

Lecture 9

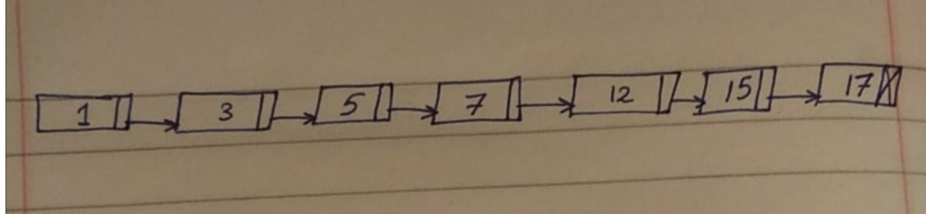
contents:

-doubly linked list.

-template.

خذنا ال linear list .. وال circular list المحاضرات اللي فاتت.. المحاضرة دي هناخد ال doubly linked list. ايه دي بقي؟

دي زي ال list العادية اللي خدناها قبل كده .. بس الفرق ان كل node ليها 2 pointers.



ليه؟

تعالى ناخذ مثال

مثلا في ال list دي انا عايز ادور على رقم معين

search(12);

هنعمل loop تدور في node .. node صح؟

تفكر ال logic هيبقى ازاي؟

هيبقى باني اعمل first واقارن ال value .. وبعدها نبدأ ال loop وجواها نعمل next ونقارن ال value لحد ما نلاقها.

لما هنلاقي ال 12 هخرج من ال loop.

لما هخرج قيم ال pointers ايه اخباها؟

ال current بقي عند ال 12 ... طيب افرض قلت بعد ال statement اللي فوق دي

search(15);

هنا هدخل ال loop اللي بتعمل next وتقارن بس ال current عند 12 فكداه next واحدة وهبقى وصلت لل 15 .. يعني

مبدأناش من الأول .. فكداه احسن . عشان لو بدأت من الأول هتاخذ وقت اطول و اطول جدا لو ال list كبيرة.

طيب لو كان جزء ال search في الكود كده

search(15);

search(5);

هعمل ايه؟؟

مش احسن حل هو اني زى ما طلعت قدام اجيب ال 15 المرة اللي فاتت .. ارجع ورا المرة دي عشان اجيب ال 5.

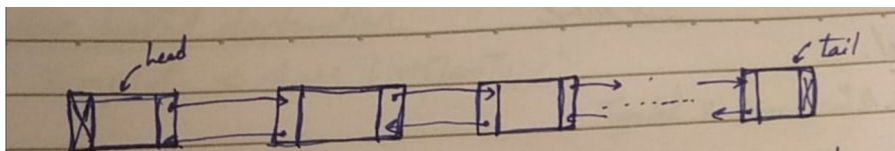
بس انا معنديش حاجة ترجعني ورا؟! يعني في المثال اللي فات انا عندي next اطلع بيها قدام .. لكن هنا معنديش حاجة ترجعني

ورا؟

عارف لو كان array كنا عملنا decrement بسهولة لرقم الخانة و قرينا .. لكن دي list امكانها من ال heap.

فعشان نرجع بسهولة فكرنا نعمل pointer لورا عكس بتاع ال next. ودي فكرة ال doubly linked list.

فزي ما شايفين في الصورة .. كل node ليها 2 pointers واحد بيشاو على ال node اللي بعدها و واحد بيشاو على ال node اللي قبلها.



ثانية واحدة.. طب مانا ممكن اعملها بال list العادية ام pointer واحد؟

اه دي حقيقة .. ممكن فعلا اعمل loop على ال nodes من أول node واسأل هل ال current بيساوي ال next→pred ولا

لا

ال condition ده في المثال بتاع الارقام بيسأل هل ال node اللي بعدى هي ال current لو اه بيبقى انا وصلت لل 7 اللي هي

قبل ال 12 اللي عندها ال current.

لو اه خلى ال $pred \rightarrow next = pred$ يعنى خلى ال `current` بدل ما يشاور على ال 12 يشاور على ال 7 وافضل اعيد فابقى كده برجع ورا من غير `doubly linked list`.

بس طبعا لو `list` كبيرة .. الكلام ده هياخد وقت كبير اوى عشان ال `loop` كبيرة.

