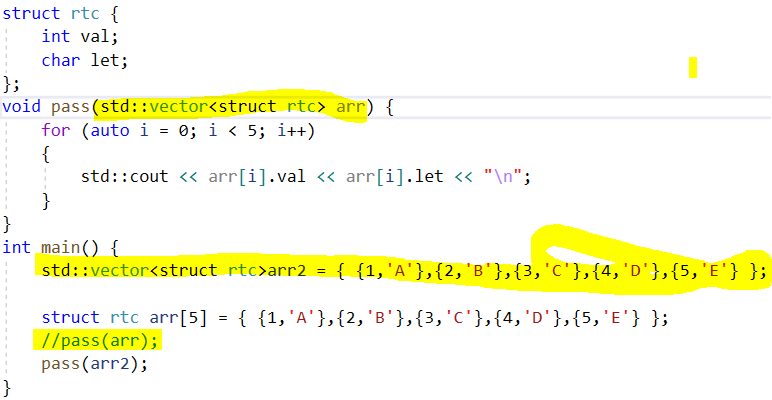
int main() {

struct rtc arr[5] = { {1,'A'},{2,'B'},{3,'C'},{4,'D'},{5,'E'} };

pass(arr);

}

Above code is match with the maps



By using STL vector passed an array of structure as an argument

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| DATA STRUCTURES CONCEPTS |

| |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Self Referential structure :- SRS is structure which contains pointer to a structure of same type. It means using the same structure the members will be defined

For example:-

struct root {

root\* left;

int data;

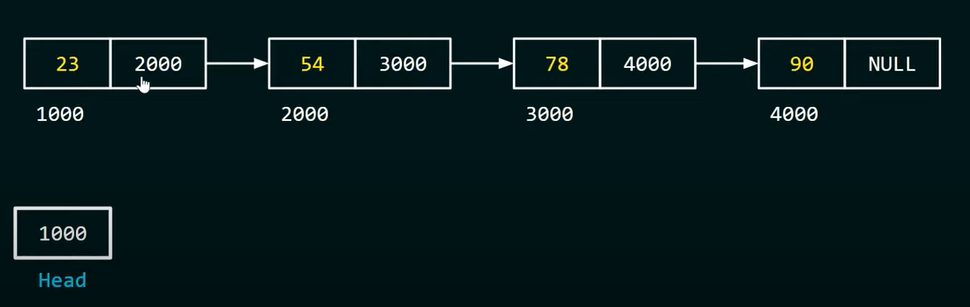
root\* right;

};

TYPES OF LINKED LIST

* Single linked list:- Nodes can connect with the forward direction nodes using their address.
* Doubly linked list:- Nodes can be link with forward and backward nodes using the address
* Circular linked list:-Nodes can be connected in circular order, it means first can be connect with last node

Single linked list



Here Head is the pointer and it is used to access the first node of linked list

**Array vs Linked list**



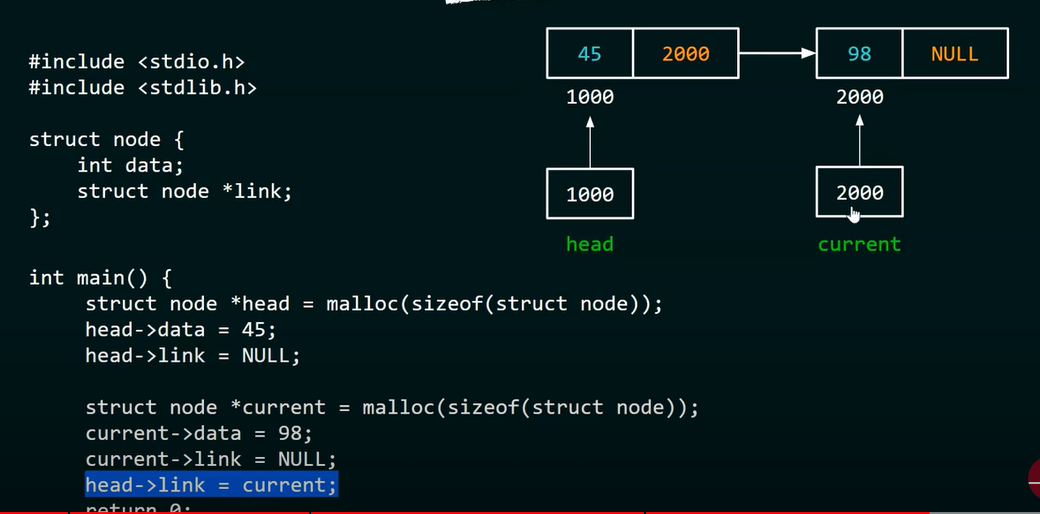
Above image shows the time complexity comparison

when counting the number of elements Array has better time complexity than linked list,and printing the elements are same time coplexity in both DS.

