***What is Data stracture***

As the name indicate , all our work will be with data , and when we speak about data stracture so we are speaking about the ways of handling or working with data , and we have many ways like seaching in data or deleting data or adding data or sorting data or saving data … , for be honest the goal of learning data stracture is not doing this only , but doing this with the best shape , in order of making our programm with hight performance , all this is specify by the type of data structure that you use 

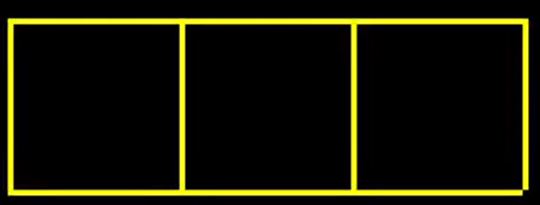
There is no best type of data structure every prolem has it best data structure and we gonna learn in this course how to find the best data structure for our problem , by example if our problem is arriving to a data so the best one will be array because if you want to arrive to any element it enought to write it index but array isn’t good for adding or deleting data

***Resizing Array***

We have many types of data structure among them we have Array but why array is a data structure , because array is a struct of sequential data , The question here why we don’t count on array and make our principle data structre and search for another types , Answear : we find many problem in array and among this problem we have resizing problem wich mean changing the size of an array , so let’s start explanation :

   int list [3];

*know the os will go to the memory and reserve three place for int .*

**

int list [3];

   list[0]=1;

   list[1]=2;

   list[2]=3;

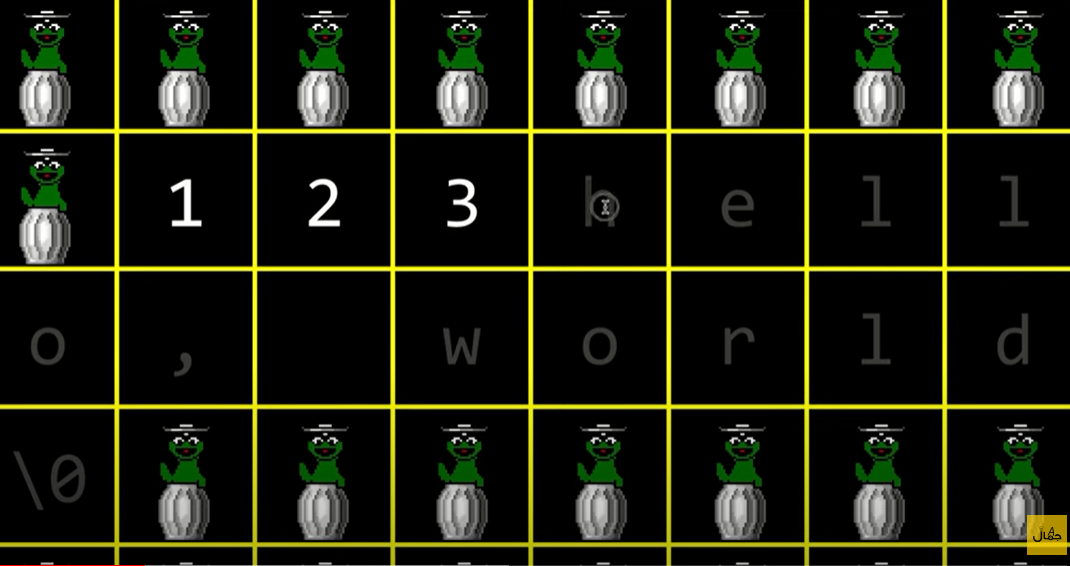
*and know os we go to those places and save in them 1 2 3 *

*the problem appear when i want to add the fourth element to this array : you will say to me that we can add it without problem but …*

*so let’s see the ram with a wide lens :*

*if you tell me that you want to add an index to this array i will ask you . are you sure that after the array there is a free place , because if there are somehing here will have a big problem , as the img show , it possible after our array find somthing saved like a char or othres ..*

