

Lecture 07

Lecture Contents:

- 1-pointers and structures/ classes
- 2-linked list

اتكلمنا المرة اللي فاتت عن ال pointers عشان عايزين ناخذ ال dynamic list بس دى معتمدة على ال pointers فقلنا هنراجع عليهم و كنا بدأنا مراجعة فعلا المرة اللي فاتت و فهمنا كمان حاجة جديدة اسمها ال heap بس كان ناقص 2 وبنقى خلصنا : مراجعة ال pointers



Pointers and Structures/Classes

12

```
main() {
    struct Test{
        int i;
        char c;
    }
    Test *tp; //defines pointer to Test struct
    Test t;
    tp = new Test;
    assert (to !=NULL);
    tp->i = 10//Call member function using arrow -> operator
    tp->c='v';

    *tp = t;
    delete tp;
}
```

فى الأول عرفت جواه 2 variables وبعدها عرفت pointer لل structure وعرفنا انه بردو .
tp=new Test;

يا heap عايزة مكان يكفيني .

assert(to !=NULL);

وهنا ال heap يرد بفلى مكان يكفى وللا لا .. هنا اكيد فى زى ما فلانا قبل كده .. لكن فى البرامج الكبيرة ممكن يبقى فى وممكن لا.

tp->i=10;

tp->c='v';

السطرين زى ما اكون بقوله:

(*tp).i=10;

(*tp).c='v';

اتفقنا المرة اللي فاتت انى لما احط ال * قبل اسم ال pointer ومش فى ال declaration بتاعه يعني بقوله حط جوة ال address

اللى بيشاور عليه ال value pointer كزة. وده معنى بردو الجملتين اللي بالسهم فوق

ونلاحظ ان السطرين اللي فيهما ال arrow tp من غير * .. هي بتعمل كده فى ال structure او ال classes.

طالما ده اقدر استخدم اى طريقة من الاثنين. وطريقة ال arrow خد بالك لازم ال - و ال <> بيقوا لازفين فى بعض .. من غير space بينهم.



Pointers and Structures/Classes

11

```
main() {
    Complex x;
    Complex *cp; //defines pointer to Complex class

    cp = new Complex(10.6,4.3); // allocate a complex ADT calling constructor 2
    assert (cp !=NULL);
    cp->ReadComplex(); //Call member function using arrow -> operator
    x.ReadComplex();
    cp->Add(x);
    //OR
    (*cp).Add(x);
    delete cp;

    cp = &x; // possible cp still alive
    cp -> Add(x); //No problem
}
```

pi = new Complex[2000];

تاني سطر ده بنعرف pointer من نوع complex وبعد كده مفيش حاجة جديدة عن اللي قلناه في ال slide اللي فوق غير ان
cp=new Complex(10.6,4.3);

هن call constructor اللي كان عملناه المحاضرات اللي فاتت اللي بياخد 2 arguments .
وممكن لو عايزين نعمل multiply مثلا .. نعمل كده

cp→multiply(c);

زى ما قلنا قبل كده السهم ده يعني هات ال contents بتاعت ال pointer

واخر سطرين فى الكود .. الجديد اللي هنا ان لما بتعمل delete compiler مش بيعرض .. بس هيقى بياخد garbage

نقدر كده نبدأ ال dynamic list . هي كانت ايه مشكلة ال static list ؟
fixed size.1
بس ممكن حلها بكم:

```
list::list(int size)
{
    listArray=new listElemType[size];
    assert(ListArray!=NULL);
}
```

كده بنعمل كمان 500 different sizes .. مبقاش خلاص .. بس فى عيب صغير وهو انى مثلا عملته 1000
وبعدها احتجت كمان

