Lecture 13

Lecture contents:

1-Trees concept and definitions
2-Static (linear) tree implementation
3-Dynamic (linked) tree implementation
4-Trees Traversal methods (preorder- inorder – postorder)
5-file.h & file.cpp implementation

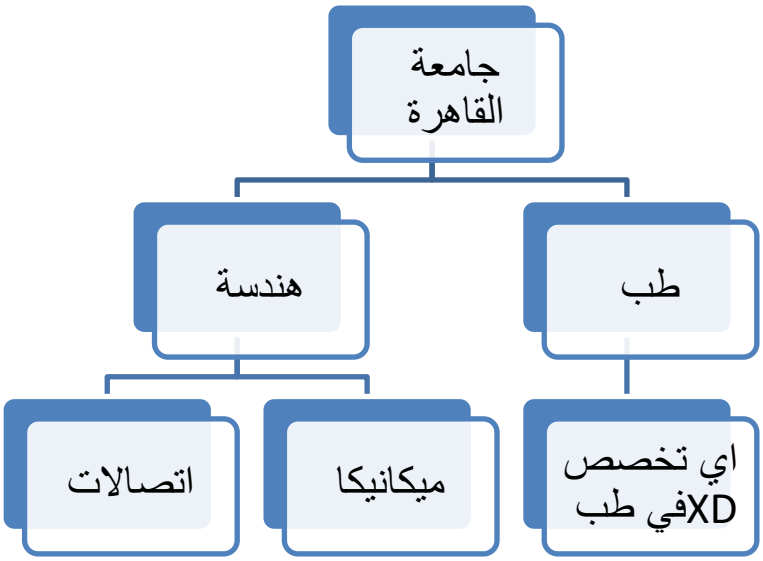
مبدئياً المحاضرة طويلة اوي معلش بس انت مش محتاج تفتح السلايدز معاها عشان كلها موجودة هنا وهتلاقي الدنيا ماشية واحدة واحدة فممكن تجري انت شوية لو فاهم الموضوع

بص كدة آدي اول صفحة خلصت :D

احنا دلوقتي في كورس ال data structureوقفنا لحد اخر حاجة مع د خالد في ال Queuesطب احنا كنا عرفنا عن ال queuesانها listعادية linearبس فيها صفة معينة انها FIFO :first in first outو كنا بنستخدمها عشان نعمل ال موديل بتاع Buffer اننا عايزين نخزن داتا مثلا جاية من حاجة سريعة زي processor و رايحة لحاجة بطيئة زي printer مثلا .فنوقف كل الداتا في طابور كده و ندخل واحد واحد .

طب دلوقتي بقي هناخد حاجة جديدة خالص و مختلفة الي حد كبير عن كل ال one dimension Linear lists اللي اخدناها من اول السنة و هي ال TREE .

طبعا ال Tree اللي هي الشجرة xD و فعلا ال data structure ده شبه الشجرة بس مقلوبة يعني ال root فوق و ال leavesتحت و ال structure ده بيبقي افضل في تمثيل اشكال كتير من الداتا يعني كمثال : لو عايزين نحط جامعة القاهرة و نحط كل الكليات اللي جواها, بعظين كل الاقسام اللي جوا كل كلية منهم ... فده اكيد مش هينفع ب linear list عادية .



فهنا احنا بنحاول نلاقي ال data structure اللي هتوفرلي :

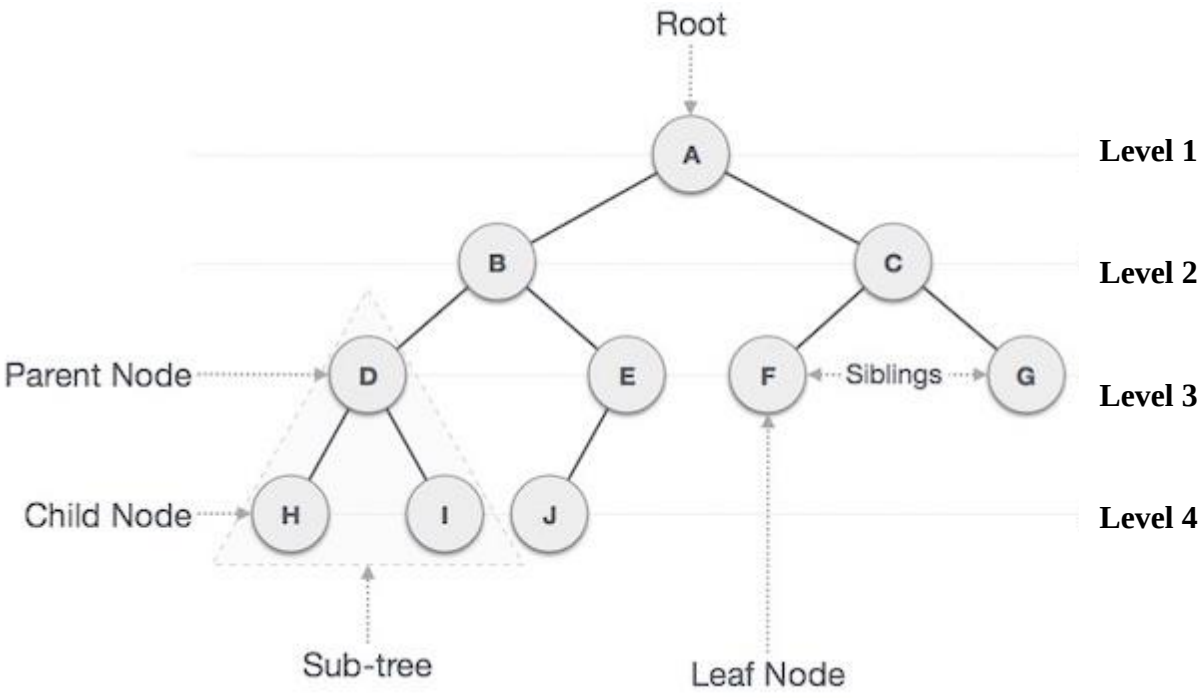
1--Best saving method for data

2-Best retrieval method for data

// من الحجات المميزة قوي في ال tree هو ان ال search فيها سهل اوي و هنشوف كده مع بعض قدام .

*/ فاكرين ال recursion ؟ اللي هو كانت فانكشن بتفضل تنده نفسها كذا مرة و لازن يكون في end condition عشان لو مفيش هتخش في لوب و تعمل stack over flow , ايوة هي ديه بقي بالظبط الطريقة اللي هنستخدمها عشان نعمل ال tree و هنعرف هتسهل علينا /*كتابة الكود قد ايه .

في شوية definitions مهمة هنحتاج نفهمها كويس و نحفظها عشان هنشتغل بيها علي طول في ال trees فنركز مع بعض يا اخونا في الحتة الجاية ديه :



Node - tree element → كل دايرة من الدوايرديه اسمها نود

Root - the node at the top →النود اللي فوق خالص اللي كل الشجرة متفرعه منها

Parent/child nodes:

كل نود بيطلع منها اي عدد من Nodes فهي هتبقي اسمها parent و النودز اللي طالعين منها دول هم اولادها child
nodes يعني مثلا : D هي parent ل H ,I اللي هم . child Nodes

Degree (of node) - number of child nodes:

يعني النود ديه ليها كام child مثلا Node D is of order 2 &Node E is of order 1

Siblings - have the same parent:

يعني اخوات يعني ليهم نفس الاب في المثال بتاعنا هم H&I

Subtree:

ديه مهمة نركز بقي : دلوقتي عندنا كونسبت كده في الشجر العظيم اللي بنعمله ده و الكونسبت ده اللي هيسهل علينا اوي ال implementation بتاع الكود و هيخلينا نقدر نستخدم فكر ال recursion في الكود , دلوقتي تخيل اننا شيلنا ال A خالص و فكينا الدراعات اللي طالعه منها , بص كده علي B , C هتلاقيهم بقول roots لشجر جديد اصغر من الشجرة الاولانية . دي ديه بالظبط فكرة ال subtree يعني كل node طالع منها nodes تانية علي بعضهم شجرة جوا الشجرة الكبيرة و الشجرة جواها شجر تاني اصغر و هكذا .