طب وطريقة ال `doubly linked list` ايه ال `cost` اللي هدفه؟
ال `cost` هو `pointer` زيادة `per node`.

طيب ايه ال `operations` اللي اقدر اعملها فى نوع ال `list` ده؟

operations:

`insert()` `first()` `next()` `previous()`

قبل ما نكتب الكود .. نتعلم حاجة جديدة اسمها ال `template` موجودة فى ال `c++` ومش موجودة فى ال `c`

Template:

عارف الاسطمة اللي كنا بنستخدمها الرسم الهندسى .. دى حاجة شبيهها .. ازاي؟

* في ال `list` كن عني في اول ال `h file` :
`typedef ListElemType`
وكنت باحط مكان النقط هتأ ال `Data type`
بتاع ال `item` الى باخزنه .
* نصترخ اننا حطينا `int` يعني ال `list` هتأ بتحتجز كل
الاماكن ل `item` نوعه `int`
`#include "List.h"`
`int main() {`
عائز ال `list` دى . اخزن فيها ارقام بالتالي و `List list1`
او `class` هيسعمل صح ..
عائز اخزن في ال `list` دى اسماء زي `List list2`
ال `contacts` مع الموبايل . هنا هيرج يشوف ال `h file`
هتلاقى ال `typedef` مضمون ل `int` يعني مش `String`
بالتالي مش هاعرف استخدم ال `list` دى صح ..
لو غيرت ال `typedef` ل `String` اول `List` كنت عايزة
ارقم بالتالي دايم واحد هتتوط .

الحل اللي اى حد فينا لسة ميعرفش ال `template` هيفكر فيه هو ..

1) هأخذ `copy` من `List.h` و `List.cpp`
2) أغير اسمهم ل `SList.h` و `SList.cpp`
3) أخش في ال `h file` الجديد `String` ال
`typedef` الجديد يبقى `String`.

الكود بعد كده هنكتبه كده

```
#include "list.h"
#include "slist.h"
int main() {
    List l_list;
    SList s_list;
}
```

* دا هيرج لـ list.h الى فيه ال
typedef معمول int فـهيجن List
* دا هيرج لـ slist.h الى فيه ال
typedef معمول string فـهيجن SList
List لـ Strings

افرض بقى رخت معايا .. انى عايز اعمل "D: char list" .. هعمل الكلام ده كله تانى والفكرة ان هو سطر واحد اللي بغيره بعد ما اعمل copy .
طب ايه رايتك انى اعمل class واحد وواقي ابعت ال data type اللي عايز اعمل بيها ال class ده وخلاص ... باستخدام file واحد h.
ده دور ال templates

• Templates are used For defining generic classes.
• كلمة Generic معناها لا تتحدد مع ال Datatype يعني لما تشبع generic class يعني class الـ code بتاعه يتنفذ مع اى ال datatype انت بـاستعماله.
• مبدأ ال Generics دا حل لو مثلاً هاتعمل library وتنزلها على البنت لازم بتقر مغطى كل الاحتمالات (Datatypes) الى ال user ممكن يبعثها لـ class بتاعه.
• بالتالى بدل ما اعمل h/cpp Files لكل class هاتعمل class واحد بس يقوم بـدور كل ال Files دي.
• Templates eliminate the need For rewriting the code.

• Type(s) used by the class becomes a parameter to the class.
• يعني ال Datatype الى هاشتغل بيه هيقدر parameter او حاجة بـاستعمال class مع كل مرة هاتعرف class جديد.
• هتستخدم ال Templates في كل ال classes الى هتعملها بعد كدا. هاتعمل دلوقتى ال Doubly linked list باستخدام ال Templates.