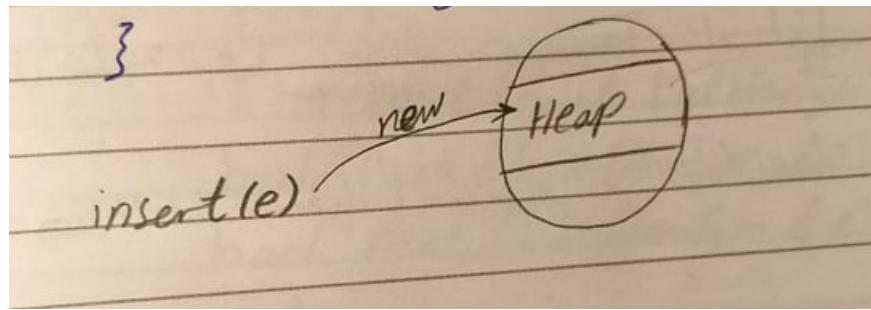
linked list ودى هتحلل بال

لو تفتقربى كمان مشكلتين .. اللي هما انى لو حبيت insert حاجة بس مش فى الاخر .. لا فى مكان وسط ال list .. فكنا بنضطر
نزويد فى الآخر و نقدر نشففت لحد المكان اللي عايزينه .. وتانى مشكلة هي ال delete انى بردو بعمل delete حاجة فى نفس ال
list وبضطر اشففت عشان المكان ده ميتبسش فاضى. والمشكلتين دول بالطريقة دى بياخدوا وقت كبير جدا.

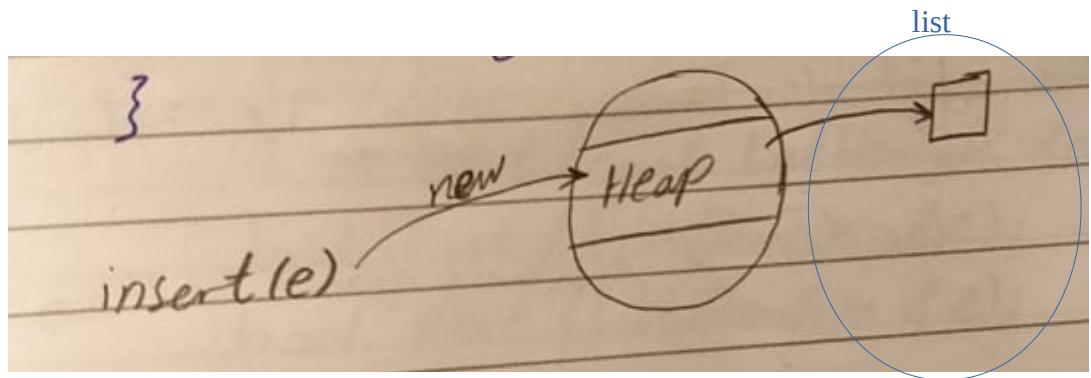
عشان كده تعالوا بقى نشوف فكرة ال dynamic list
؟heap
فاكرین ال
انا مثلا لما اكتب

insert(e);