Ancestor/ Descendant:

بما اننا قلنا ان فيه ابهات و امهات و اخوات خلينا نعتبرها انها شجرة العيلة XD و مرحناش بعيد يعني ما هي شجرة بردو . دلوقتي لو وقفنا عند B فهي parent ل D ماشي زي الفل و D بردو parent ل H & I زي الفل بردو , فكده B تبقي جد ل H,I و بردو A يعتبر جد او ancestor فاي واحد في سلسلة العيلة يعني جدي او جد جدي او جد جد جدي ... هو ancestor ليا. و العكس بقي descendent يعني ابن او ابن ابني او ابن بنت بنت بنت بنتي XD هي ديه بالظبط اللي توضح المعني . فكده اي Node في ال treeبتعاتنا ديه يعتبر descendent ل A .

Leaves - nodes without children:

دول الNodes اللي في اخر العنقود اللي ملهمش اولاد خالص . Nodes with no children

Internal nodes - nodes with children:

كل اللي ليهم اولاد يعني كل الشجرة معادا ال leaves

Edge/branch/link/arc - connection between one node and another:

ديه سهلة مش محتاجة شرح ده اي خط او سهم بين اتنين Nodes

Path - sequence of edges from root to node:

يعني لما تحط القلم علي ال root و تفضل ماشي لحد ما توصل لل Node اللي انت عايزها.

Length of the path - number of edges in the path:

بتبدا من عند Node معينة و تروح لوحدة تانية انت ماشي بقي شوف عديت علي كام Link

Depth of a node - length of path from root to node:

زي فوق بالظبط بس الفرق انك بتبدا هنا من عند ال Root و ترح لل Node ال انت عايزها.

Levels - the number of edges between the node the root+1:

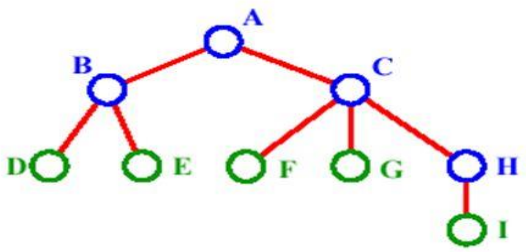
زي ما مكتوب هم عدد ال edges من الرروت لحد ال Node اللي انت عايزها و بتجمع عليهم واحد ديه مهمة اوي عشان احنا بنفرض ان ال root اللفل بتاعها 1 مش 0

Height – maximum level of a node in a tree:

ده طول الشجرة و هو بيساوي عدد ال levels -1

و ديه سلايد من النت فيها شوية امثلة علي الكلام اللي فوق.

Trees: Terminology



- *A* is the *root node*
- *B* is the *parent* of D and E
- *C* is the *sibling* of B
- *D* and *E* are the *children* of B
- *D, E, F, G, I* are *external nodes*, or *leaves*
- *A, B, C, H* are *internal nodes*
- The *depth*, *level*, or *path length* of *E* is 2
- The *height* of the tree is 3
- The *degree* of node *B* is 2

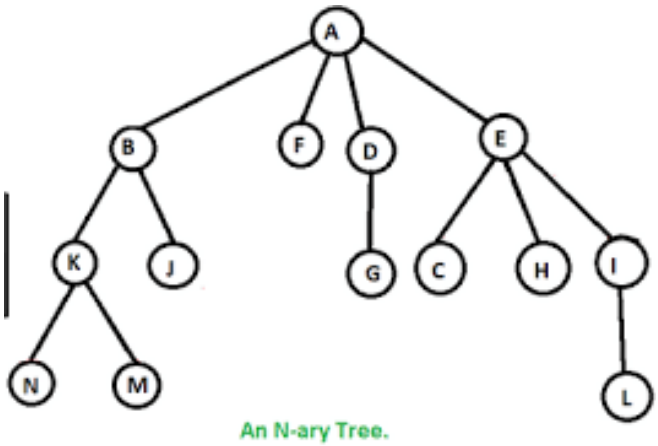
Property: |*edges*| = |*nodes*| - 1

طيب دلوقتي خلاص خلصنا التعريفات و كله بقي فل , نخش علي الجد بقي :

احان عندنا ال Tree data structure بيتقسم لكذا نوع , و كل واحد ليها حاجة هو مميز فيها و معمول عشانها , يعني مثلا عندنا :

1- N-ary Tree: each Node has N Children at maximum

و ديه ال general case عندنا و N مهم نركز في معناه : اللي هو اكبر عدد من ال children ل Node واحدة في ال tree ديه . يعني ايه الكلام ده ؟ يعني مثلا في الصورة ديه :



An N-ary Tree.

اكبر عدد من ال children هو عند A اللي هو 4 فديه كده 4-arry tree و نلاحظ ان فيه Nodes عندها اقل من 4 و nodesمعندهاش خالص بس احنا بنشتغل علي اكبر رقم من الاولاد ل Node واحدة .

2-Binary tree:

ديه بقي اللي احنا هنشتغل عليها طول الكورس و هي special case من اللي فوق و بردو ممكن نسميها : 2-arry tree .
لو فهمنا اللي فوق هنفهم الكلام الجي ده :

The binary tree has 3 cases for each Node:

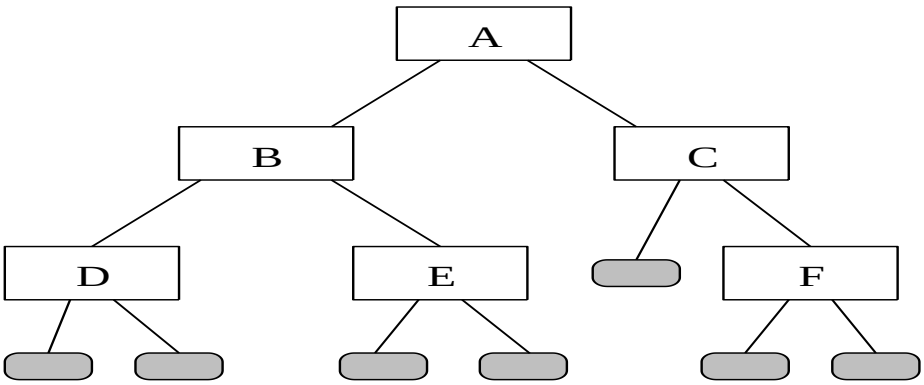
1-has 0 childe (order 0)

2-has 1 childe (order 1)

3-has 2 childe (order 2)

يعني ايه الكلام ده : يعني كل Node عندي في ال tree هيبقي ليها حالة من ال3 اللي فوق دول و كده هتبقي ديه اسمها binary tree .

ده مثال لل Binary tree :

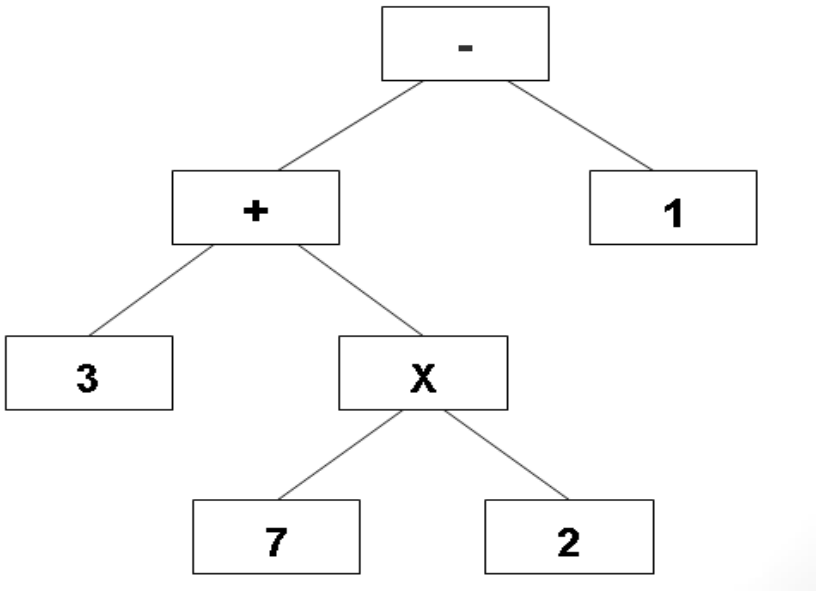


تعالو نعمل Flash forward كده و نشوف مثال لاستخدام ال Binary Tree ديه في مثال من الواقع المتهالك :