* Array is a sequential and consecutive memory representation, and the data which we have to store that is fixed, while defining the array ,it’s size must be defined.
* Linked list is not consecutive memory representation it is a linked representation because each nodes should be connected and scattered with each other

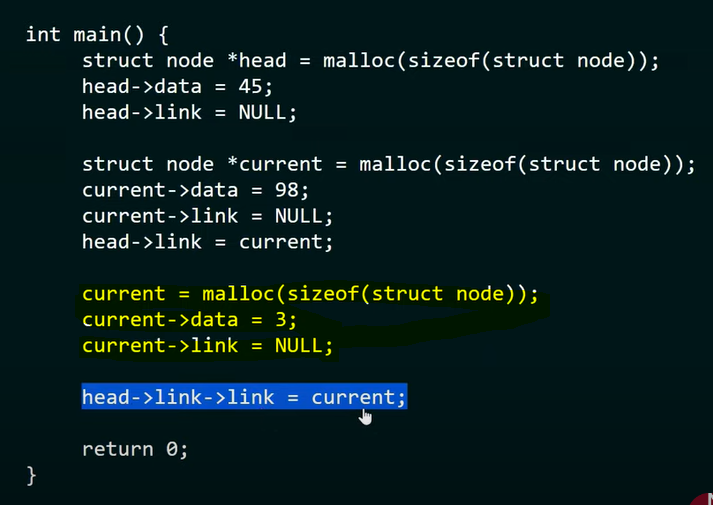
**LINKED LIST PROGRAMMING**

* If we considered a single linked list there must be a single link and we could have multiple data members, it means single pointer should be present in structure and could be a multiple data members.



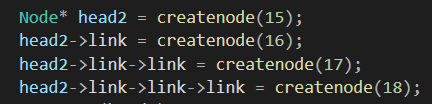
Here there is a head link is updated with the address of current pointer

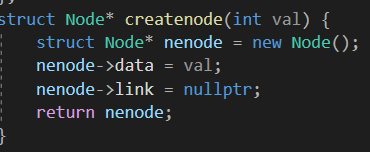
If we are not creating the current pointer, head pointer needs to point the 2nd node on that time 1st node reference would have loss, so by creating the current pointer to point a second node and that will be link with the head->link, in this case 1st node is contains the address of 2nd node



In above code instead of creating the pointer to every single node reusing the pointer of second node, by reusing the pointer we can able to access all the pointer links,

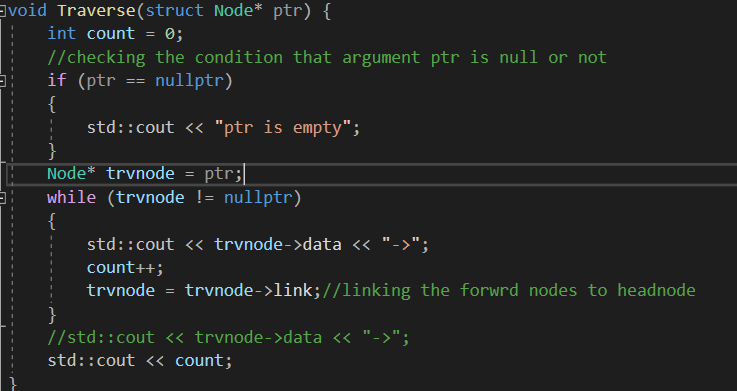
**CREATING THE NODES USING FUNCTION**





**TRAVESRING A SONGLE LINKED LIST**

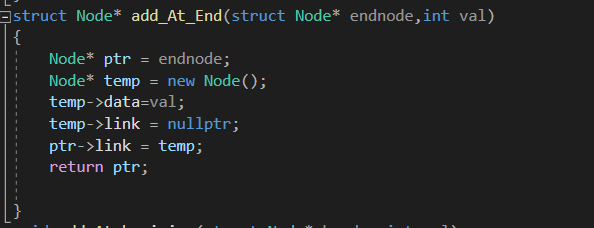
Traversing a single linked list means visiting a each node of a single linked list until the end node is reached.



