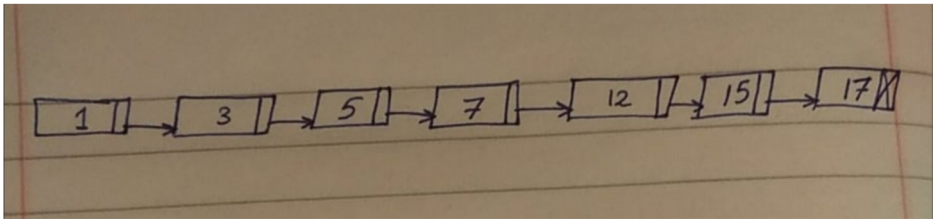
Lecture 9

contents:
-doubly linked list.
-template.

خدنا الlinear list .. والcircular list المحاضرات اللى فاتت.. المحاضرة دى هناخد الdoubly linked list.
ايه دى بقى؟
دى زى الlist العادية اللى خدناها قبل كده ..بس الفرق ان كل node ليها 2 pointers.



ليه؟
تعالى ناخد مثال

مثلا فى الlist دى انا عايز ادور على رقم معين

```
search(12);
```

هنعمل loop تدور فى node..node صح؟
تفتكر الlogic هيبقى ازاى؟
هيبقى بانى اعمل first واقارن الvalue .. وبعدها نبدأ الloop وجواها نعمل next ونقارن الvalue لحد ما نلاقيها.
لما هنلاقى ال12 هخرج من الloop.
لما هخرج قيم الpointers ايه اخباها؟
الcurrent بقى عند ال12... طيب افرض قلت بعد الstatement اللى فوق دى

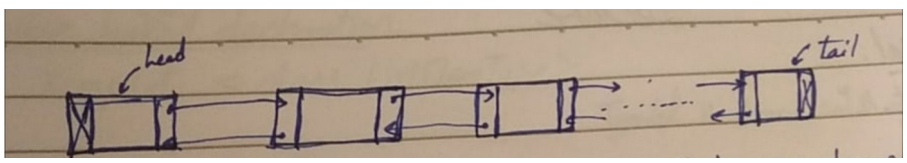
```
search(15);
```

هنا هدخل الloop اللى بتعمل next وتقارن بس الcurrent عند 12 فكده next واحدة وهبقى وصلت لل15 .. يعنى مبدأناش من الأول .. فكده احسن . عشان لو بدأت من الأول هتاخد وقت اطول و اطول جدا لو الlist كبيرة.
طيب لو كان جزء الsearch فى الكود كده

```
search(15);
search(5);
```

هعمل ايه؟؟
مش احسن حل هو انى زى ما طلعت قدام اجيب ال15 المرة اللى فاتت ..ارجع ورا المرة دى عشان اجيب ال5.
بس انا معنديش حاجة ترجعنى ورا؟! يعنى فى المثال اللى فات انا عندى next اطلع بيها قدام .. لكن هنا معنديش حاجة ترجعنى ورا؟
عارف لو كان array كنا عملنا decrement بسهولة لرقم الخانة و قرينا .. لكن دى list اماكنها من الheap.
فعشان نرجع بسهولة فكرنا نعمل pointer لورا عكس بتاع الnext .ودى فكرة الdoubly linked list.

فزى ما شايفين فى الصورة .. كل node ليها 2pointers واحد بيشاور على الnode اللى بعدها و واحد بيشاور على ال node اللى قبلها.



ثانية واحدة.. طب مانا ممكن اعملها بالlist العادية ام pointer واحد؟
اه دى حقيقة .. ممكن فعلا اعمل loop على الnodes من أول node واسأل هل الcurrent بيساوى الnext→pred وللا لا
الcondition ده فى المثال بتاع الارقام بيسأل هل الnode اللى بعدى هى الcurrent لو اه يبقى انا وصلت لل7 اللى هى قبل ال12 اللى عندها الcurrent.

لو اه خلى الpred→next=pred يعنى خلى الcurrent بدل ما يشاور على ال12 يشاور على ال7
وافضل اعيد فابقى كده برجع ورا من غير doubly linked list .

بس طبعا لو list كبيرة ..الكلام ده هياخد وقت كبير اوى عشان الloop كبيرة.

طب وطريقة الdoubly linked list ايه ال cost اللى هدفعه؟
الcost هو pointer زيادة per node.