دلوقتي ل عندنا Expression زي ده و عايزين نحلة بالترتيب ف احنا فاكرين طبعا من ايام د ايهاب بنمشي ازي اول حاجة اللي بين الاقواس بعدين الضرب و القسمة من الشمال لليمين بعدين الجمع و الطرح من الشمال لليمين :

- 3+7*2-1
- 3+(7*2)-1 grouping for * precedence
- (3+(7*2))-1 left->right associative + vs. -

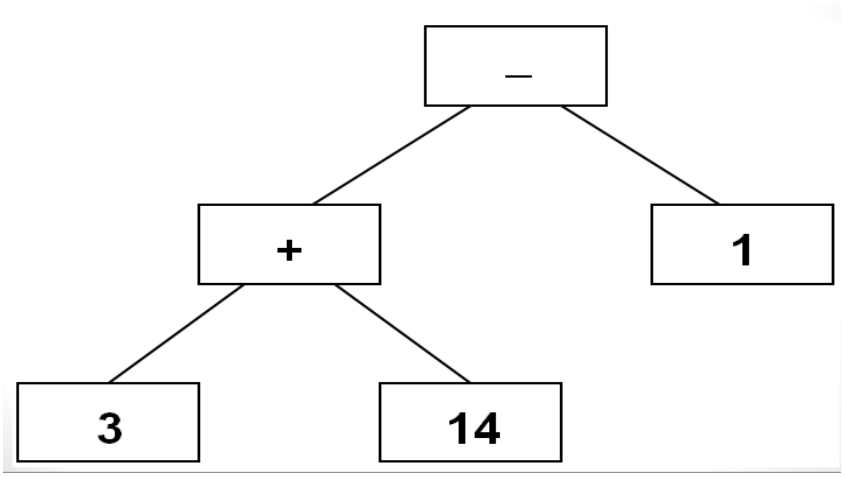
طب يلا نرصهم جوا ال Binary tree و نشوف ممكن نستفيد ازاي منها اننا نمشي ال expression صح بالترتيب و احنا بن traverse فيها :



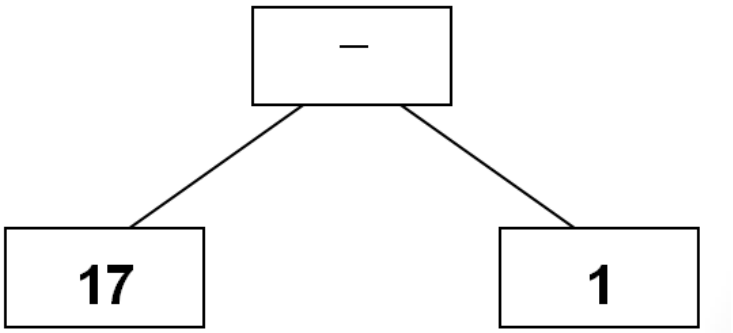
من غير تفاصيل يعني ,احنا رصيناهم بالطريقة ديه عشان لما اجي امشي دلوقتي يطلعوا صح في الترتيب تعالوا نشوف ازاي :

احنا هنبدا نجمع من تحت خالص : اول حاجة هنبدا باخر لفل تحت اللي هو لفل 4 و نعمل ال operation بتاعته اللي هي ال parent بتاعت ال operands اللي في اخر لفل يعني هنضرب 7*2 = 14

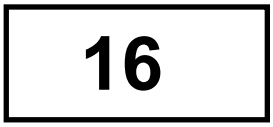
بعد كده نشيل ال subtree دي و نحط الناتج بتاعها مكانها :



و نعمل نفس الكلام كمان مرة , افتكر اننا ماشيين من تحت لفوق من اخر لفل تحت لحد ال .Root



طب كمان مرة والنبي يا ريس XD



كده خلاص بعد ما نخلص ال tree كلهاهنلاقي الناتج طلعلنا صح .

و ده كان مثال في استخدام ال Binary tree في اننا ن evaluate mathematical expression فكر كده لو عزنا نحل نفس ال expression جه ب linear list كانت هتاخد مجهود قد ايه .

Implementing Binary Trees:

- Two methods:
  1. Linear representation - array
  2. Linked representation – pointers

// زي الل lists بالظبط عندنا بردو هنا طريقتين ال arrays و ال dynamic .

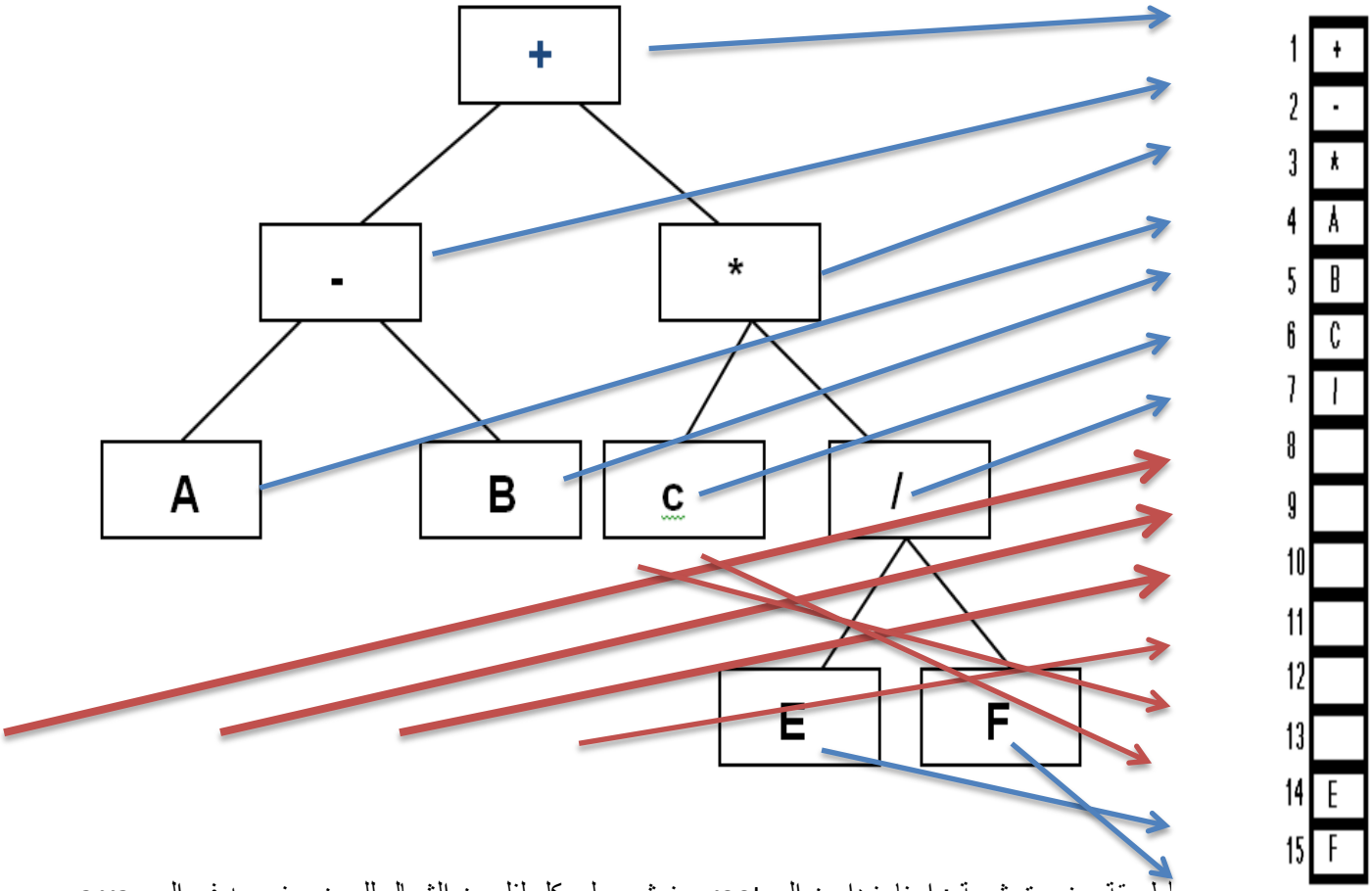
اول طريقة هناخدها بمثال عشان نفهم :

BT Linear representation :

1. Allocate an array of size $2^{(depth+1)}$-1
2. Store root in location 1
3. For node in location n, store left child in location 2n, right child in location 2n+1

Example: What is the needed array size for depth=3?