*and this shapes mean that there is a free place in memory .*

**

*And the solutin Here is making onther array that contain 4 places , then start move items to the new array , then the old array we will not use it so we will free it .*

#include <stdio.h>

 int main (void){

The result :

1

2

3

4

   int list [3];

   list[0]=1;

   list[1]=2;

   list[2]=3;

   int temp[4];

   for(int i = 0 ; i<3 ; i++){

      temp[i]=list[i];

   }

    temp[3]=4;

    for(int i = 0 ; i<4 ; i++){

      printf("%i\n",temp[i]);

   }

 }

*But we can do that we another method .*

*As we know* «  **int list *[3]*** *« 🡺 reserve for us three places in momory for int , we can do that using malloc*

 int\* list = malloc(3\*sizeof(int));

**Here we reserve three places in memory for int . and sure malloc retrurn the place of the first index so retrun an address so we will save it int\* , and make in mind this is also named an array , because it is the same thing , and for prove this to you , we will do the same program using this method .**

#include <stdio.h>

 #include <stdlib.h>

 int main (void){

       int\* list = malloc(3\*sizeof(int));

       if(list == NULL){

        return 1 ;

       }

       list [0] = 1;

       list [1] = 2;

       list [2] = 3;

       int\* tmp = malloc(4\*sizeof(int));

If list == nulll so programm will go out but before going our it shoud free list .

        if(tmp == NULL){

free(list);

           return 1 ;

         }

      for (int i = 0 ; i<3 ; i++){

        tmp[i]=list[i];

      }

      free(list);

      tmp[3]=4;

      for (int i=0 ;i<4;i++){

        printf("%i \n",tmp[i]);

      }

      free(tmp);

 }

**Note***: Diffrence between the first (normal) array and array using malloc . it simple the first array is making in stack but the array using malloc is making in heap so you have to free the memory manualy .*

*But if this array contain my product and i will resize and continue working on it , so here we will make list point on tmp wich mean make list point on the first index in temp ,so temp=list , so list will contain 4 elements now .*

#include <stdio.h>

 #include <stdlib.h>

 int main (void){

       int\* list = malloc(3\*sizeof(int));

       if(list == NULL){

        return 1 ;

       }

       list [0] = 1;

       list [1] = 2;

       list [2] = 3;

       int\* tmp = malloc(4\*sizeof(int));

        if(tmp == NULL){

           return 1 ;

         }

      for (int i = 0 ; i<3 ; i++){

        tmp[i]=list[i];

      }

      free(list);

      tmp[3]=4;

      list=tmp;

      for (int i=0 ;i<4;i++){

        printf("%i \n",list[i]);

      }

      free(list);

 }

*As you see in the last we free ‘list’ but here you can ask me and say why we don’t free tmp , ANSWEAR because list = tmp so if we free tmp we will free also list and vice versa because they are the same . but if we free list and tmp we will get error because we free them two times.*

*We tell os to free the place that list point then we order to free the place that tmp point so we will free the same place two times because they point the same place in memory ;*

*All this operation will take n steps to be done.*

*If you have 1000 product it we do 100 steps , for move all items from the old array to the new array , imagine with me that you have one million product it will do one million steps, imagine the process will be so slow*

***Know we will learn Realoc() function .***

Realoc is like malloc , malloc reserve a place in memory and realloc change the address in memory . so with realloc i can take data from the old place and put it in a new size in memory , and it make an automatic free for the old address

*Realloc take two arguments , 1- the old place or where will bring the data , what is the old array 2- the new size of the array .*

#include <stdio.h>

 #include <stdlib.h>

 int main (void){

       int\* list = malloc(3\*sizeof(int));

       if(list == NULL){

        return 1 ;

       }

       list [0] = 1;

       list [1] = 2;

       list [2] = 3;

         list = realloc(list , 4\*sizeof(int));

        if(list == NULL){

           return 1 ;