نكتب ال h. بقى

template < class ListElemType>

الجملة دى كده حلت محل ال typedef

```

class DLLIST{
public:
    DLLIST();
    ~DLLIST();
    bool insert(const ListElemType &e);
    bool first(ListElemType &e);
    bool next(ListElemType &e);
private:
    struct Node;
    typedef Node *Link;
    struct Node{
        ListElemType elem;
        Link next;
        Link prev;
    }
    Link head;
    Link tail;
    Link current;
};

```

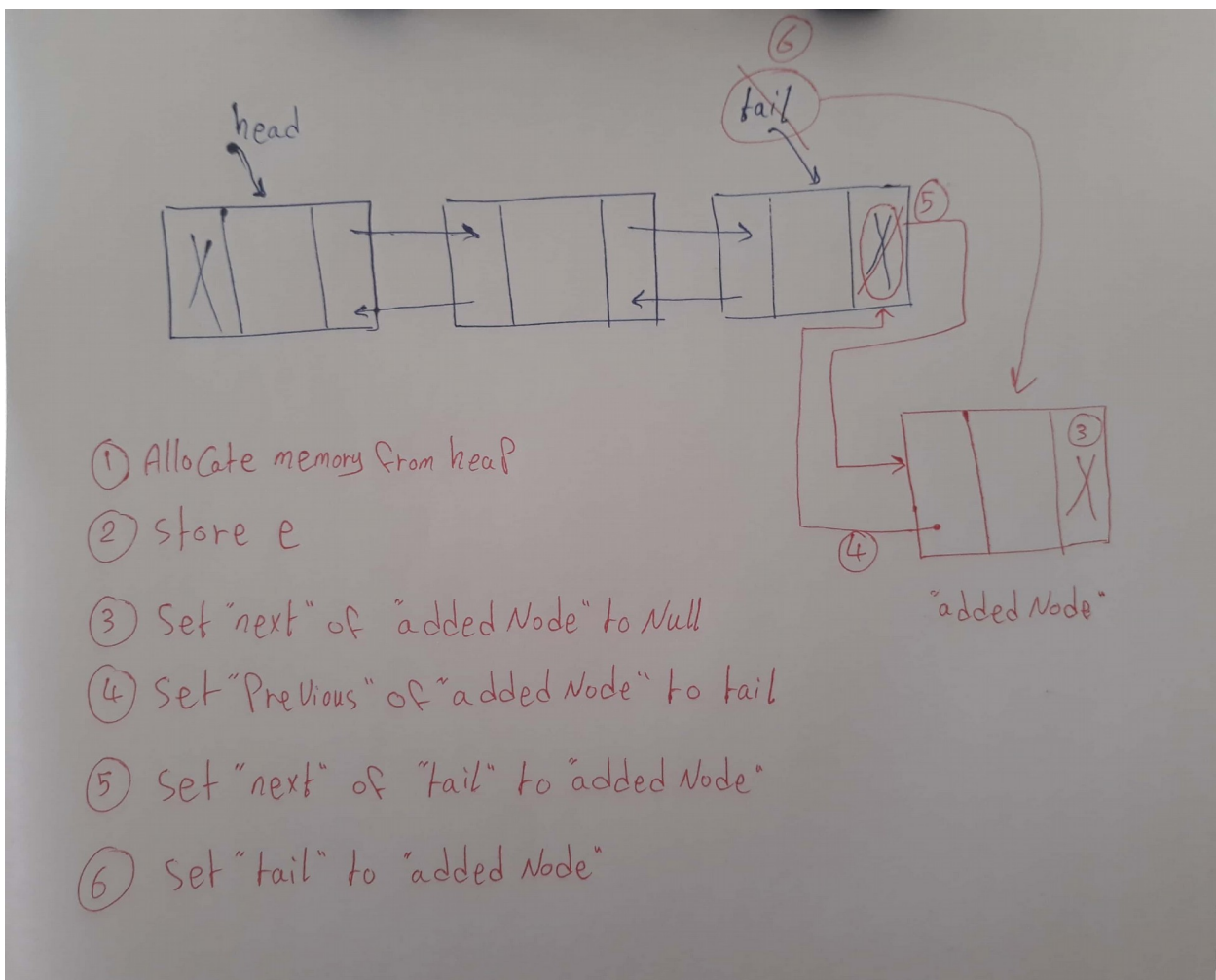
ده ال destructor .. معملنا هوش بس مهم فالدكتور قال نعمله exercise .

كل ده مفهوش حاجة جديدة غير اول سطر بتاع ال template واننا عرفنا pointer اسمه prev في ال structure ال node .
 خد بالك الدكتور قال ان في ال template لازم كل حاجة تبقى في ال h .. ليه؟ قال دور عليها على النت لو interested ...
 المهم ان اللي جاي اللي هو تعريفات ال functions كمان هتبقى في ال h مش بس اللي فوق.