اول حاجة تيجي في دماغنا ازاي هنحط الشكل المعقد بتاع ال tree ده جوا array باطول كده ؟ تعالوا نشوف : اول حاجة طول ال array هيكون $2^{(depth+1)}-1$ هنفهم الرقم ده لما يجي نرصهم متقلقش:"(



لطريقة وضحت شوية : احنا بنبدا من ال root و نمشي علي كل لفل من الشمال لليمين و نرصه في ال array و و ي جات ملهاش children هنفترض انها عندا كبر عدد من ال children in the tree اللي هم اتنين هنا عشان ديه binary tree و نحطهم فاضيين عشان ديه static list و لو عوزت احطلهم بعد كده يبقي ليهم مكان محجوز .

دلوقتي هناخد واجب و الدكتور قال مهم اننا نعمله و نركز فيه جامد علشان اي حاجة مبتتشرحش في المحاضرة هوب دبل كيك **بتيجي في الامتحان**...

عايزين بقى نكتب كود بيعمل access for a specific node in a tree .. واحنا شغالين في الكود هنلاقي ان لازم ال size بتاع ال array يكون $2^{(depth+1)}-1$ .. ليه بقى لازم يكون كده؟ الدكتور قال شوف انت ليه ☹؟

نرجع لموضوعنا بقى ..
بعد م شوفنا ال static implementation for binary trees كده هي حلوة ولا ال dynamic implementation هتبقى أحسن؟
مفيش حاجة اسمها أحسن .. دايما فيه trade off بيحصل..نبص كده على ال static implementation (using arrays)

- Fast access (given a node, its children and parent can be found very quickly) ☺
- Slow updates (inserts and deletions require physical reordering) ☹
- Wasted space (partially filled trees) ☹

في ال static implementation اقدر اعمل اكسيس بسرعة اوي لل nodes بتاعتي لان كل node ليها اندكس معين جوه ال array و دي حاجة حلوة ☺
بس فيه مشكلة بقى هتظهر لما احب ازود node في النص .. هحتاج اشفت كل ال array بتاعي لتحت و ده بياخد وقت خصوصا لو ال array طويل (physical reordering)
مشكلة كمان هي ال wasted space زي م قولنا قبل كده و انا بخزن الشجرة بتاعتي في الميموري انا بعمل حساب لل nodesاللي ممكن تتحط و هي actually مش موجودة بس انا بردو بعمل حسابها و المشكلة دي بتظهر اوي لما الحجم بتاع الشجرة يكبر مننا ☹

نروح نبص بقى على ال dynamic implementation.. كل node بتتكون من جزئين:

1- Data stored
2- Pointers to children

و ناخد بالنا بقى من الكلمتين دول: انا شخصيا كnode مش عارف انا فين.. بس ال Parentبتاعي عارف انا فين... و الناس هتوصلي عن طريق ال parentبتاعي ☺

In trees pointers point downwards only.

نبص بقى على ال different operations in trees :

- **Characteristics**
  - A Binary Tree ADT T stores data of some type (btElementType)
- **Operations**
  - isEmpty
  - getData
  - insert
  - left
  - right
  - makeLeft
  - makeRight

هنعرف ال implementation بتاعه كل function بعد شوية .. بس ناخد شوية hints كده الفانكشنز دي بتعمل ايه..

Insert: used to put new elements in the middle of the tree.

Left/right: used to traverse the tree, only read elements in left or right node. Returns a pointer to binary tree left or right.

makeLeft/makeRight: used to extend the tree, to add new node at the leaves.
اللي هبعته لل node اللي عايز ازودعليها سواء left or right لازم ابعتلها binary tree و النود اللي باعتلها الكلام ده
يكون فيها 2 pointers واحد لleft binary tree و واحد ل binary tree right

Recall:

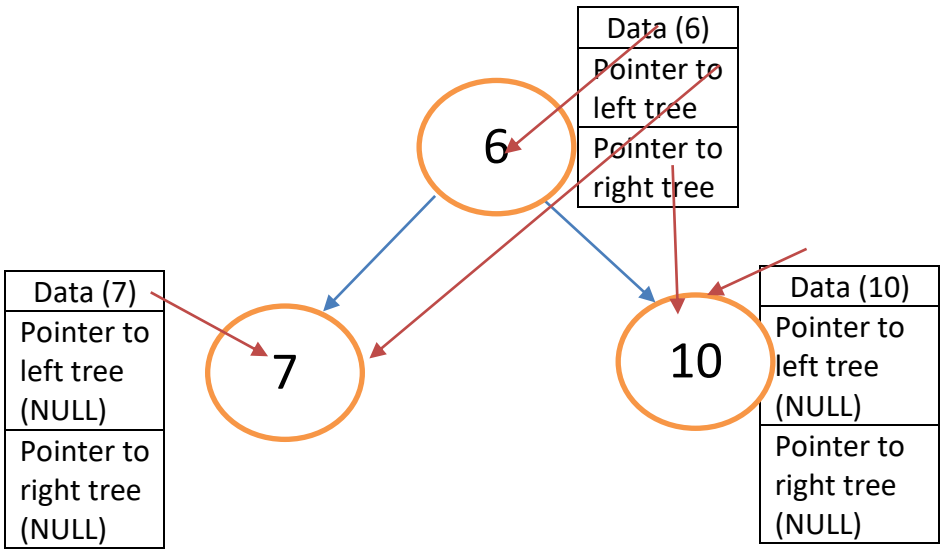
The very first node during the initialization of the binary tree is a NULL node.

Each tree consists of a node and 2 subtrees and so on.

Each node is a binary tree.

Leaves have NULL nodes.

We have to make **destructors** NEVER FORGETHEM



- طب سؤال يطرح نفسه بقى.. ايه الفرق بين insert and makeLeft/makeRight؟
  insert is used to insert a node between a parent and a child
  بنحدد بالظبط هنعمل insert بين انهي parentو انهي (child (Left or Right
  لكن makeLeft/makeRight بتعمل new nodes عند ال leaves

الدكتور قال بردو فكروا ازاي نعمل implementationل insert function
ممكن تيجي في الفاينال ☹

دلوقتى عايزين نعرف ازاى نقرأ كل الdata اللى فى الtree؟؟
لو حبينا نعمل كده فى الlist كنا بنعرف .. بأننا ناخد node node بالترتيب و نقرا اللى فيهم .. بس دلوقتى الtree ملهاش ترتيب ؟
فعملوا 3 طرق نقدر نقرا بيهم كل الdata اللى فى الtree:

Methods of Tree Traversal

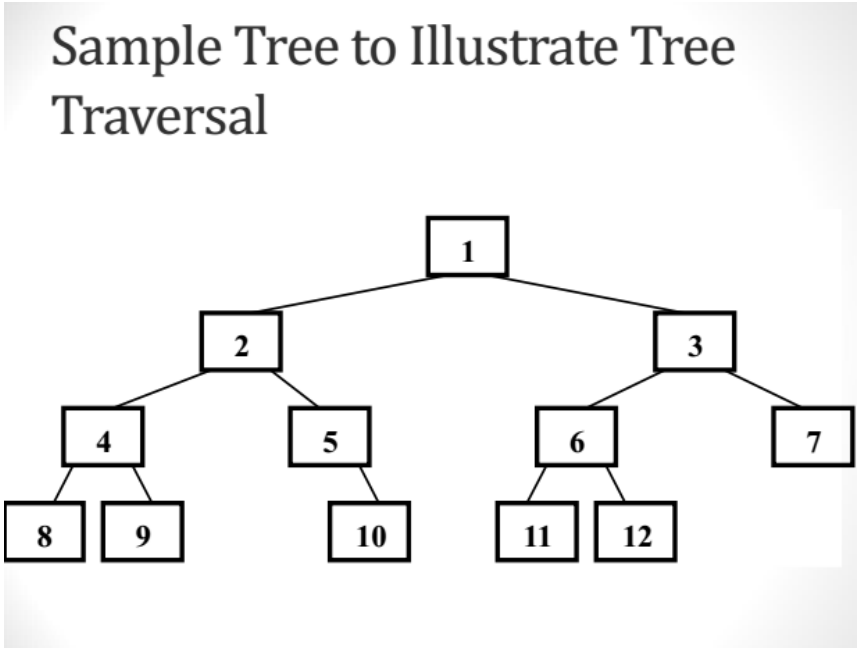
- Must visit every element once
- Must not miss any
- Three basic types
  - preorder
  - inorder
  - postorder