         }

         list[3]=4;

      for (int i=0 ;i<4;i++){

        printf("%i \n",list[i]);

      }

      free(list);

 }

***Linked List***

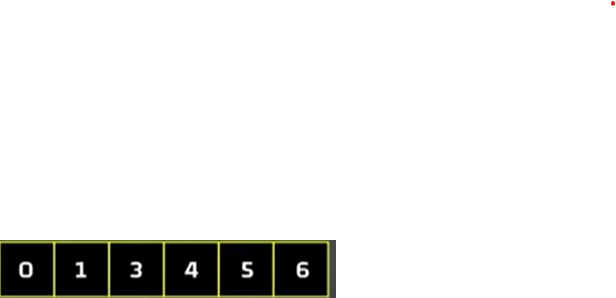
Linked list is a method for saving / storing the data like array ,

**Qu** : Why we searched for onther way to save data , know that array do the same function ?

**Answear :**  Array has many problem , and linked list comes with solution for these problems .

As we know array save data in a sequential manner and this as it is a feature it is also a flaw , when data is saved by a sequential manner here we will have a difficulty in adding or deleting an element in this array because if you delete an element from the mediem of the array you should move all element in other place , + resizing array problem .

**

**

*Here i tried to add an element (3) to an array :*

#include <stdio.h>

 #include <stdlib.h>

 int main (void){

    int\* list = malloc(6\*sizeof(int));

    list[0]=0;

    list[1]=1;

    list[2]=2;

    list[3]=4;

    list[4]=5;

    list[5]=6;

     for(int i = 0 ; i<6 ; i++){

             printf("%i",list[i]);

         }

         printf("\n");

        int\* tmp = malloc(7\*sizeof(int));

           for (int i=0 ; i<3 ; i++){

              tmp[i]=list[i];

           }

              tmp[3]=3;

         for(int i = 3 ; i<6 ; i++){

Result :

**0 1 2 4 5 6**

**0 1 2 3 4 5 6**

            tmp[i+1]=list[i];

         }

           list = tmp ;

     for(int i = 0 ; i<7 ; i++){

             printf("%i",list[i]);

         }

            printf("\n");

 }

*Here i tried to delete an element (2) from ther array :*

#include <stdio.h>

 #include <stdlib.h>

 int main (void){

      int\* list = malloc(6\*sizeof(int));

      if (list==NULL){

         return 1 ;

      }

    list[0]=0;

    list[1]=1;

    list[2]=2;

    list[3]=2;

    list[4]=3;

    list[5]=4;

            for (int i = 0 ; i<6 ; i++){

                printf("%i ",list[i]);

            }

             printf("\n");

            for (int i = 3 ; i<5 ; i++){

               list [i]=list[i+1];

            }

              list[5]='\0';

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

Result :

0 1 2 2 3 4

0 1 2 3 4

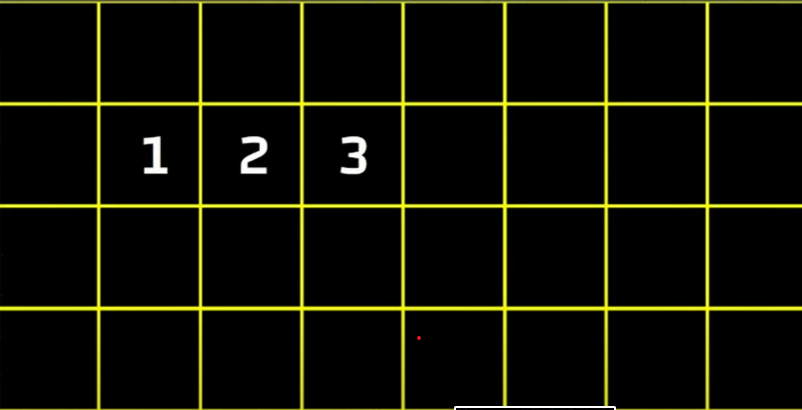
                printf("%i ",list[i]);