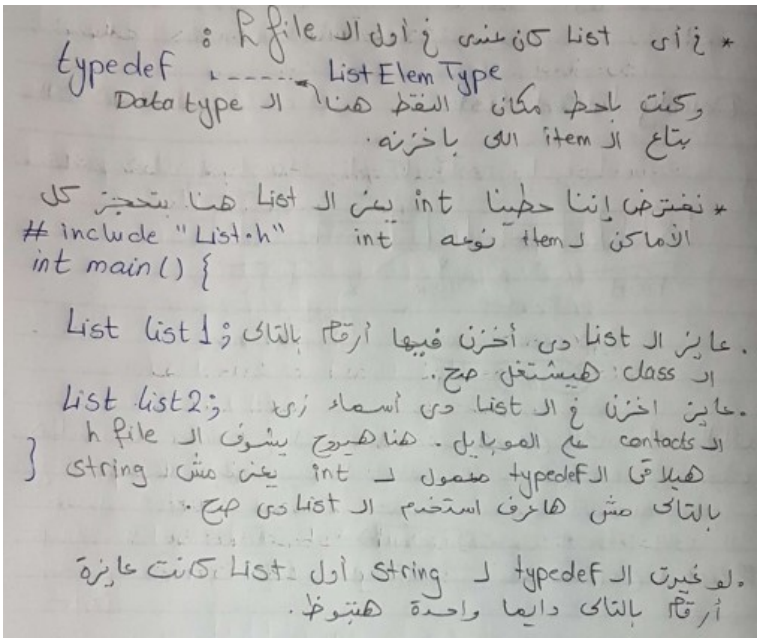
طيب ايه الoperations اللى اقدر اعملها فى نوع الlist ده؟

operations:
insert() first() next() previous()

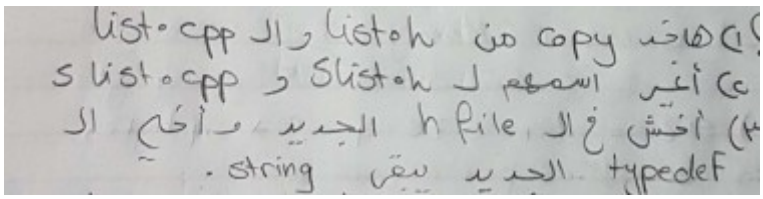
قبل ما نكتب الكود .. نتعلم حاجة جديدة اسمها الtemplate موجودة فى الc++ ومش موجودة فى الc

Template:

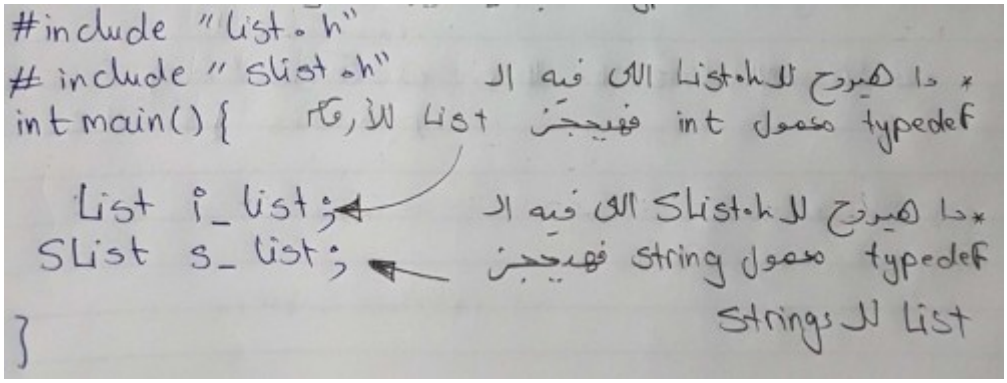
عارف الاسطمبة اللى كنا بنستخدمها الرسم الهندسى .. دى حاجة شبهها .. ازاى؟



الحل اللى اى حد فينا لسة ميعرفش الtemplate هيفكر فيه هو..