نشوف بقى ازاى هند implement ال insert & previous باستخدام ال template .. بس كالعادة تعالى نديزاين الاول :

general case:

معنا list جاهزة وه insert عليها في الآخر :



وكالعادة برضه عشان نثبت ان ترتيب ال instructions مهم تعالى نلعب في الترتيب ونشوف ايه اللي هيحصل .. لو عملنا (4) قبل (3) هيحصل مشكلة؟؟ .. الاجابة لأ، هم مالهمش علاقة تربطهم ببعض فمش مشكلة مين اتعمل قبل مين طب لو عملنا (5) قبل (4)؟؟ مش مشكلة برضه طب لو (6) قبل (5)؟؟ .. لأ دي المشكلة بقي لأن كدة ال node هتلف على نفسها زي ما شوفنا في المحاضرات اللي قبل كدة .. فيعني اه الترتيب مهم بس يعني هي مش قاعدة اني لازم اعمل كذا وبعد كدة كذا وبعد كذا .. انا بس عايز ابقى عارف كويس اوي انا بعمل ايه واللي بعمله ده هيكون ايه تأثيره .. لو مفيهوش مشكلة خلاص يبقى go on

Special case: Empty list

Handwritten notes for the 'Special case: Empty list':

- ① allocate memory from heap.
- ② store e
- ③ assign "next" of "added Node" to Null
- ④ assign "Prev" of "added Node" to Null
- ⑤ set "head" to "added Node"
- ⑥ set "tail" to "added Node"

Diagram showing a linked list structure with two nodes. The first node contains '4' and the second node contains '3'. Both nodes have an 'X' in the next field, indicating they are null. Arrows labeled 'head' and 'tail' point to the first node (4).

Handwritten note: لو نسيت اعمل الخطوات دول من هيقى عندي reference افترار جعله وبالتالي يبقى كدة طائي ماعلتيش linked list ... فعاينفعش نلصهاهم

لو هنبص على الفرق اللي بين ال 2 cases اللي فوق هنلاقي ان اول 3 steps هم هم . بس 4 step شكلها كدة يقول انها مختلفة .. بس تعالى نشوف هي مختلفة فعلا ولا لأ
 في ال general case بنخلي ال previous بتاع ال node الجديدة يشاور على tail .. وفي ال special case بنخليه يشاور على NULL
 بس انا عارف ان في ال special case دي ال tail اصلا ب NULL .. فانا ممكن اخلي في الحالتين ال previous بتاع ال node الجديدة يشاور على tail ويبقى كدة مفيش اختلاف في 4 step
 و 6 step برضه مفيش اختلاف ... بس 5 step فيها اختلاف وهي دي ال step الوحيدة اللي هن check فيها على ال case

نكتب الكود بقى :

مبدئياً بس خلي بالك كويس جداً إن الكود اللي هنكتبه ده هيكون في ال **h**. مش في ال **cpp** .. برغم اننا هنكتب ال **implementation** اللي متعودين انه يكون في ال **cpp** دائماً
ودة عشان في ال **templates** ماينفعش نخط ال **implementation** في ال **cpp** .. لازم يكون في ال **h**. يا إما الكود مش هيب **compile** .. الدكتور مقالش ليه وقال لو حد عايز يعرف يـ **search** على النت (بس قال ان الموضوع **advanced** اوي وبرة عن الكورس بتاعنا)

عايزين برضه ناخذ بالنا من حاجة مهمة .. في ال **templates** لازم نكتب السطر اللي بند **declare** فيه ال **template** قبل اي **member function** اللي هو دة :

template <class ListElemType>

و دي حاجة رخمة بس يعني ما عندناش غير كدة .. وكمان بنغير شكل ال **definition** بتاع ال **member function** شوية .. يعني بدل ما ال **insert** كان بيتكتب كدة :

bool Dllist :: insert (const ListElemType &e) {

هنكتبه كدة :

bool Dllist<ListElemType> :: insert (const ListElemType &e) {

ولو عايز تعرف ليه كان لازم نعمل التغيير دة انزل تحت شوية لحد ما نوصل لكود ال **main** وهتفهم ليه حطينا ال **<....>** دي