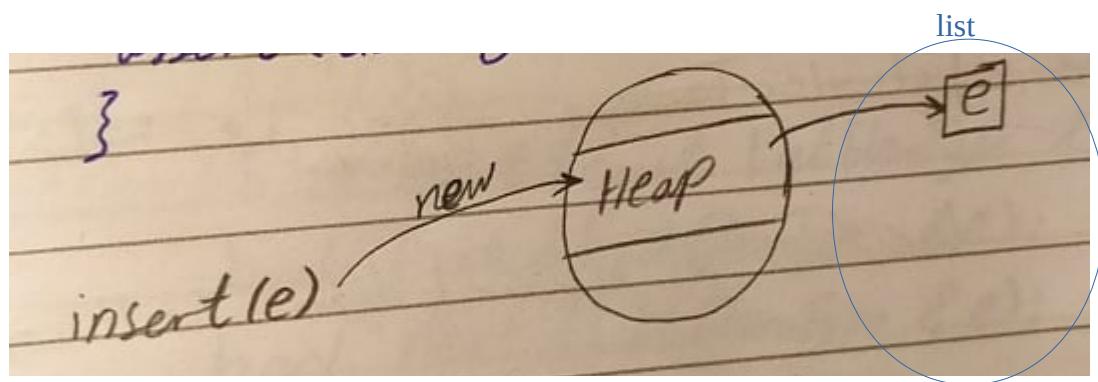
هروح اقول لل heap انا عايز مكان



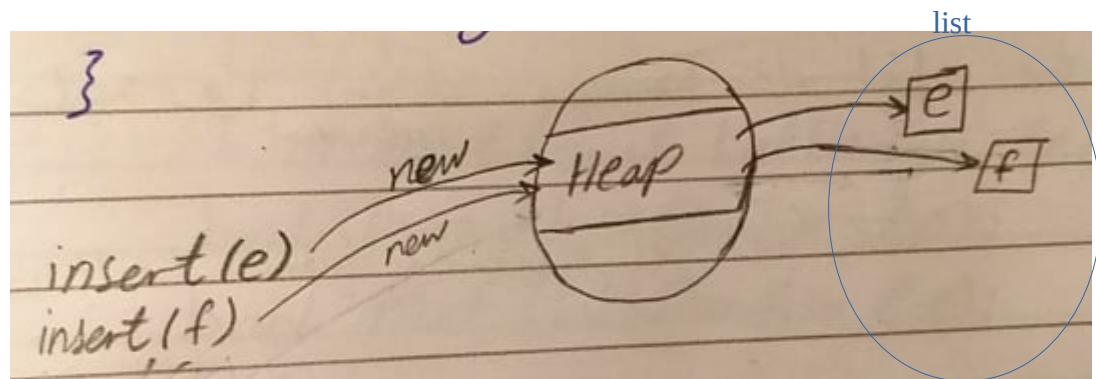
البيضة دى المفروض انها ال memory كلها يعني D .. وده جزء ال heap فيها.. الدكتور رسمه كده برضو D .. المهم يعني بعد ما طلبت مكان ال heap هنتبني مكان



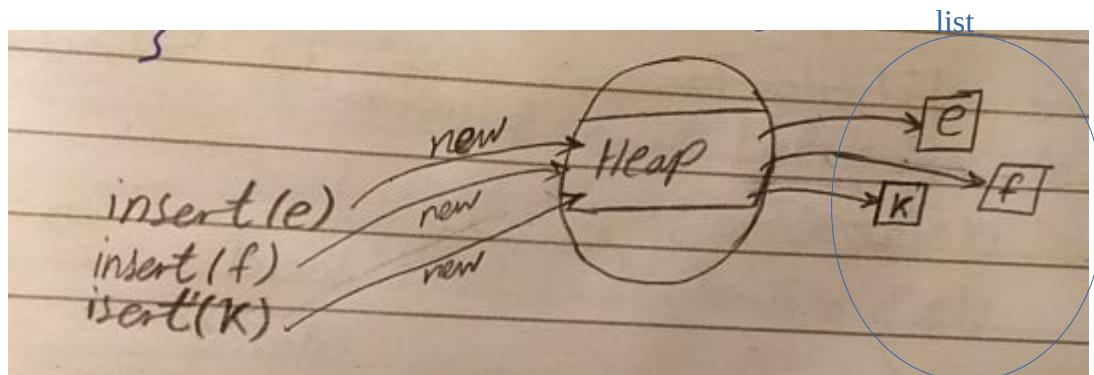
وهحط فيه ال e



طب نعمل insert تانى كده.



كمان insert معلش : ”D



ايه بقى العلاقة بين الحاجات دى وبعض؟ يعني كده ايه ترتيبهم؟ هى ال new بترجع address مثلاً؟
لا

طيب ايه الحل؟

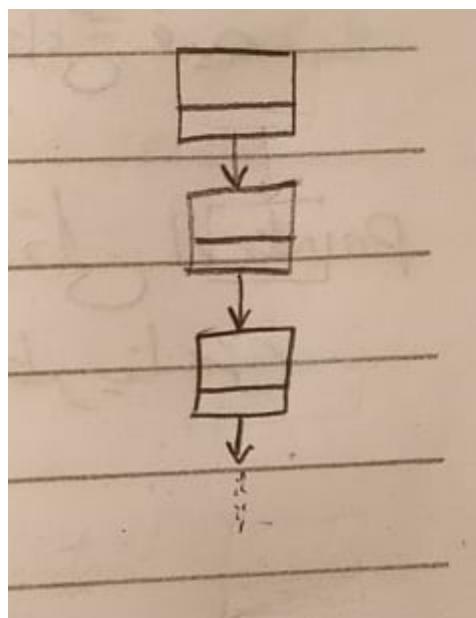
اعمل array شايل الاماكن ورا بعض .. ممكن بس كده رجعنا لمشكلة ال array تانى مشاكله اللي بنهرب منها بال list.