اول نوع اسمه الpreorder traverse:

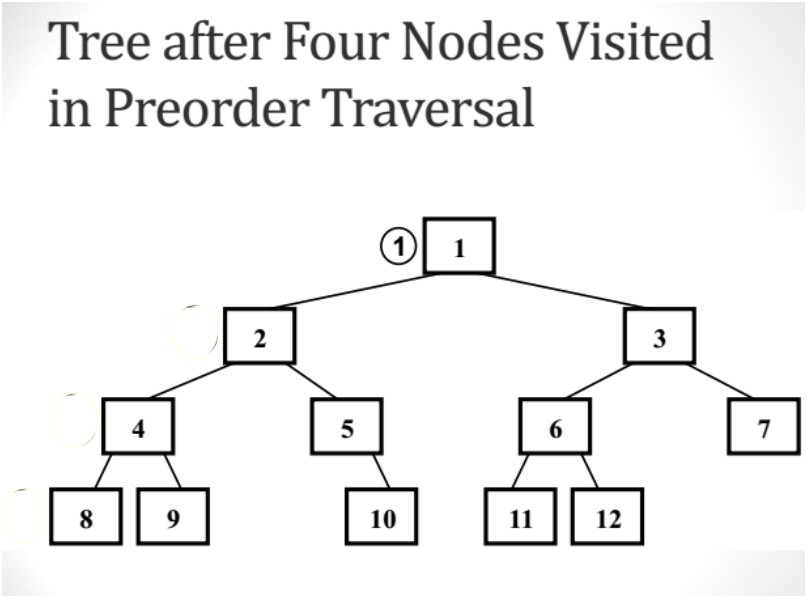
Preorder Traversal

- if the tree is not empty
- visit the root
- preOrderTraverse(left child)
- preOrderTraverse(right child)

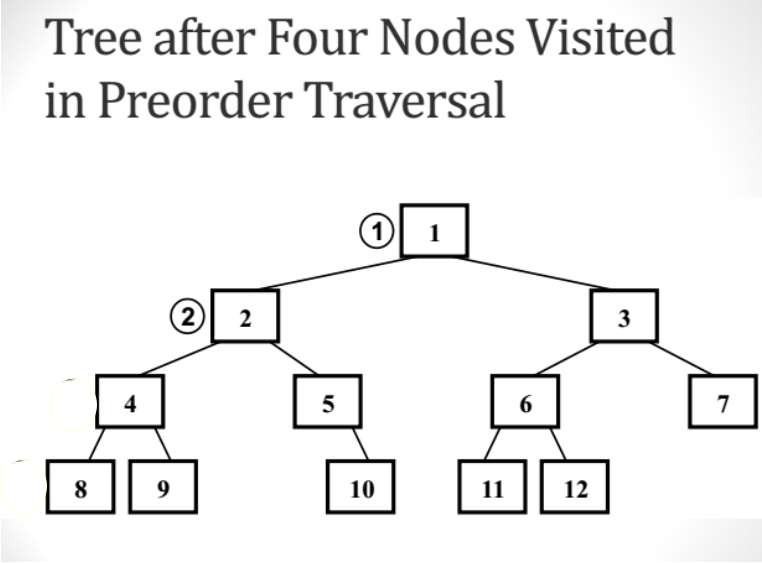
يعنى ايه الكلام ده .. يعنى بقرأ الdata اللى فى الroot وبعدها الleft وبعدها الright ...يعنى ايه؟
ناخد مثال نشرح عليه



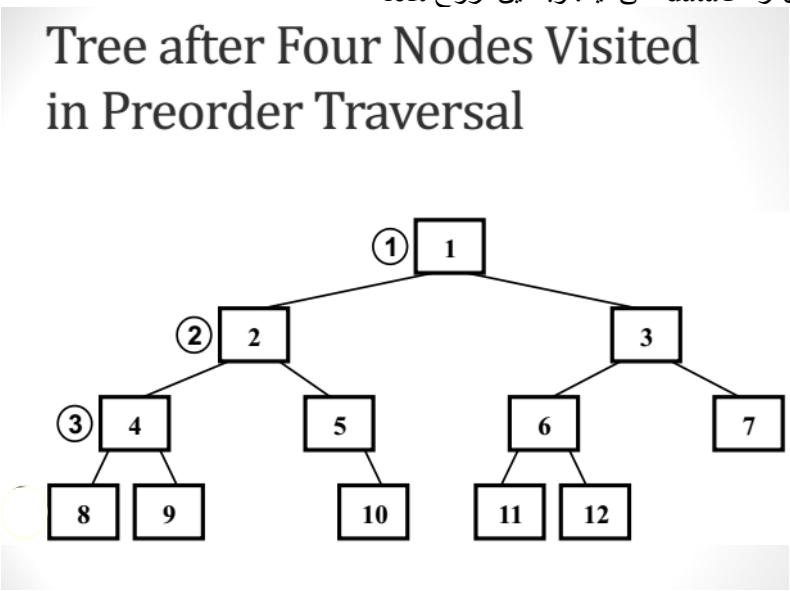
قلنا طريقة الpreorder traverse هى الroot ثم الleft ثم الright .
يبقى اول حاجة هقرا الdata بتاعتها هى الroot اللى فيه 1



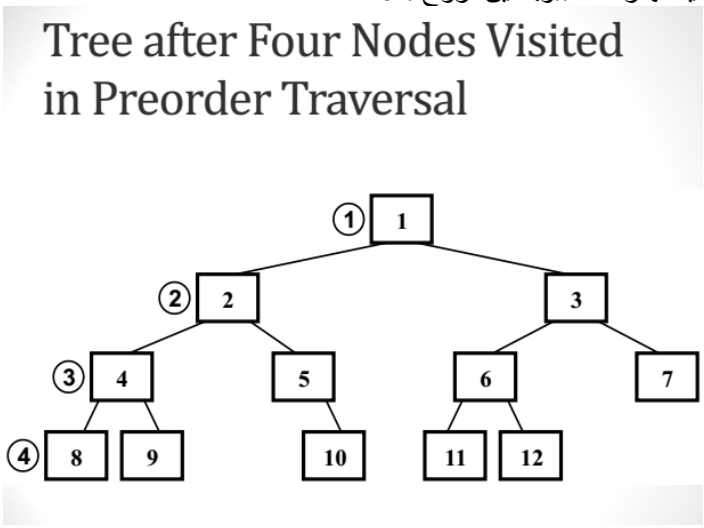
وبعدها الleft بتاع الroot ده



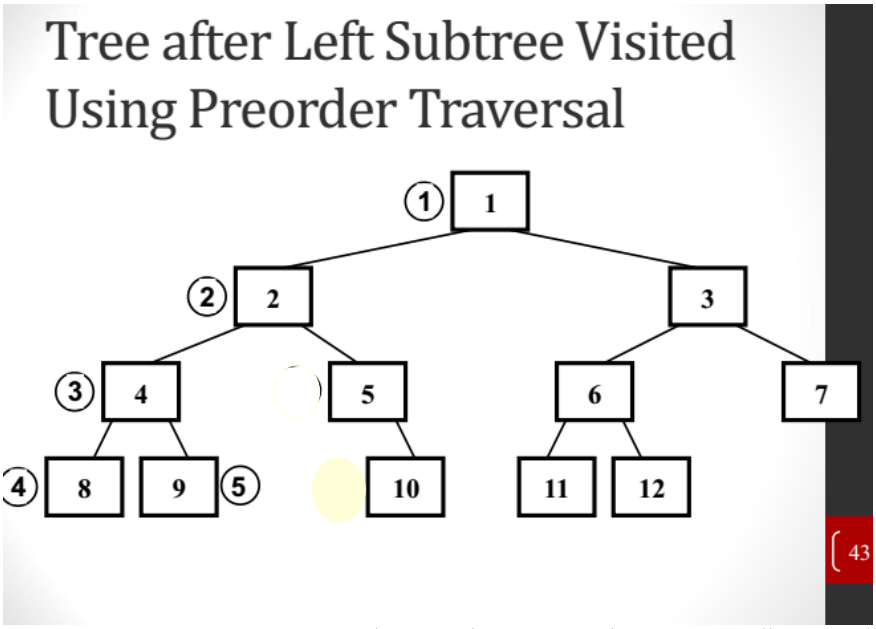
الleft ده بقى root جديد فهقرأ الdata اللى فيه. وبعدين اروح left