المهم نكتب الكود بقى بجد المرة دي : **D**

[Dllist.h](#)

template <class ListElemType>

bool Dllist<ListElemType> :: insert (const ListElemType &e){

link addedNode; //create the new node to be inserted in list

//step 1

addedNode = new Node; //allocate space from heap

if (addedNode == NULL){

return false;

}

//step 2

addedNode → elem = e;

//step 3

addedNode → next = NULL;

//step 4

addedNode → prev = tail; //this statement is valid for both empty & non-empty lists

//وقلنا برضه المرة اللي فاتت ان دة optimization بس مش اكثر .. يعني لو ماخذناش بالنا منه وعملنا 2 cases مش مشكلة في الامتحانات

//step 5

if (head == NULL) { //it's an empty list

head = addedNode;

}

else {

tail → next = addedNode;

}

```
//step 6
tail = addedNode;
return true;
} // end of member function
```

على السريع بقى كدة تعالى نعمل ال previous كمان :

```
template <class ListElemType>
bool Dlist <ListElemType> :: previous (ListElemType &e) {

/*
احنا هنا ال statement دي كنا كاتبينها عشان نـ check لو كانت empty list او ال current node اللي واقف عليها
دي كانت اول node وبالتالي ال previous بتاعها بـ NULL ... بس دي مش حاجة قوية فيبلاش نعملها وخلصنا نبدلها بـ
اللي جاية دي : */

if (current == NULL || current → prev == NULL) {
    /* ايه يعني اللي فرق head من current ؟؟ .. اللي فرق هو ان current بتـ check على نفس اللي ال head كانت بتـ
    check عليه .. بس كمان بتـ check حصل ان ال user عمل call لـ first الاول وبالتالي بقى فيه current مش بـ
    NULL ولا لا .. نكمل الكود بقى */

    return false;
}

else {
    current = current → prev;
    e = current → elem;
    return true;
}
} // end of member function
```

كدة خلصنا .. بس احنا لسة ماشوفناش ايه فايده ال template او نقطة القوة بتاعتها .. فتعالى نشوف

هنكتب كود ال main .. وانا لسبب ما عايز اdefine مرة doubly linked list of integers ومرة doubly linked list of strings ومرة doubly linked list of class (complex) .. فيوجود ال template اللي عملته دة هقدر اعمل كدة :

[main.cpp:](#)

```
#include"Dlist.h"
#include"Complex.h"

main (){

Dlist <int> iList;
Dlist <string> sList;
Dlist <Complex> cList;
}
```

يبقى انا كدة استفدت من ال template اني عملت نفس ال class ب datatypes مختلفة من غير ما اغير اي حاجة في الكود .. عشان كدة قولنا ان شغلنا في ال classes اللي جي كله هيكون ب templates

بس هل ال template دة جميل كدة في كل حاجة؟؟
للأسف لا

لازم اكون واخذ بالي كويس اوي ان ال operations اللي جوة ال class تكون defined على ال datatype دة ..

يعني مثلاً في previous احنا كتبنا ال statement دي :