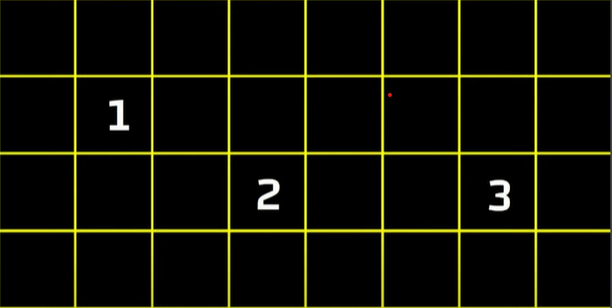
            }

           printf("\n");

         free(list);

 }

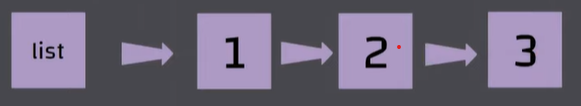
**So in the past the scientist meet , and put the problem on the table , and ask : where is the problem , and the aswear that array sotres data by a sequential manner , so simply the solution was in sotring data by not squential manner , but here there is no link between the element not like array , in array the first index is linked with the second the vice versa . **

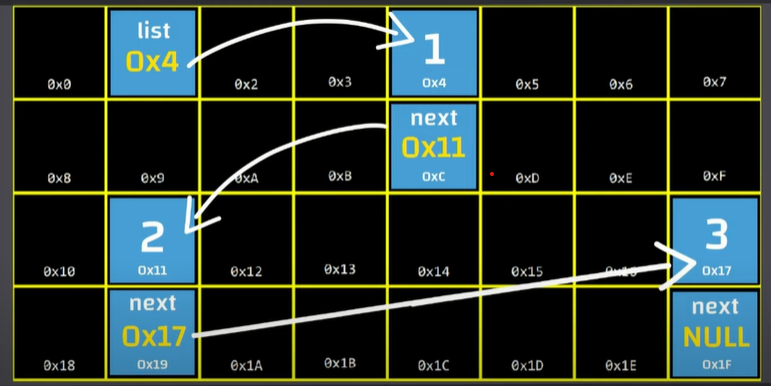
****So here the scientist decide to store data with a random manner , but there is no link between , and here pointer idea came , so we will link elemnt using pointers

Une image contenant texte

Description générée automatiquementThe element linked using pointers

Know we will start explaining linked list basics :

Imagine with that you want you store 1 2 3 in memory using linked list : the first thing you have to do a pointer that point the first number , and the first number point on the second the the scond point on the third , and the third point null because it is the last element in the linked list

So simply all elemnt contain two part , the data (number) and the pointer wich point the next elemnt , let’s see how this look in memory , as we said the element of linked list don’t sotre by a sequential manner , and linked list elemnt are linked using pointer . 

Note : without begging number you can’t access on linked list element ,and this point indicate the first of linked list .

As we said all element contains two parts , data and the pointer this collection named **Node** .

So this sound good but how can i do that with code ?

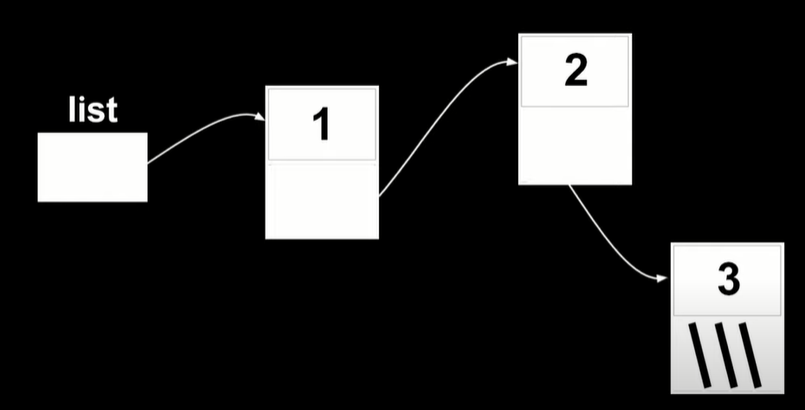
So here i will give a Qu : which thing in c that can group many data types in one element ,

the ANSWEAR : simply we will use Struct .