In above code how the traversing happening is

* In while there is a condition that trvnode is not equal to nullptr and in every iteration the trvnode will be updated with next node.
* After trvnode reaches the last node, then there is a null so the loop will get false because there is a null ptr at last node then it will come out of the loop.

**ADDING THE NODES AT THE END**



In above example the node is adding at the end. And after creating the node ,it is connected with head node.  
**EXAMPLE CODES**

struct Node {

Node\* link;

int data;

};

struct Node\* createnode(int val) {

struct Node\* nenode = new Node();

nenode->data = val;

nenode->link = nullptr;

return nenode;

}

void Traverse(struct Node\* ptr) {

int count = 0;

//checking the condition that argument ptr is null or not

if (ptr == nullptr)

{

std::cout << "ptr is empty";

}

Node\* trvnode = ptr;

while (trvnode != nullptr)

{

std::cout << trvnode->data << "->";

count++;

trvnode = trvnode->link;//linking the forwrd nodes to headnode

}

//std::cout << trvnode->data << "->";

std::cout << count;

}

struct Node\* add\_At\_End(struct Node\* endnode,int val)

{

Node\* ptr = endnode;

Node\* temp = new Node();

temp->data=val;

temp->link = nullptr;

ptr->link = temp;

return ptr;

}

void add\_At\_begining(struct Node\* bnode, int val)

{

Node\* ptr = bnode;

Node\* beginnode = new Node();

beginnode->data = val;

beginnode->link = bnode;

}

int main()

{

/\*Node\* head = new Node();

head->data = 12;

head->link = nullptr;

Node\* current = new Node();

current->data = 13;

current->link = nullptr;

head->link = current;

current = new Node();

current->data = 14;

current->link = nullptr;

head->link->link = current;\*/

Node\* head2 = createnode(15);

head2->link = createnode(16);

head2->link->link = createnode(17);

head2->link->link->link = createnode(18);

Node\* endnode = head2;

endnode = add\_At\_End(endnode, 24);

endnode = add\_At\_End(endnode, 25);

Traverse(endnode);

}

**INSERTING A NODE AT BEGINNING**

struct Node\* addatbegin(struct Node\* current, int val)

{

Node\* ptr = current;

//ptr->data = val;

Node\* newnode = new Node();

newnode->data = val;

newnode->link = ptr;

ptr = newnode;

return ptr;

}

int main() {

//oropertaotr();

Node\* head = new Node();

head->data = 12;

head->link = nullptr;

Node\* current = new Node();

current->data = 15;

current->link = nullptr;

head->link = current;

head = addatbegin(head, 15);

Node\* ptr = head;

while (ptr != nullptr)

{

std::cout << ptr->data << "->";

ptr = ptr->link;

}

//traverse(head);

}

**INSERTING AT AND AND BEGINNING PARALLELY**

void traverse(struct Node\* head) {

int count = 0;

if (head == nullptr)return;

Node\* trvnode = new Node();

trvnode = head;

while (trvnode != nullptr)

{

std::cout << trvnode->data << "->";

trvnode = trvnode->link;

count++;

}

}

// we can use double pointer because to update the node of main function by using dereferencing and we have to send as an pass by refference instead of pass by value in main function

// if we are not used & here means the nodes becomes to local to addat end function so it might not reflect to main function

void addatend(struct Node\*\* current, int val)

void addatbegin(struct Node\* &current, int val)

{

Node\* newnode = new Node();

newnode->data = val;

newnode->link = current;

current = newnode;

//return current;

}

// we can use double pointer because to update the node of main function by using dereferencing and we have to send as an pass by refference instead of pass by value in main function

// if we are not used & here means the nodes becomes to local to addat end function so it might not reflect to main function

void addatend(struct Node\*\* &current, int val)

void addatend(struct Node\* &current, int val)

{

Node\* temp = new Node();

temp->data = val;

temp->link = current;

current = temp;

}

int main() {

Node\* head = nullptr;

addatbegin(head, 1);

addatbegin(head, 2);

addatbegin(head, 3);

std::cout << "add at begin to linked list is\n";

traverse(head);

addatend(head,100);

addatend(head, 200);

addatend(head, 300);

std::cout << "\n add at end is\n";

traverse(head);