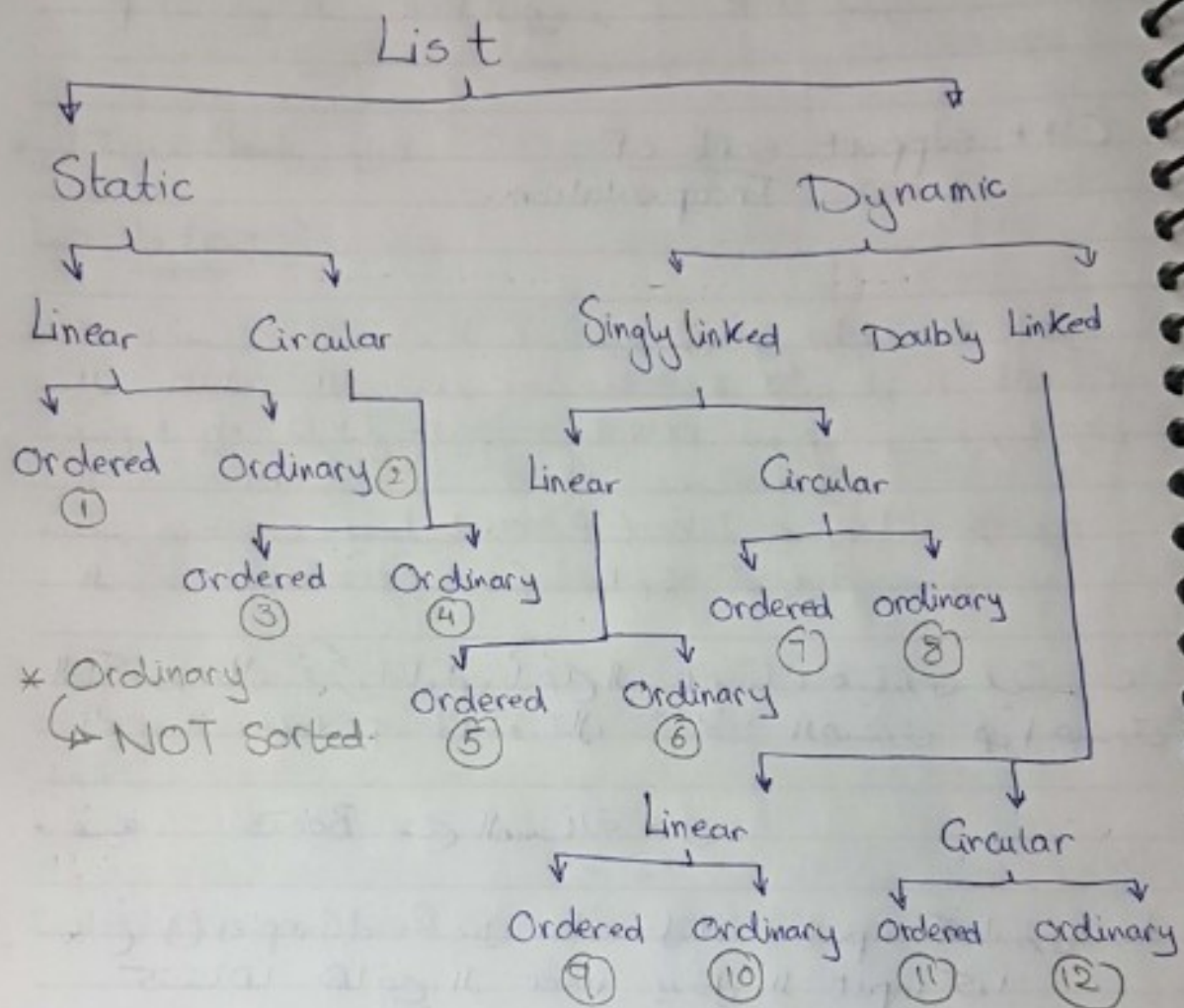
```
e = current → elem;
```

لو كانت ال list معمولة ب integer data type يبقى اشطة تمام
طب لو معمولة ب string .. يبقى ساعتها هشوف انا عامل ال include لل library اللي فيها operation overloading
عشان تقارن strings ببعضها ولا لا .. لو لا يبقى ال = دي هتتعامل مع ال strings على انها pointers ودة مش صح
او ممكن ال list تبقى معمولة ب Complex data type وانا مابقاش عامل ال operator overloading لل complex دة
فممكن يحصل مشكلة برضه
فعامة لازم اكون واخذ بالي كويس من الموضوع دة بالذات مع ال pointers & arrays

مشكلة كمان في ال templates .. انها بوظتلنا ال encapsulation اللي قعدنا نعمله من اول الترم .. لأن ال template
بال implementation بتاع ال class بقوا موجودين في ال h. وبالتالي ال user بقى يقدر يشوفهم عادي

كدة احنا خلصنا ال lists خالص والمحاضرة الجاية هناخد ال queues ان شاء الله
الدكتور بدأ يـ recover اللي اخدناه من ال lists وقال اننا اخدنا 12 lists مختلفة بالتقسيمه دي ... والصفحة دي من دراييف
2018 ^^

* ايه ال Lists ال صاها ؟



* نقدر دلوقتى نعمل ال Lists 12 دول. اشرحهم لى

⑪ Doubly linked Circular Ordered

* مع الامتحان ليقول نستخدم على انى List من دول.