طب ايه الحل؟؟

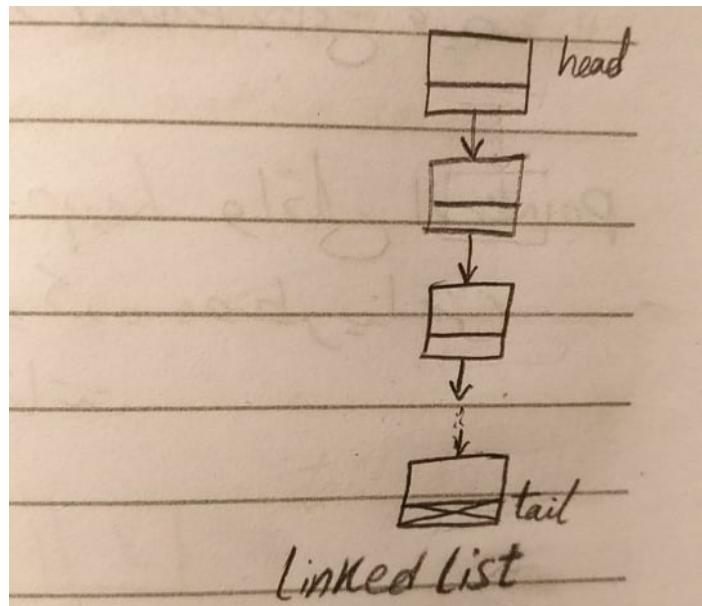
فاكر فى ابتدائى .. الطابور اللي كنا بنعمله واحداً رايحين اوضة الرسم مثلاً وللانازلين فى حصة الالعاب .. اول واحد يقف واللى
بيجي بعده يحط ايده على كتف اللي قدامه وهكذا لحد اخر واحد .. هي الفكرة دى الحل.
ان كل مكان بيصل على اللي بعده .. ده باستخدام pointer

يعنى ايه بردو؟

يعنى كل مكان هاده من ال heap هحط فيه ال element اللي عايز احطه ومعاه pointer بيشار على المكان اللي بعده.



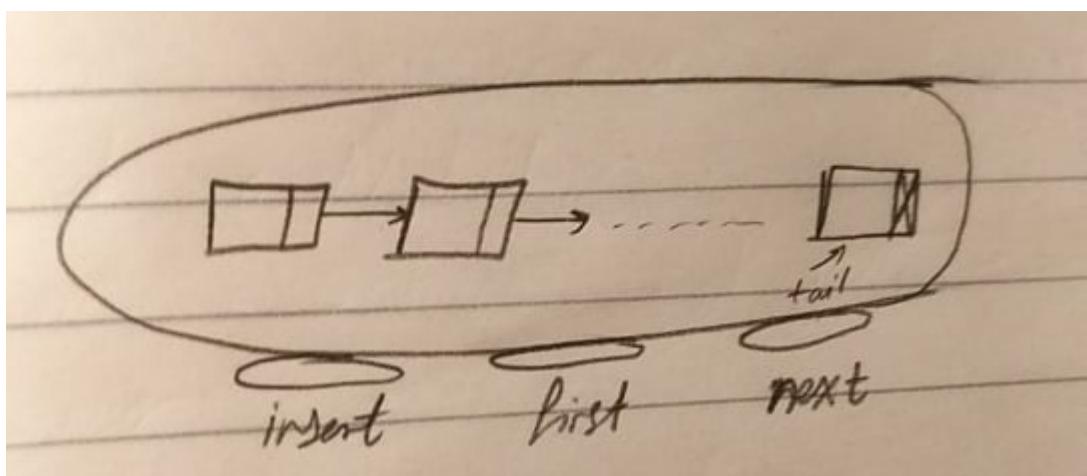
لحد اخر مكان وهىقى اسمه tail عشان مش هىقى بيشار على حاجة ما هو الاخير
واول اسمه element عشان مفيش حاجة بتشار عليه.