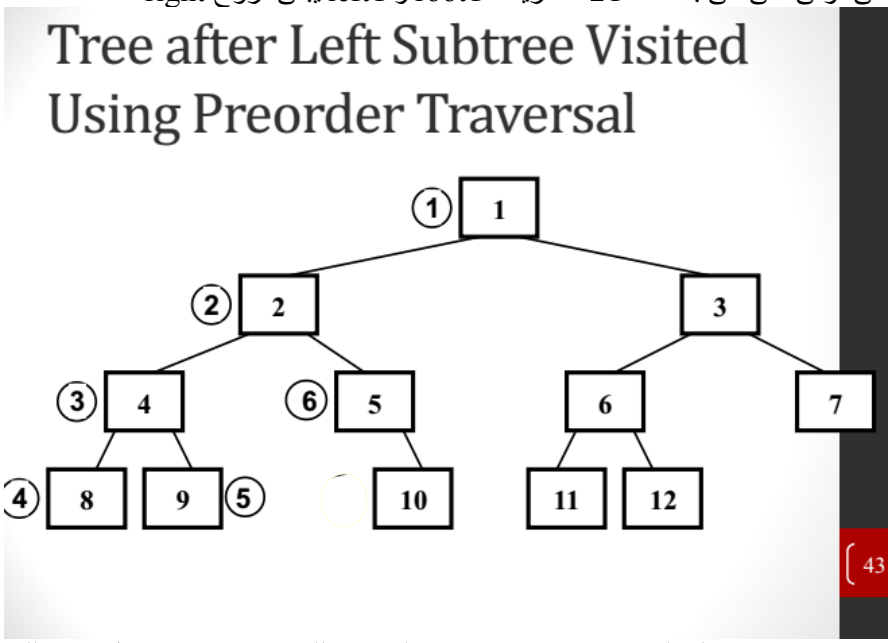
الnode دى بردو بقت root جديد فهقرأ ال4..وبعدين اروح left



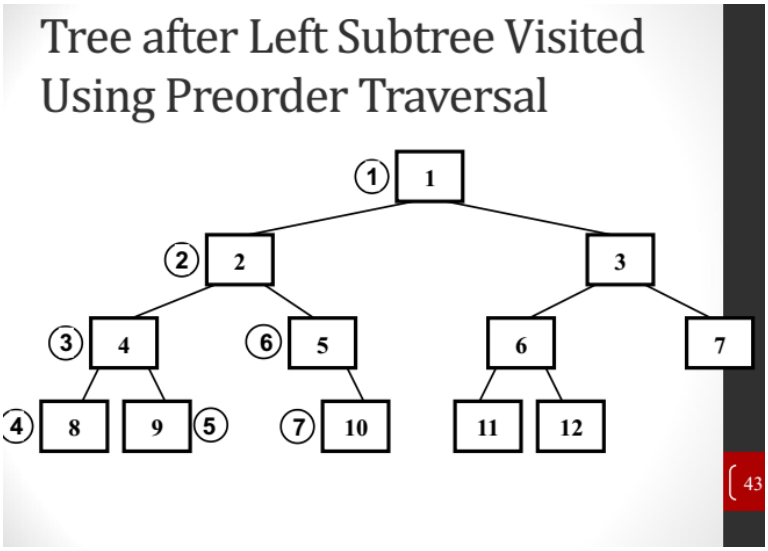
وبعدها هنروح فين .. زى ما الترتيب بيقول root ثم left ثم right
يعنى هروح للnode اللى فيها 3 ؟
لا مش مرة واحدة كده .. اصلا هنعرف ازاى من ال8 نروح لل4 مرة واحدة فالمنطقى دلوقتى اننا نكمل اخر tree كنا بنشوف الdata بتاعتها اللى هى الtree بتاعت ال4 انا عملت فيها root ثم left وكنا واقفين فى الخطوة دى .. نكمل بقيت الترتيب و نروح right



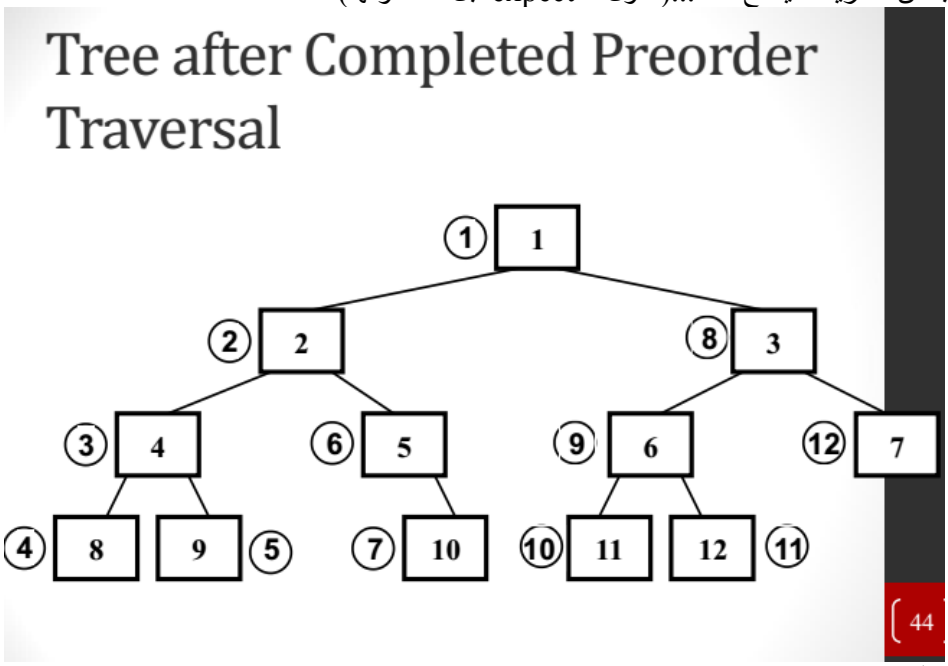
وبعدين اطلع للtree اللى فوقى اللى هى بتاعت ال2 انا قريت الrootوالleftيبقى نروح right



الnode دى جديدة وبقت root جديدة طالما ليها children يبقى كده بالترتيب اللى حفظناه انا هقرا الrootاللى هى ال5 وبعدين اروح left مفيش left يبقى اروح right فهقرأ ال10



والnode دى ملهاش children فخلاص .. هطلع بقى للtree اللى فوقى ال2 خلاص خلصناها بعدها ال1 عملنا فيها الroot والleft نعمل الright بقى بنفس الطريقة هيطلع كده ...(حاول تexpect قبل ما تشوفها)