* **Know we have the big is storing 1 2 3 in a linked list .**

*So the first thing for making a linked list is the begging point that a pointer that point the first node , and the node contain two parts , data and the pointer that point the next node , the last element pointer point null .*

*This Is linked list Combination :*



Make in mind that every node that you will create you will do it in heap section , so we will use malloc and free functions

Know we will do a data type that contain a number and a pointer that point next node .

typedef struct node{

  int number ;

  struct node\* next ;

}node;

AS we said many times that the first thing is the begging point , and this begging point is pointer that pointer the first node , and since we don’t create node yet so we make pointer point null

node\* list  = NULL ;



know we create the begging point

let’s create the first element or the first node .

AS we say we have to create node in heap section , so we will use malloc that reseve a place and save it addres the var .

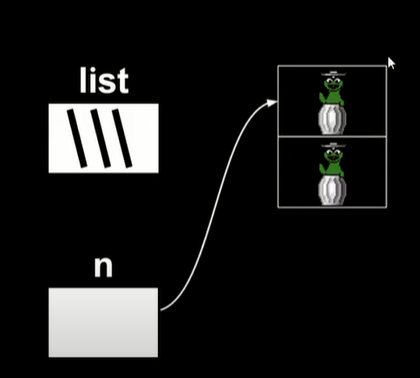
int main (void){

       node\* list  = NULL ;

       node\* n = malloc(sizeof(node));

 }

As we say in malloc round bracket we write the size so inesead of writing it manualy we use sizeof(node) , and we know that int take 4 byte and address take 8 bytes .



Know I will do two thing :

***1- making list point on node***

***2- putting values in the node***

Let’s adding values in the node , as we know n is a pointer that point the first node , so we will enter it the enter number

\*n.number=1 ; but this method is not good and know it make an error during the compiling , we use n->number = 1 ;

So The arrow -> is an abbrivation \* .

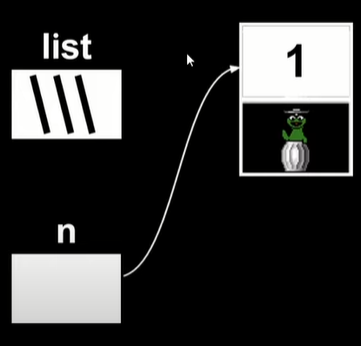
Beffore arrow we write an address a pointer

int main (void){

       node\* list  = NULL ;

       node \*n = malloc(sizeof(node));

       n -> number = 1 ;

 }

And know we will fill the second place by a pointer wich point NULL because it is the last element in linked list .

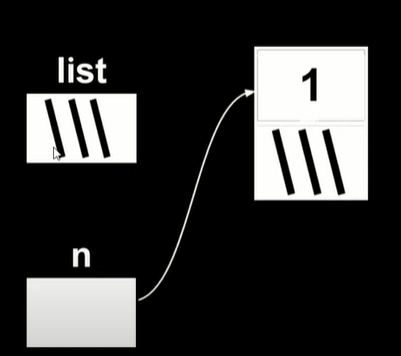
int main (void){

       node\* list  = NULL ;

       node \*n = malloc(sizeof(node));

       n -> number = 1 ;

       n -> next = NULL ;

 }

And know we will join the firt node to the linked list so we have to make list point on the first node , so it will have the save address that n have so list and n will point the same place so :

int main (void){

       node\* list  = NULL ;