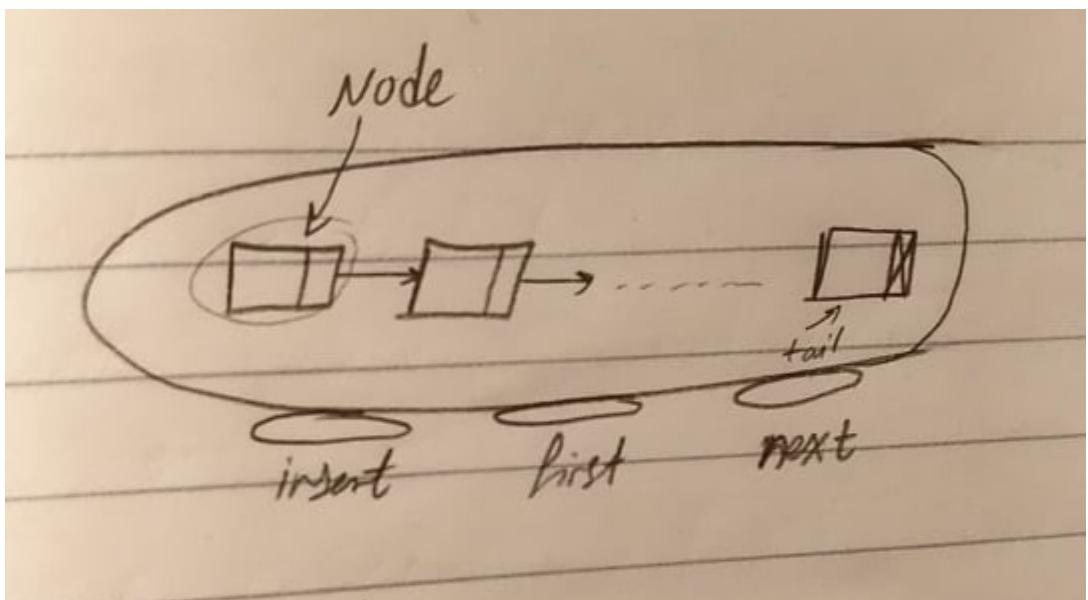
ودى اسمها linked list . ال stack كمان ما هو الا linked list بس بقدر اشوف فيه اول element بس . لكن هنا هنقدر نعمل insert و delete ونشوف اي element عايزينه مش بس الأول .

وفي حاجة كمان اسمها queue نقدر نشوف اول element واخر element هنأخذ كل ده في الكورس قدام .. ماعدا ال graph . وفي انواع تانية زي ال tree binary وال graph .

نشوف بقى ال dynamic list باستخدام ال linked list .

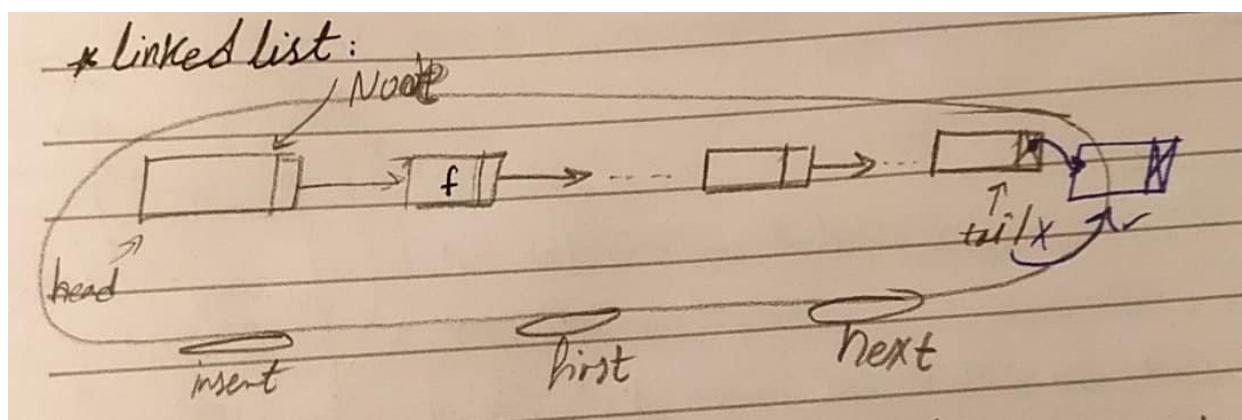


ده شكلها و ليها نفس ال interfaces كل implementation هيخالف زي ما هنشوف دلوقتى .. وهنشوف هنا كمان قوة ال abstraction ان ده نفس ال interface insert و ال details نفس الكلام .. بس ال مختلف