}

**INSERTING ARRAY AT BEGIN**

void add\_Array() {

int arr[6] = { 1,2,3,4,5 };

for (int i = 4; i >= 0; i--)

{

arr[i+1] = arr[i];

}

arr[0] = 6;

for (int i = 0; i <= 5; i++)

{

std::cout << arr[i];

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

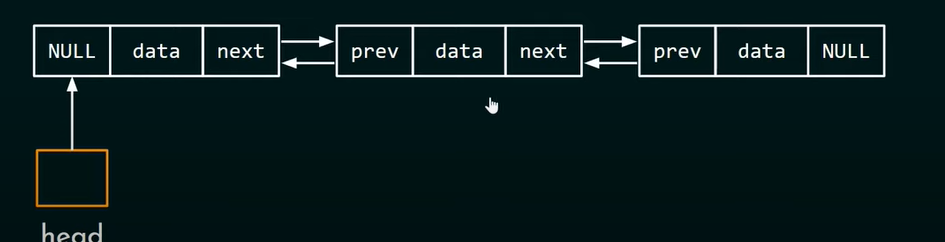
**DOUBLY LINKED LIST**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

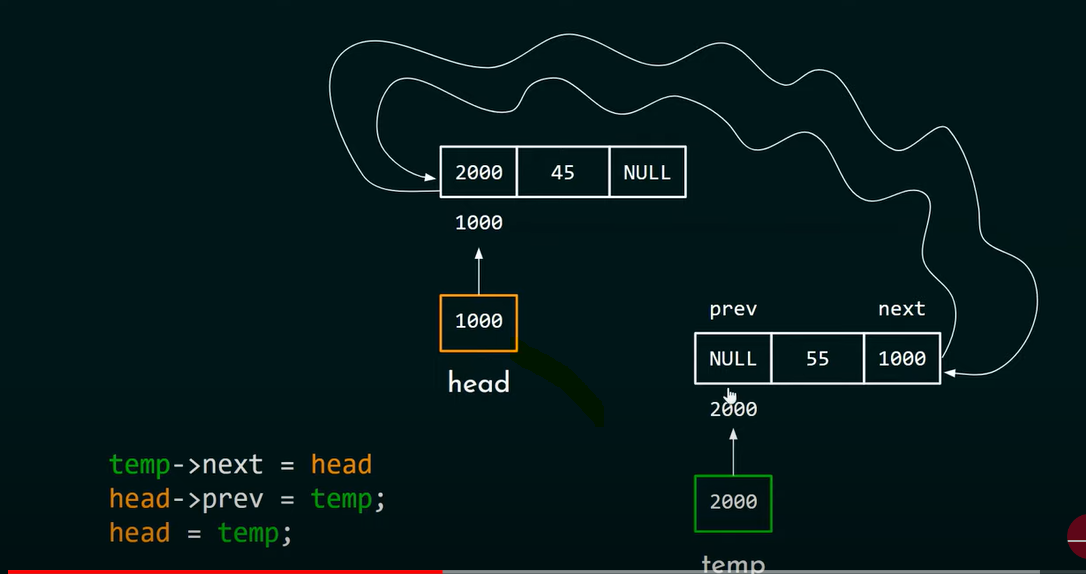
What is meant by doubly linked list?

Doubly linked is a different as Singly linked list in the way of having a extra pointer for each node that points to previous node together with the next pointer and its data , here the node can go backward and forward also

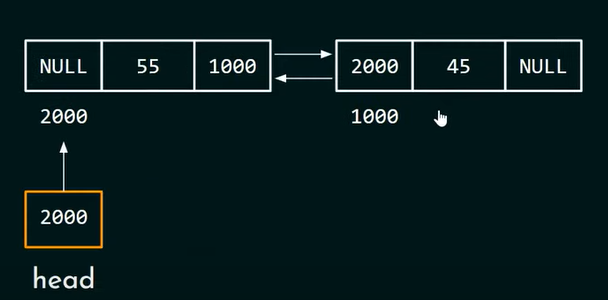
By creating separate pointer in the self referential structure we can able to create doubly linked list



Above fig shows the structure of Doubly linked list.



In above diagram show the scenario of inserting the node at beginning of a root node.