نشوف الكود بتاعها بالمرة

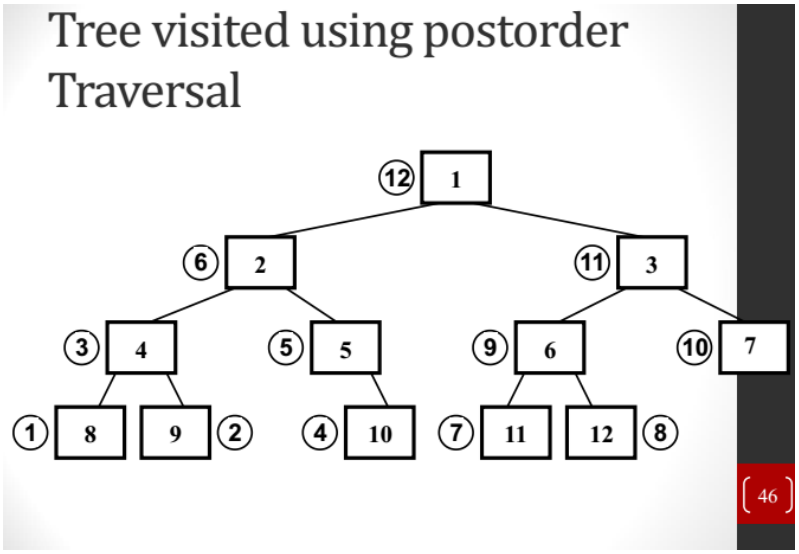
Binary Tree Traversals: preorder

```
typedef BinaryTree < int > btint;
typedef btint * btintp;
void preOrderTraverse(btintp bt)
{  if (!bt->isEmpty()) {  // if not empty
    // visit tree
    cout << bt->getData() << '\t';
   // traverse left child
    preOrderTraverse(bt->left());
    // traverse right child
    preOrderTraverse(bt->right());
  }
}
```

شايف اللى انا شايفاه؟
الfunction بتنده على نفسها !!.. ده الrecursion اللى خدناه زمان مع دكتور نعمت تقريبا .. المهم يعنى الدكتور قال نقدر نعمل ال function من غير recursion بس هتبقى طويلة و معقدة جدا .. واحنا عندنا الfunction بالrecursion اتعملت فى كام سطر بس نبقى حاطين condition صح عشان متبقاش زى الinfinite loop .

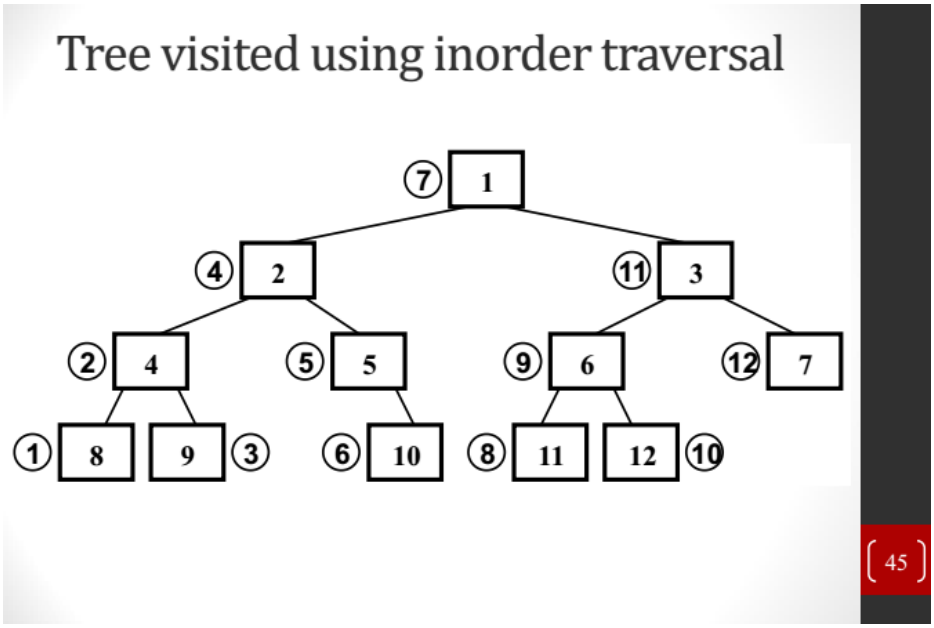
نحاول نفهم الكود بقى .. عرفنا tree الdata اللى فيها int الطريقة المكتوبة فى اول سطر دى طريقة كتابة الtemplate لو فاكرين... المهم وعملنا typrdef ل pointer من نوعها باسم btintp
الfunction ببعتلها الtree اللى هى مثلا الroot فى اول مرة
جوة الfunction بيطبع الdata اللى الptr مشاور عليها وبعدين تنده على نفسها تانى بس تبعت الleft بتاعها و كمان مرة وتبعت ال right بتاعها
ثانية واحدة .. يعنى كده مش بتمشى زى ما عملنا فوق .. احنا فوق مشينا left كتير وبعدين بقينا نبص على الright ؟؟
لا بتمشى زى ما عملنا عشان لما بتنده على نفسها تحط الleft هتبدأ الfunction تانى بالptr الجديد وتنده تانى نفسها الleft لحد ما الcondition اللى جوة الleft() ميتحققش هتروح للright بقى
طب وهطلع لفوق ازاى زى ما عملنا ؟
دلوقتى اخر left الassert اللى جواها وقفتنى .. فهخرج منها بpointer بيشاور على الtree اللى فوقى عادى ونكمل كده ..
فكر فيها بالراحة وطبقها على الtree اللى فوق هتفهمها بس لازم تعملها بنفسك مرة واحدة حتى عشان تفهم ماشية ازاى.

دلوقتى نشوف النوع التانى postorder اللى هو بيبدأ من الleft ثم الright ثم الroot



فهتلاقى فى السلايد اول node نروح نجيب الdata بتاعتها هى الnode اللى على الleft خالص
والتانية هى الright والتالتة الroot كده خلصنا الleft بالنسبة للnode اللى فيها اتنين نعمل الright بقى .. طب ليه خد الdata اللى تحت خالص الاول .. اللى هى 10؟؟.. عشان ال5 تعتبر root للـ10 واحنا مش بنقرا الroot الاول فى الtransverse ده … فهيقى الleft بتاع الخمسة اللى هو مش موجود … نروح للright اللى هو ال10 ثم الroot اللى هو الخمسة .. كده عملنا الleft والright بتوع الاتنين … نعمل الroot اللى هو الاتنين.
طب بالنسبة للواحد نفس الكلام .. احنا كده قرينا الleft بتوعه نقرا الright بقى قبل ما نقراه.
وهكذا

اخر نوع بقى الinorder traverse



ده اللى هو الleft ثم الroot ثم الright
ومفيش حاجة جديدة فى الinorder غير الترتيب لكن نفس الفكرة .

Inorder traversal

```
void inOrderTraverse(btintp bt)
{
  if (!bt->isEmpty()) {
    // traverse left child
    inOrderTraverse(bt->left());
    // visit tree
    cout << bt->getData() << '\t';
    // traverse right child
    inOrderTraverse(bt->right());
  }
}
```

49

عشان نحفظهم الاسم بيبقى حسب الrootفين يعنى الinorder اللى هو الrootبتتقرى بعد ما بنقرا الleftوقبل ما بنقرا الright .. فى النص

الpostorder الrootبتتقرى بعد الleftو الright

الpreorder الrootبتتقرى قبل الleft والright

وده الquiz اللى كان فى اخر المحاضرة

For the following tree, answer the following

1) The number of internal nodes in the tree are

a) 2 b) 8 c) 6 d) none of the above

2) The height of the tree is

a) 4 b) 3 c) 2 d) none of the above

3) If a postorder traverse is used, what is the order of accessing the root?

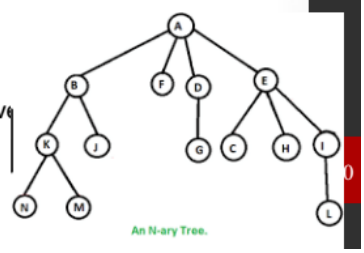
a) 1 b) 14 c) 3 d) none of the above

4) Is B node considered a tree?

a)True b) False

5) If a preorder traverse is used, what is the order of accessing the root?

a) 1 b) 14 c) 3 d) none of the above



An N-ary Tree.

كدة احنا خلصنا المحاضرة .. تعالى بقى نشوف ازاي بنكتب الكود بتاع الكلام اللي فوق دة كله ..