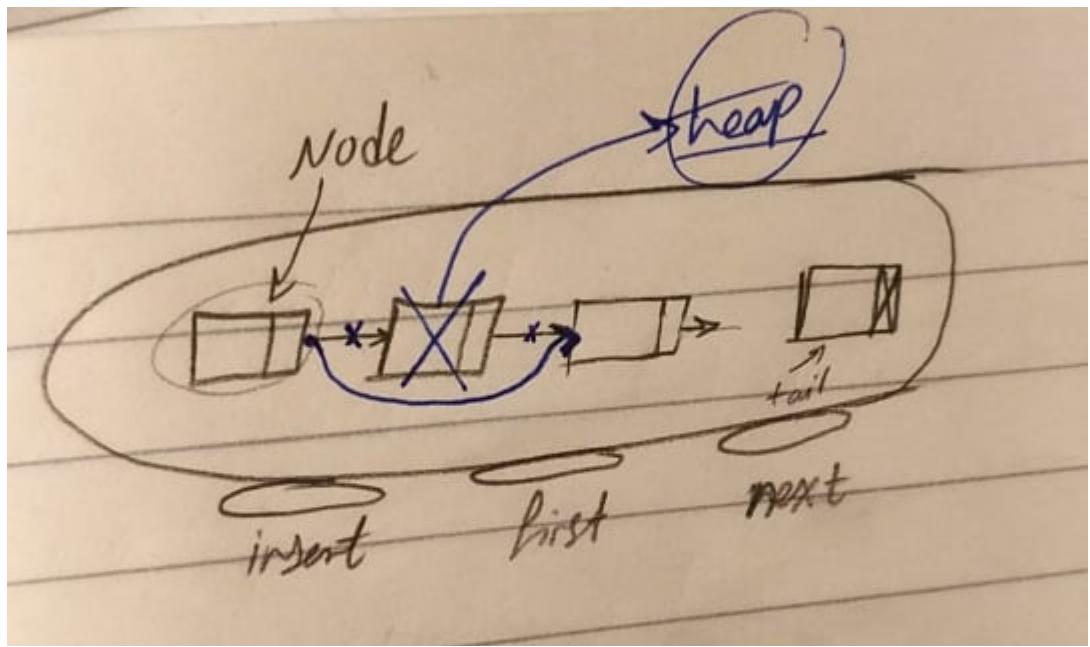


كل مكان من دول اسمه node
الnode دى structure فيها ال element عشان فيها ال element اللي بخزنه وال pointer اللي بি�شاور على ال element اللي بعده.

فمبدياً insert ازاي؟
هقول لل heap اديني حته .. هخزن فيها ال element و هخلن tail pointer بيشاور عليه و يبقى ال element الجديد هو ال tail.

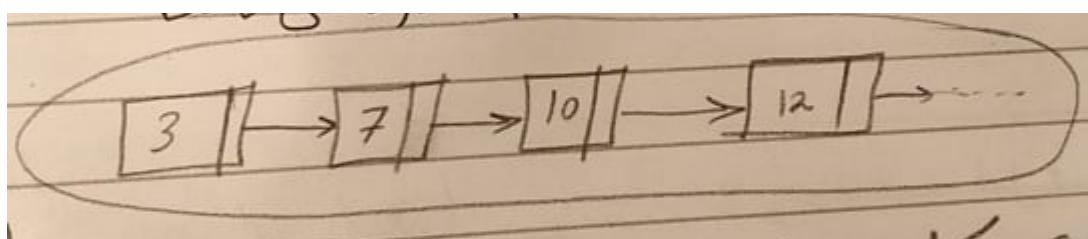


ولو عايز اعمل delete .. هروح ارجع المكان لل heap و اخلن pointer اللي قبله مش بيشاور عليه لا .. بيشاور على المكان اللي بعد ال node اللي مسحتها وبكده حلينا مشكلة الوقت اللى كان بيتأخد فى تشفيف ال elements



طب ومشكلة ال ordered list ??

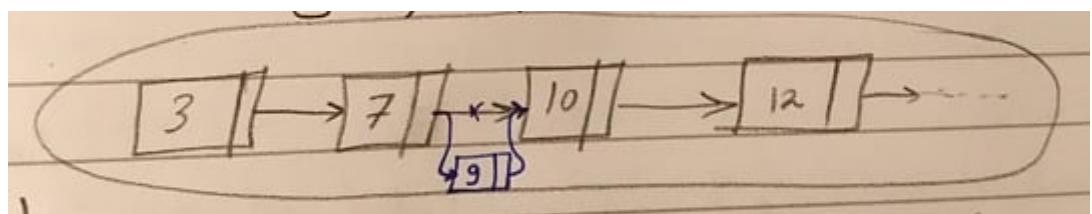
لو عندي ال list كده:



ومكتوب

insert(9);

هقدر اقaren لحد ما اعرف المكان الصح لـ 9 اللي هو هنا بين الـ 7 والـ 10 واخلى pointer المكان اللي المفروض يكون قبلها يشاور عليها و اخلى pointer الـ 9 يشاور على مكان اللي بعدها .