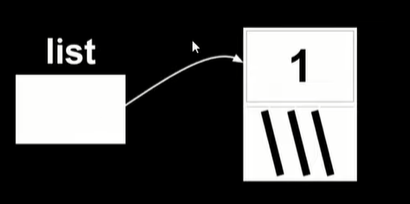
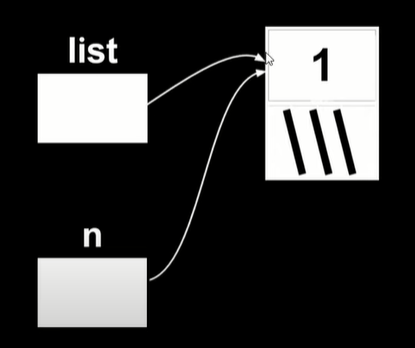
       node \*n = malloc(sizeof(node));

       n -> number = 1 ;

       n -> next = NULL ;

       list = n  ;

 }

Very good , but know i have a small fault . I you malloc and I forget the condition if malloc doesn’t find a place in memory .

node \*n = malloc(sizeof(node));

       if (n==NULL){

        return 1 ;

       }

Know we have to create the second node ,

n = malloc(sizeof(node));

if (n==NULL){

free(list)

        return 1 ;

       }

But here if the programm walk out , so here we will quit the programm without free n and as we know list = n so free(n) = free(list) ;

I think that you want to ask me two questions ,

1- why we use n || 2 – why we don’t declare n

We use n because we don’t need it know so we can use here instead of creating a new node point + n is decleard so we don’t have declare it the next time , so we can use n because it’ s

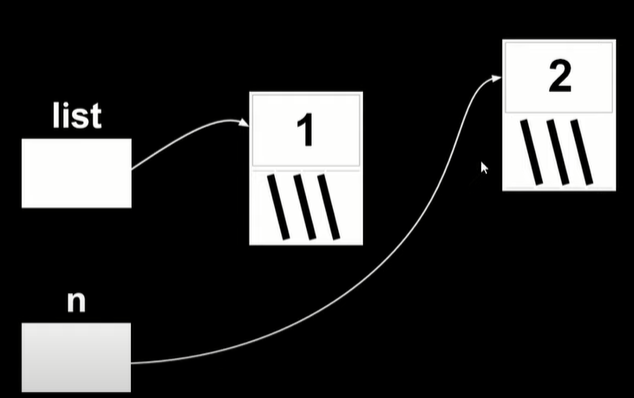
It’s a node pointer + it’s declared

 n = malloc(sizeof(node));

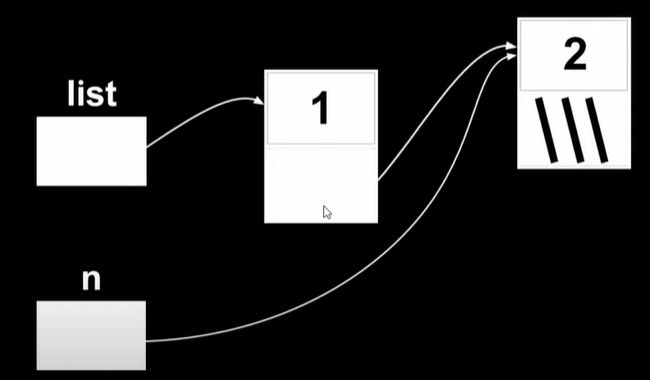
       n -> number =  2 ;

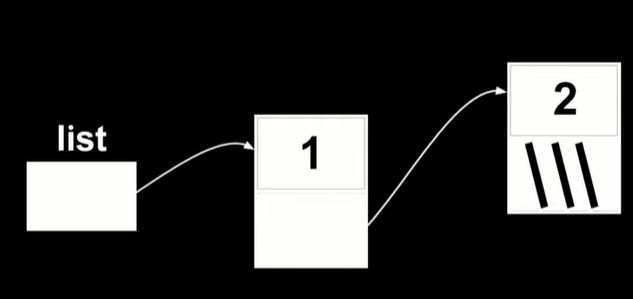
       n -> next = NULL ;

so know i fill the second node , as the img show :