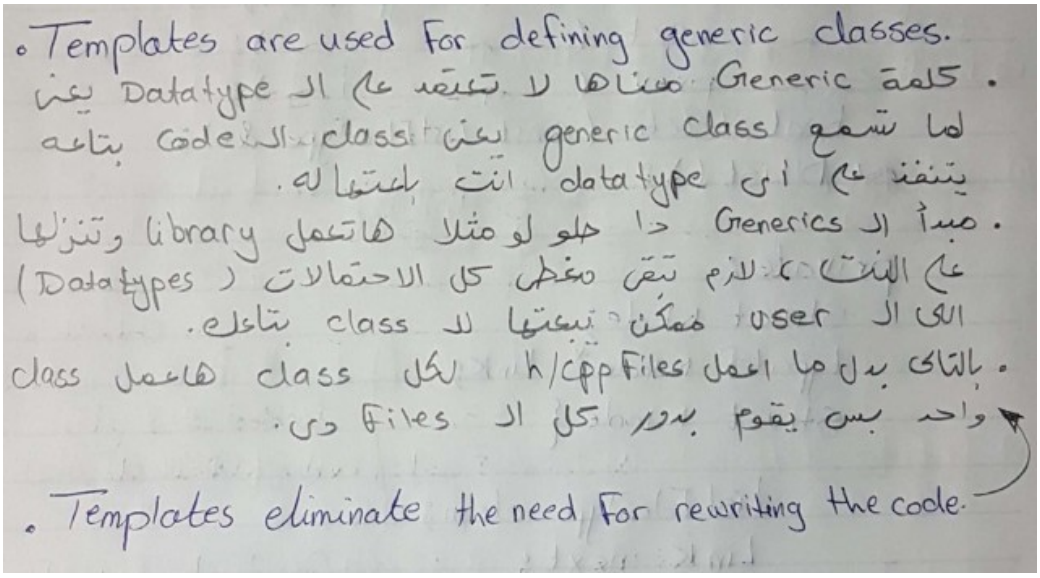
الكود بعد كده هنكتبه كده

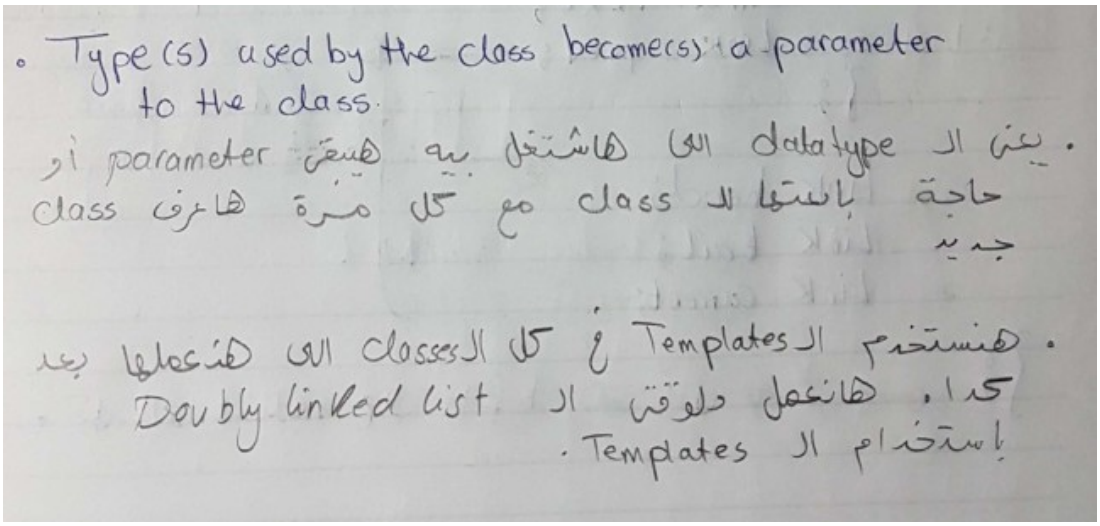


افرض بقى رخمت معايا .. انى عايز اعمل D": char list .. هعمل الكلام ده كله تانى والفكرة ان هو سطر واحد اللى بغيره بعد ما اعمل copy .

طب ايه رايك انى اعمل class واحد وابقى ابعت الdata type اللى عايز اعمل بيها الclass ده وخلاص ... باستخدام file واحد .h