كالعادة بنبدأ بال file.h اللي بنحط فيه ال declarations بتاعتنا كلها .. وزي ما اتفقنا اننا كل حاجة بنعملها و هنعملها بعد كدة هتكون generic عن طريق ال templates .. فهنبدأ نعرّف ال class بتاعنا ونكتب ال member functions بتاعت ال public section : ((الدكتور هنا عامل حاجة كويسة .. هو كاتب ال user guide بتاع كل "function ال conditions وال return type ودي حاجة بتسهل الموضوع جدا على ال user))
** لو عايز تفتكر ال member functions اللي عايزين نعملها اطلع بص عليها في صفحة 12

Binary Tree Header File

```
template < class btElementType >
class BinaryTree {
public:
  BinaryTree();
  bool isEmpty() const;
  // Precondition: None.
  // Postcondition: None.
  // Returns: true if and only if T is an empty tree
```

getData( ), insert( )

```
btElementType getData() const;
  // getData is an accessor
  // Precondition: !this->isEmpty()
  // Postcondition: None
  // Returns: data associated with the root of the tree

void insert(const btElementType & d);
  // Precondition: none
  // Postconditions: this->getData() == d;     !this->isEmpty()
```

** ايه دة هو ايه const اللي محطوطة بعد isEmpty & getData دي؟؟ .. احنا مش متعودين عليها ..!
هقولك ياسيدي دي اسمها constant function وبتبقى for security بس مش اكتر (زي ما بنعمل مع ال variables اللي مش عايزينها تتغير في الكود) .. من الآخر كدة انا مش عايز ال functions 2 دول يغيروا حاجة في ال member variables بتاعتي (اللي انا بحميهم في ال private section)، فلو حصل بطريقةٍ ما ان ال functions دي غيرت حاجة في ال member variables واحنا قايلين لل compiler ان الـ function دي constant ، يبقى ساعتها ال compiler هيدينا error (((وخلي بالك كدة ال function ماتقدرش تغير حاجة في ال member variable آه، بس هي تقدر تـ access ال member variable وتقرا اللي جواه عادي جداً لأنها member function لنفس ال class .. فهي في الحقيقة ماغيرتش حاجة في ال function وبالتالي انا اقدر اشيل const دي خالص)))

left( ) and right( )

```
BinaryTree * left();
  // Precondition: !this->isEmpty()
  // Postcondition: None
  // Returns: (a pointer to) the left child of T

BinaryTree * right();
  // Precondition: !this->isEmpty()
  // Postcondition: None
  // Returns: (a pointer to) the right child of T
```

makeLeft( ), makeRight( )

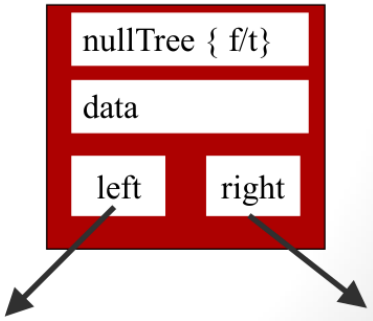
```
void makeLeft(BinaryTree * T1);
  // Precondition: !this->isEmpty();
       this->left()->isEmpty()
  // Postcondition: this->left() == T1

void makeRight(BinaryTree * T1);
  // Precondition: !this->isEmpty();
       this->right()->isEmpty()
  // Postcondition: this->right() == T1
```

لو كانت الدنيا ملخبطة معاك في شرح ال left & makeLeft اعتقد دلوقتي بعد ما شوفت سطرين الكود دول الدنيا احسن (((احنا في ال left بنقرا او بنبص على node موجودة في ال tree وعشان كدة ال return type بتاعنا هنا هو pointer to tree class .. بس ال makeLeft بنديها ال pointer اللي بيشاور على ال node اللي عايزين نزود عليها node جديدة في ال left branch، وعشان كدة هي مش محتاجة تـ return حاجة

كدة خلصنا ال public section وهندخل في ال private :
زي ما قولنا احنا عايزين ال node بتاعتنا يكون فيها مكان لل data اللي هتتخزن فيها ومحتاجين pointer يشاور على ال left child و pointer تاني يشاور على ال right child .. ودة اللي هنعمله دلوقتي :

Private Section

```
private:
  bool nullTree;
  btElementType treeData;
  BinaryTree * leftTree;
  BinaryTree * rightTree;
};
```



بس ايه null tree دة؟؟!
دة indicator كدة بيقولي هل ال tree (او ال sub tree .. متنساش :D) دي عندها children ولا لأ .. وهنحتاج ال indicator دة كتير قدام واحنا بنـ implement ال file.cpp

نكتب ال file.cpp بقى .. معلش المحاضرة طويلة بس خلاص مفيهاش كلام جديد :

Implementation file: Constructor

```
template < class btElementType >
BinaryTree < btElementType > :: BinaryTree()
{
  nullTree = true;
  leftTree = NULL;
  rightTree = NULL;
}
```

بديهي ان اي tree جديدة تكون فاضية في الاول

isEmpty( )

```
template < class btElementType >
bool
BinaryTree < btElementType > :: isEmpty() const
{
  return nullTree;
}
```

ال indicator العبيط اللي عملناه فوق خلّص لنا function كاملة .. لو مكناش عملنا ال indicator دة كنا هنحتاج نـ check لو ال node دي مكانش فيها داتا ونشوف هل ال left & right بيشاوروا على null ولا لأ

insert( )

```
template < class btElementType >
void BinaryTree < btElementType >
:: insert(const btElementType & d)
{
  treeData = d;
  if (nullTree) {
    nullTree = false;
    leftTree = new BinaryTree;
    rightTree = new BinaryTree;
  } }
```

انا هنا بشوف الأول هل ال node اللي عايز أ insert فيها دي null tree ولا لأ .. لو مش null tree فانا مش محتاج غير اني أ update الداتا اللي جوة ال node دي وخلاص
طب لو آه؟؟ .. لأ يبقى المفروض ال node دي ماتبقاش null tree خلاص لأن بقى فيها داتا .. فخلينا ال left pointer يشاور على tree جديدة (Null tree) وال right pointer يشاور على null tree جديدة برضه وبكدة ال node اللي كانت null tree مابقتش null tree

getData( )

```
template < class btElementType >
btElementType
BinaryTree < btElementType > :: getData() const
{
  assert(!isEmpty());
  return treeData;
}
```

left( )

```
template < class btElementType >
BinaryTree < btElementType > *
BinaryTree < btElementType > :: left()
{
  assert(!isEmpty());
  return leftTree;
}
```

right( )

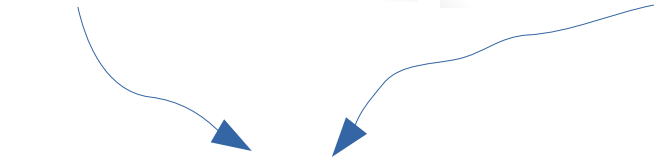
```
template < class btElementType >
BinaryTree < btElementType > *
BinaryTree < btElementType > :: right()
{
  assert(!isEmpty());
  return rightTree;
}
```

makeLeft( )

```
template < class btElementType >
void BinaryTree < btElementType >
:: makeLeft(BinaryTree * T1)
{
  assert(!isEmpty());
  assert(left()->isEmpty());
  delete left();  // could be nullTree true, w/data
  leftTree = T1;
}
```

makeRight( )

```
template < class btElementType >
void BinaryTree < btElementType >
:: makeRight(BinaryTree * T1)
{
  assert(!isEmpty());
  assert(right()->isEmpty());
  delete right();
  rightTree = T1;
}
```



احنا هنا كنا عايزين نزود node جديدة على الشمال او اليمين فكنا محتاجين نشوف الاول هل ال node اللي عايزين نزود عليها دي null ولا لأ .. لو مش null يبقى فيه حاجة مش طبيعية في الموضوع فهنخرج برة ال function من غير مانعمل حاجة
طب لو آه؟؟ ساعتها هنـ assert تاني ونشوف هل ال left node دي null ولا لأ .. المفروض تبقى null .. بس احنا مش عايزينها تبقى null دلوقتي .. فهنـ delete ال null ونحط ال pointer to node الجديد اللي عايزين نحطه

كدة الحمد لله ربنا كرمنا وخلصنا ال file.h & file.cpp .. المحاضرة الجاية بقى ان شاء الله هنكمل ونشوف ال client program ممكن يبقى عامل ازاي

من عرف ما يطلب هان عليه ما يبذل ^_^