1. Temp->next = head:- we are updated next node with head pointer
2. Head->prev=temp:- we are transferring the head previous pointer to temp node so connection will establish
3. Head=temp in this case the temp node will become the first node and head will take the ownership of temp node.
4. 

The above fig shows resultant linked list looks after adding the node at beginning

#include<iostream>

struct DBLNK

{

DBLNK\* Prev;

int data;

DBLNK\* Next;

};

struct DBLNK\* createnode(struct DBLNK\* Node, int val)

{

struct DBLNK\* temp = new DBLNK();

temp->Prev = nullptr;

temp->data = val;

temp->Next = nullptr;

Node = temp;

return Node;

}

struct DBLNK\* add\_atbeg(struct DBLNK\* Node, int val)

{

struct DBLNK\* temp = new DBLNK();

temp->Prev = nullptr;

temp->data = val;

temp->Next = nullptr;

temp->Next = Node;

Node->Prev = temp;

Node = temp;

return Node;

}

struct DBLNK\* add\_Atend(struct DBLNK\* Node, int val)

{

DBLNK\* temp = new DBLNK();

temp->data = val;

DBLNK\* tp;

tp = Node;

while (tp->Next != nullptr)

{

tp = tp->Next;

}

tp->Next = temp;

temp->Prev = tp;

tp = temp;

return Node;

}

int main() {

struct DBLNK\* head = nullptr;

struct DBLNK\* ptr;

head = createnode(head, 50);

head = add\_atbeg(head, 30);

head = add\_atbeg(head, 100);

head = add\_Atend(head, 200);

head = add\_atbeg(head, 500);

ptr = head;

while (ptr != nullptr)

{

std::cout << ptr->data << "->";

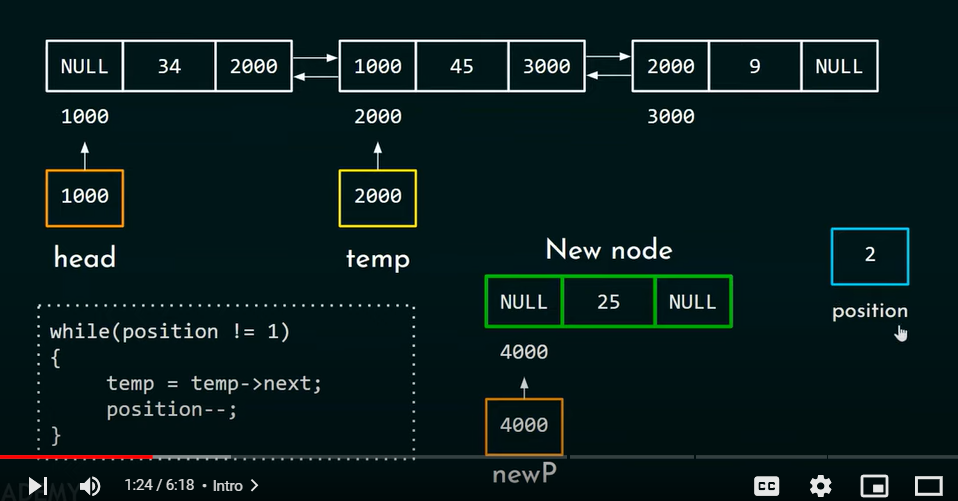
ptr = ptr->Next;

}

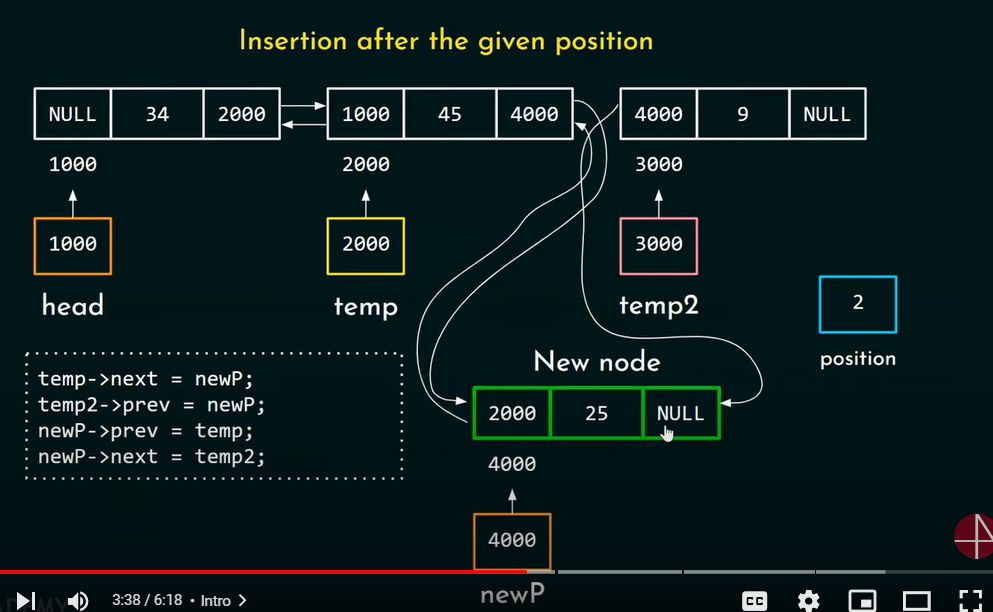
std::cout << "null pointer";