ده دور الtemplates





نكتب الـ.h بقى

template < class ListElemType>

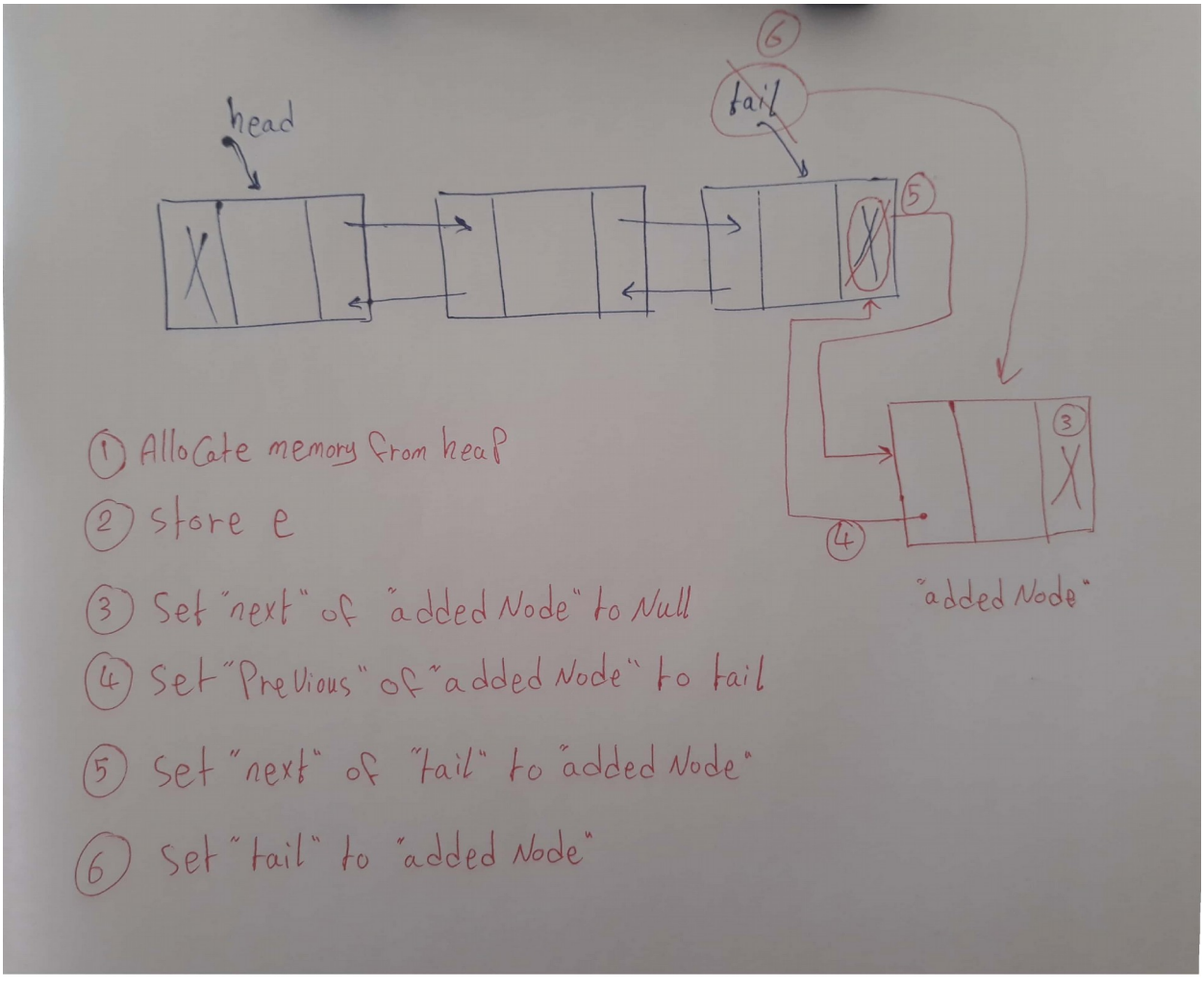
الجملة دى كده حلت محل الtypedef

```
class DLLIST{
	public:
		DLLIST();
		~DLLIST();
```
ده الdestructor .. معملناهوش بس مهم فالدكتور قال نعمله exercise .
```
		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;
};
```
كل ده مفهوش حاجة جديدة غير اول سطر بتاع الtemplate واننا عرفنا pointer اسمه prev فى structure الnode.
خد بالك الدكتور قال ان فى الtemplate لازم كل حاجة تبقى فى الh. .. ليه؟ قال دور عليها على النت لو interested …
المهم ان اللى جاى اللى هو تعريفات الfunctions كمان هتبقى فى الh. مش بس اللى فوق.