فكده كل المشاكل الحمد لله اتحلت .

طب لو عايز اعمل ال circular list ? كل اللي هعمله انى اخلى ال tail يشاور على ال head بس D :

نكتب بقى الكود :

List.h

```
typedef <...> ListElemType;
class List{
public:
    List();
    bool insert(const ListElemType &e);
    bool first(ListElemType &e);
    bool next(ListElemType &e);
//حد هنا کلام قدیم..
```

کده عملنا ال structure node الى قلنا الى عليه فوق .

از اى فى تعريف لـ node pointer الى احنا اصلاً لسة بنعرفها؟

الحقيقة ان ال pointer عادى نعمل معاه كده .. ده مجرد بيشاور على address .. لكن لو كنت كتبت كده /*

Node next;

* کان هيدى error /
فيه طريقة تانية نكتب بيها الكود */

```
struct Node; // forward declaration
typedef Node* link;
struct Node{
    ListElemType item;
    link next;
};
```

الدكتور قال ان الطريقة الثانية دى احسن .. يعني making sense اكتر واثيک وواضحة اكتر .

دلوقتى بقى هعرف شوية حاجات هحتاجها : /*

```
link head; //ده عشان ال first
link tail; //دى عشان ال insert
link current; //دى عشان اعمل next
}
```

کدة زي الف .. نكتب بقى ال : List.cpp

List.cpp:

```
List ::List () { //initialize all links with null values
    head = NULL;
    tail = NULL;
    current = NULL;
}
```

```
bool List::insert (const ListElemType &e) {
```

/* في الاول بس قبل ما نكتب الكود محتاجين فكر في ال cases المختلفة اللي ممكن تحصل عشان حاول بقدر الإمكان اننا مانتقاجئش ب logical errors في ال runtime فعايزين نعمل حسابنا عليها من الاول خالص .. ففكر في اول case اللي هي ال general case

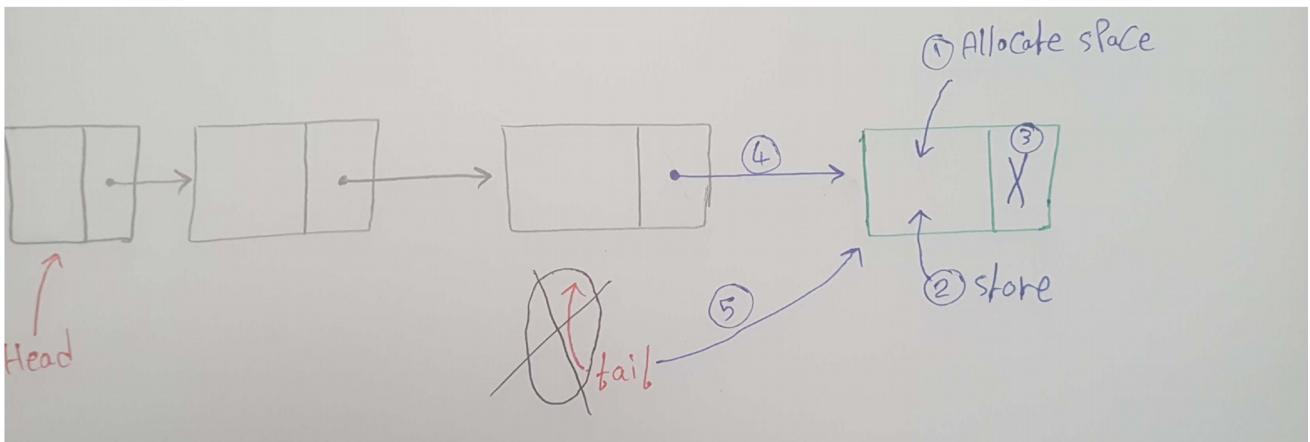
General case: insert new item in an existing list

يعني معانا list موجودة جاهزة واحنا عايزين نزود عليه node جديدة ... قول مثلاً دي ال linked list اللي موجودة ناو :