}

INSERTION OF NODES BETWEEN THW POSITION



Above figure shows the while loop for position to add the New node in a Linked list



INSERTIION AT POSITION

struct Node\* newp = nullptr;

newp= createnode(newp, 110);

struct Node\* temp2 = nullptr;

struct Node\* ptr = head;

while (pos != 1)

{

ptr = ptr->next;

pos--;

}

if (ptr->next == nullptr) {

ptr->next = newp;

newp->prev = ptr;

}

else {

temp2 = ptr->next;

ptr->next = newp;

temp2->prev = newp;

newp->prev = ptr;

newp->next = temp2;

ptr = temp2;

}

return head;

Cisco Question To exchange the words first to last and last to first

int main() {

std::string Stringword = "Hello world";

std::string first;

std::string secondword;

bool space = false;

for (char ch : Stringword)

{

if (ch == ' ') {

space = true;

}

else {

if (space)

{

secondword += ch;

}

else {

first += ch;

}

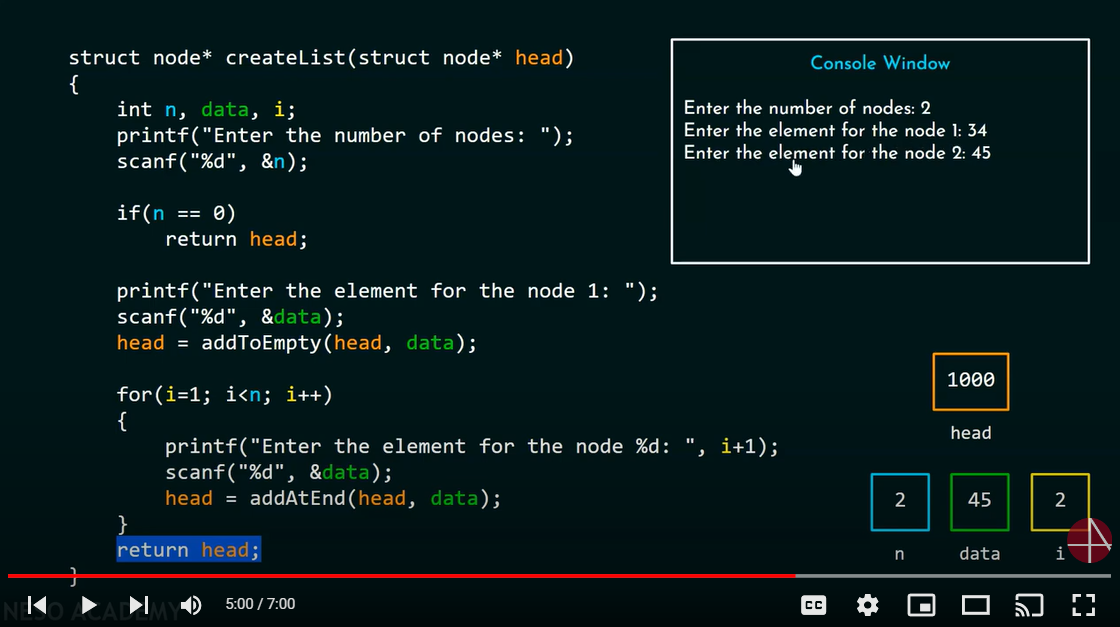
}

}

std::cout << secondword << " " << first << "\n";

return 0;

}



Above figure show the creation of entire doubly linked list b using console input