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

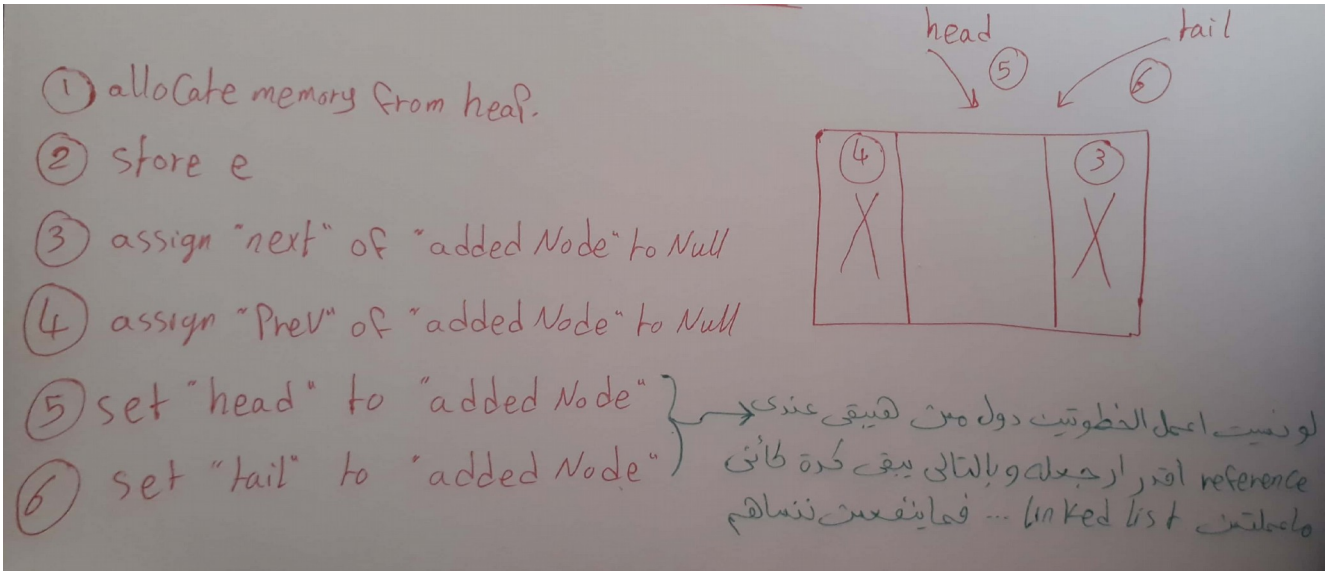
general case:

معايا list جاهزة وهـ insert عليها في الاخر :



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 بقى 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 كدة كذا .. انا بس عايز ابقى عارف كويس اوي انا بعمل ايه واللي بعمله دة هيكون ايه تأثيره .. لو مفيهوش مشكلة خلاص يبقى go on

Special case: Empty list



لو هنبص على الفرق اللي بين ال cases 2 اللي فوق هنلاقي ان اول steps 3 هم هم . بس step 4 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 step 4
و step 6 برضه مفيهاش اختلاف … بس step 5 فيها اختلاف وهي دي ال step الوحيدة اللي هنـ check فيها على ال case

نكتب الكود بقى :
مبدئياً بس خلي بالك كويس جداً إن الكود اللي هنكتبه دة هيكون في ال .h مش في ال .cpp .. برغم اننا هنكتب ال implementation اللي متعودين انه يكون في ال .cpp دايماً
ودة عشان في ال templates ماينفعش نحط ال implementation في ال .cpp .. لازم يكون في ال .h يا إما الكود مش هيـ 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 ناخد بالنا من حاجة مهمة .. في ال templates لازم نكتب السطر اللي بنـ declare فيه ال template قبل اي member function اللي هو دة :

template <class ListElemType>

و دي حاجة رخمة بس يعني ماعندناش 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 اكتر .. يعني لو ماخدناش 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 Dllist <ListElemType> :: previous (ListElemType &e) {

/*
if (head == NULL || current → prev == NULL){
   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 ب NULL ... بس دي مش حاجة قوية فبلاش نعملها وخلونا نبدلها ب
   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 ل first الاول وبالتالي بقى فيه current مش ب
      NULL ولا لأ .. 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 فايدة ال template او نقطة القوة بتاعتها .. فتعالى نشوف