Know i want to make the first node point to the second node , so we will make first node next point on the same place that n point ,

       list -> next = n ;

and know we don’t nedd « n » : 

Know let ‘s create the third node :

  n = malloc(sizeof(node));

       if (n ==NULL ){

         free(list -> next);

         free(list);

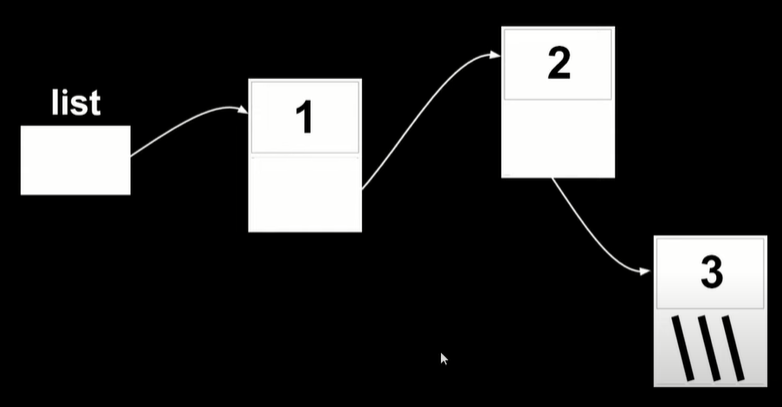
         return 1 ;

       }

       n -> number = 3 ;

       n -> next  = NULL;

       list -> next -> next = n ;



Now , focus you attention on «  if statement «  imagine with me that we write :

       if (n ==NULL ){

         free(list);

         free(list -> next);

         return 1 ;

       }

First thing you free list which mean the node that list point on (the first node ) , but know you free the first node how can you free the second node and you free the mediator bettwen list and the second node bacause we access to the nodes using the head point or the begging point , so the right manner is free the second node then the first node .

***Know i have a big mission is printing all the number which are in the linked list :***

 printf("%i  ", list -> number );

         printf("%i   ",list -> next -> number  );

         printf("%i   ",   list -> next -> next -> number );

no no don’t use this method imagine with me that you have 10000 element so this method is absoloutly not good , so we will use For loop :

 for ( node\* tmp = list ;  tmp  !=  NULL  ;    tmp = tmp->next ){

               printf("%i  ",  tmp ->number );

        }

As we know for loop contain 3 places , the first place for start counter and the second for condition and third for steps ,

!!تمعن الكود

We don’t forgite for free the blocks because we use malloc

  free(list -> next -> next );

               free(list ->  next );

               free(list );

This method isn’t good imagine that you want to free 1000 elements , so we have to use while loop :