واحداً عزيزٍ نزودُ عليها node جديدة .. تفتكر هنعمل ايه؟؟

محاتجين نعمل 5 خطوات .. اول حاجة نAllocate مكان جديد في ال heap لل node الجديدة دي .. بعد كدة نحط فيها ال
الي احنا عايزينها (اللي هي هنا هتنقى e) .. بعد كدة هنفكر بقى ازاي نربط ال node دي بال linked list الموجودة من غير
مانبظر حاجة فيها، احنا عايزين نخلي ال node دي اخر element فهنجلي ال next بتاعها يشاور على null .. رابع خطوة
انتا نربط ال node الجديدة دي باللي قبلها، فهنجلي ال next بتاعت ال node اللي قبلها (اللي هي اسمها null) تبقى بتشاور على
ال node الجديدة .. خامس حاجة انتا نقول لل node الجديدة دي بقى اخر element بدل ال
node اللي قبلها (اللي هي كانت الاخيره قبل ما نزود ال node الجديدة)
في يعني من الآخر الخطوات بتاعتنا هتبقي كدة بالترتيب:

- 1- Allocate memory space in heap
 - 2- Store e
 - 3- Assign its “next” pointer to null
 - 4- Assign the previous “next” pointer to the new node
 - 5- Assign the “tail” pointer to the new node



طيب دي كدة كانت اول case .. نشوف بقى case تانية، مثلاً بيقى لسة مفيش اي node موجودة في ال linked list، وبالتالي لما نعمل insert لـ node جديدة هتبقى هي اول node واخر node في نفس الوقت :
في ال case دي هنعمل نفس اللي عملناه في ال general case تقريباً، الفرق بس اننا هنخلي ال head & tail يشاوروا على ال node اللي هتتحط جديدة :

1- Allocate memory space in heap

2- Store e

3- Assign its “next” pointer to null

4- Assign the “head” pointer to the new node

5- Assign the “tail” pointer to the new node

نكمي الكود بقى: /*

```

addedNode = new Node; // (1)
if (addedNode == NULL){ // just a check
    return false;
}
else {
    addedNode → item = e; // (2)
    addedNode → next = NULL; // as it's the last node (3)
    if (head == NULL) {
        head = addedNode; // (4)
    }
    else {
        tail → next = addedNode;
    }
    tail = addedNode; // (5)
}
return true;
} // terminate the if condition

```

```
} //terminate the “insert” function
```

مشكلة ال dynamic allocation انه يحتاج تركيز كبير اوي من ال programmer وهو شغال .. يعني بيقى واخد باله كويس اوي من order instructions اللي بيعملها .. يعني مثلا في ال general case لو انا جيت عملت (4) قبل (5) بيقى كدة هيحصل مصيبة ..

فمن الآخر عشان نبعد عن المشكلة بتاعت الترتيب دي لما نبقى نـ delete او نـ modify بيقى في الاول نعمل link بين ال existing node وال new node وبعدين نـ modify اللي احنا عايزين نعمله

طبعا من الكلام اللي فوق دة اكيد عرفنا ان مهم جدا جدا جدا اني لما افكر اكتب كود dynamic allocation بيقى اعمل ديزاين الاول وبعد كدة اكتب الكود، وده عشان ابقى عارف كويس انا بعمل ايه

نكتب 2 كمان على السريع /* functions

```
book List::first (ListElemType &e){
```

//المفروض ان ال condition يكون فيه على الاقل 1 element في ال list

```
    if (head == NULL){
```

```
        return false;
```

```
    }
```

```
    else {
```

```
        current = head;
```

```
        e= current → item;
```

```
        return true;
```

```
    }
```

```
bool List::next (ListElemType &e) {
```

//لو ال list فيها first او لو انا واقف عند اخر مكان وما عنديش next بيقى مش هيفتح نكمel ال function

```
if (current == NULL || current == tail) { // we could've said that : current ==NULL || current → next
```

```
                                == NULL
```

```
        return false;
```

```
}
```

```
else {
```

```
        current = current → next;
```

```
        e = current → element;
```

```
        return true;
```

```
}
```

```
}
```

كدة خلصنا الحمد لله بس فيه exercise الدكتور قال نمشي ايدينا فيه .. نعمل insertion في اول ال list، وهذا بقى مش هحتاج نعمل tail .. فعايزين برضه نجاوب على سؤال ليه مش هحتاج نعملها