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

main.cpp:

```
#include"Dllist.h"
#include"Complex.h"

main (){

Dllist <int> iList;
Dllist <string> sList;
Dllist <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 لأ
لازم اكون واخد بالي كويس اوي ان ال operations اللي جوة ال class تكون defined على ال datatype دة ..

يعني مثلا في previous احنا كتبنا ال statement دي :

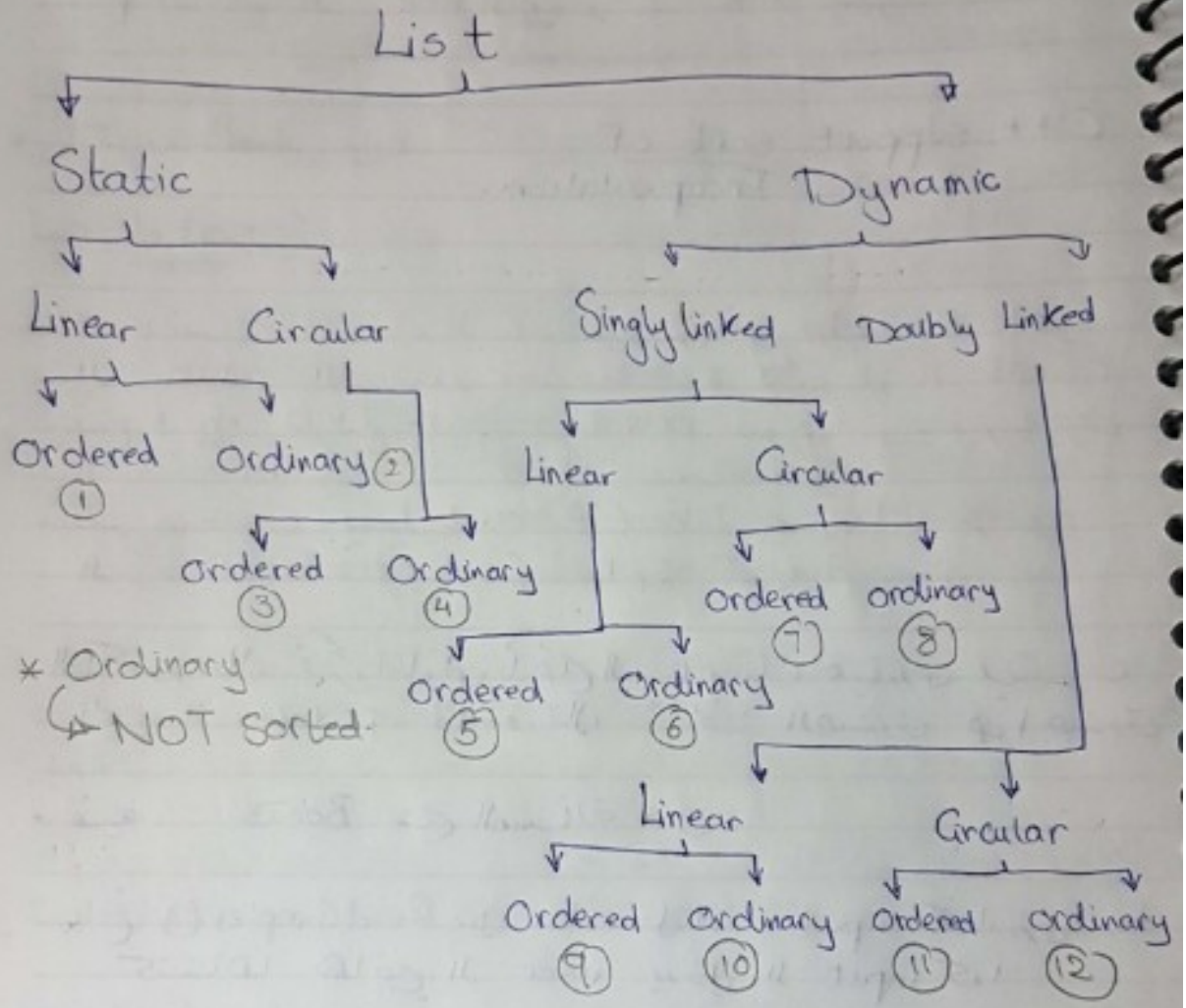
```
e = current → elem;
```

لو كانت ال list معمولة ب integer data type يبقى اشطة تمام
طب لو معمولة ب string .. يبقى ساعتها هشوف انا عامل include لل library اللي فيها operation overloading عشان تقارن strings ببعضها ولا لأ .. لو لأ 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 operator overloading لل complex دة فممكن يحصل مشكلة برضه
فعامةً لازم اكون واخد 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 بـ recover اللي اخدناه من ال lists وقال اننا اخدنا 12 lists مختلفة بالتقسيمة دي … والصفحة دي من درايف 2018 ^^

* ايه الـ Lists اللي جوا ؟



* نقدر دلوقتي نعمل الـ 12 Lists دول. أجيبهم لك

(11) Doubly linked Circular Ordered

* في الامتحان هيقول نشتغل مع اللي List من دول.