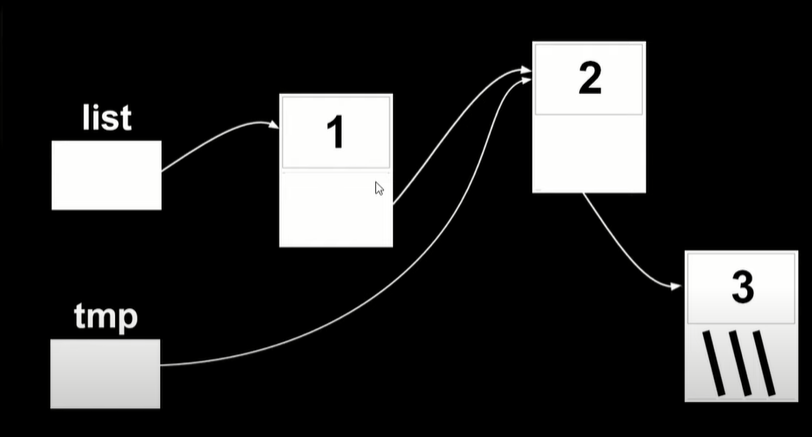
            while(list != NULL){

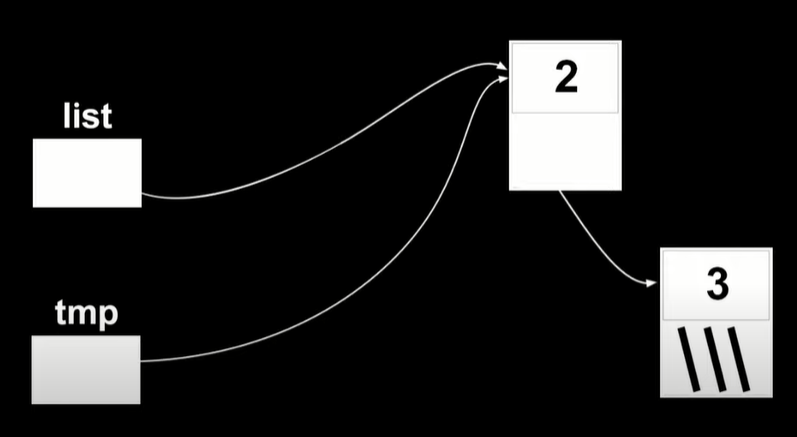
             node\* tmp = list -> next ;

             free(list );

             list = tmp ;

            }



Know we will talk about resizing , yes it is the problem that make us search for another data structure and not array .

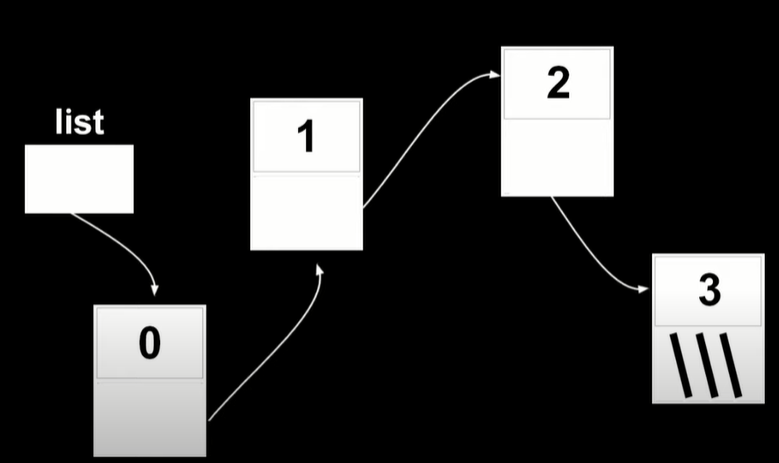
Know we will try to add ‘’ 0 ‘’ at the begging of the list :

  n = malloc(sizeof(node));

            if (n != NULL){

            n -> number = 0 ;

            n -> next = list ;

            list = n ;}

***so At the end the code is :***

#include <stdio.h>

 #include <stdlib.h>

typedef struct node{

  int number ;

  struct node\* next ;

}node;

 int main (void){

       node\* list  = NULL ;

       node\* n = malloc(sizeof(node));

       n -> number = 1 ;

       n -> next = NULL ;

       list = n  ;

       n = malloc(sizeof(node));

       n -> number =  2 ;

       n -> next = NULL ;

       list -> next = n ;

       n = malloc(sizeof(node));

       if (n ==NULL ){

          free(list -> next);

          free(list);

         return 1 ;

       }

       n -> number = 3 ;

       n -> next  = NULL;

       list -> next -> next = n ;

            n = malloc(sizeof(node));

            if (n != NULL){

            n -> number = 0 ;

            n -> next = list ;

            list = n ;}

   for ( node\* tmp = list ;  tmp  !=  NULL  ;    tmp = tmp->next ){

               printf("%i  ",  tmp ->number );

        }

               printf("\n");

            while(list != NULL){

             node\* tmp = list -> next ;

             free(list );

             list = tmp ;

            }

 }