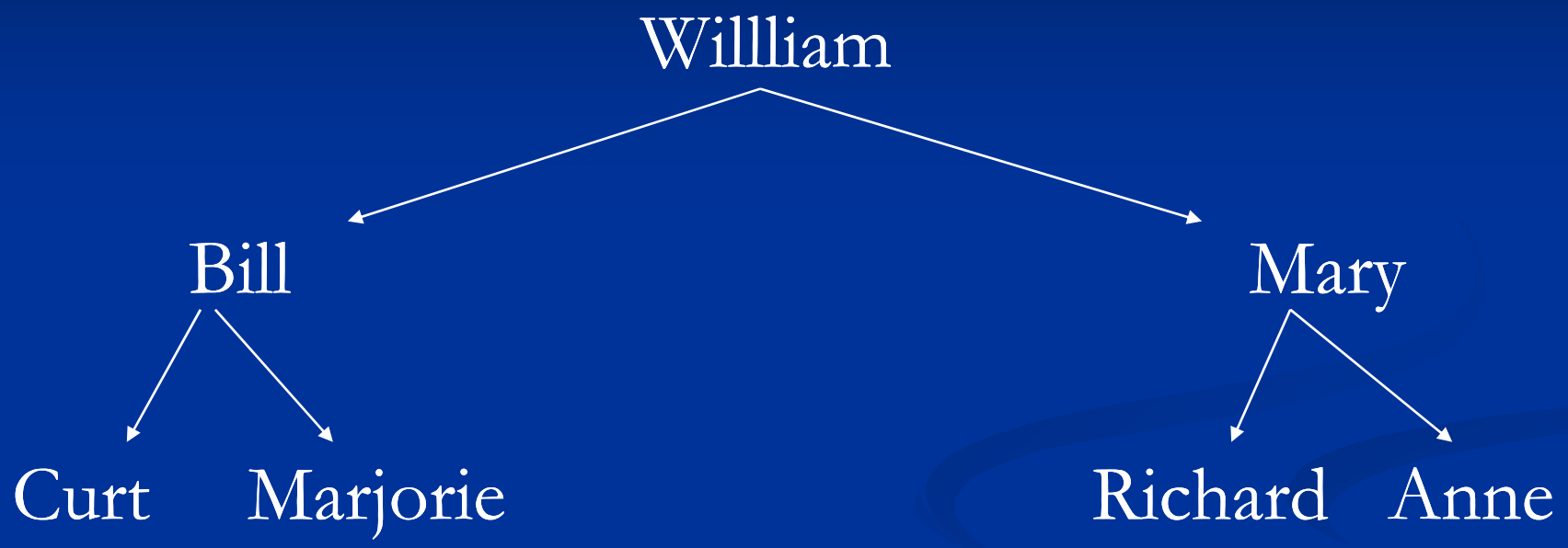
درختان

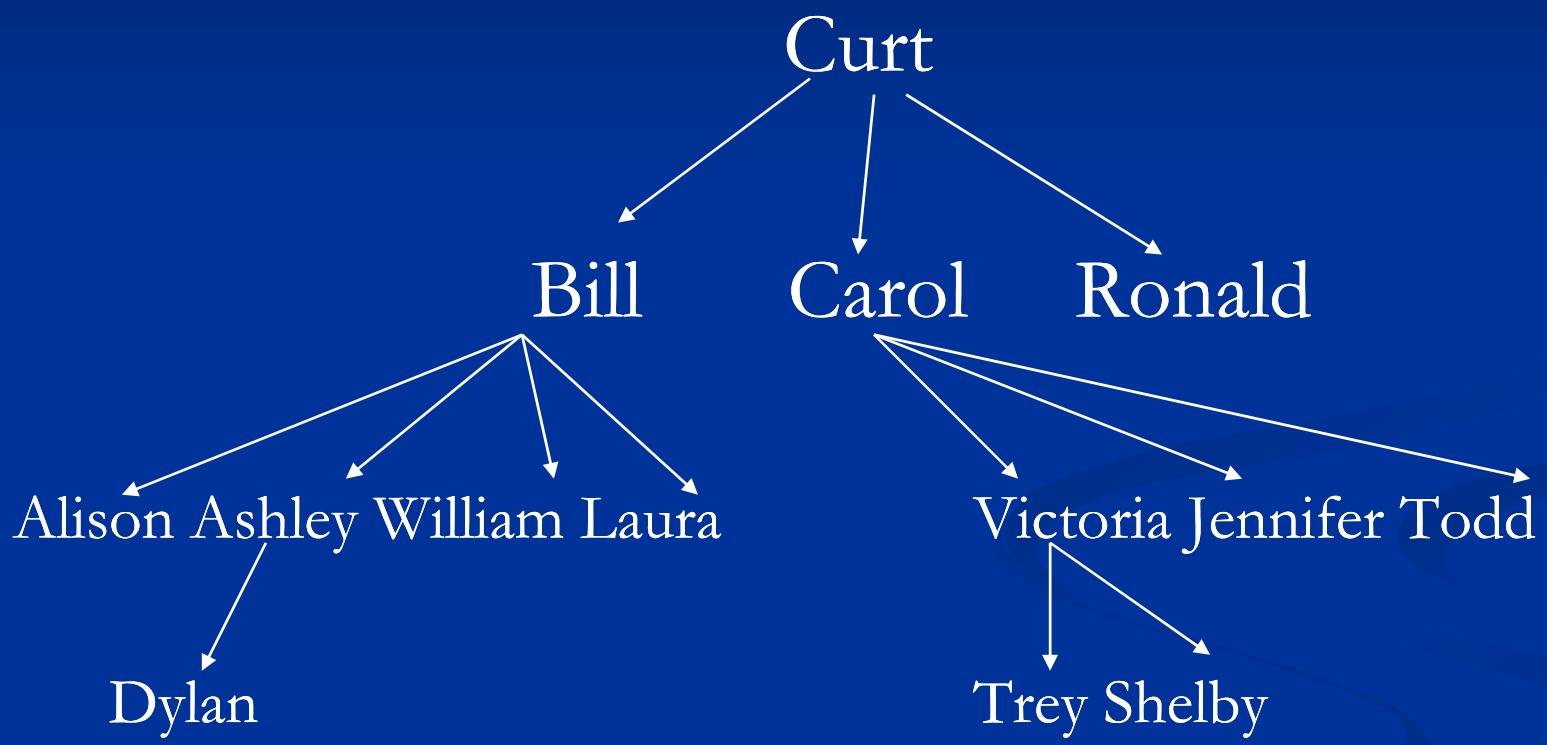
مثال



آیتمها با شاخه ها به هم وصل شده اند

معنای شاخه: والد

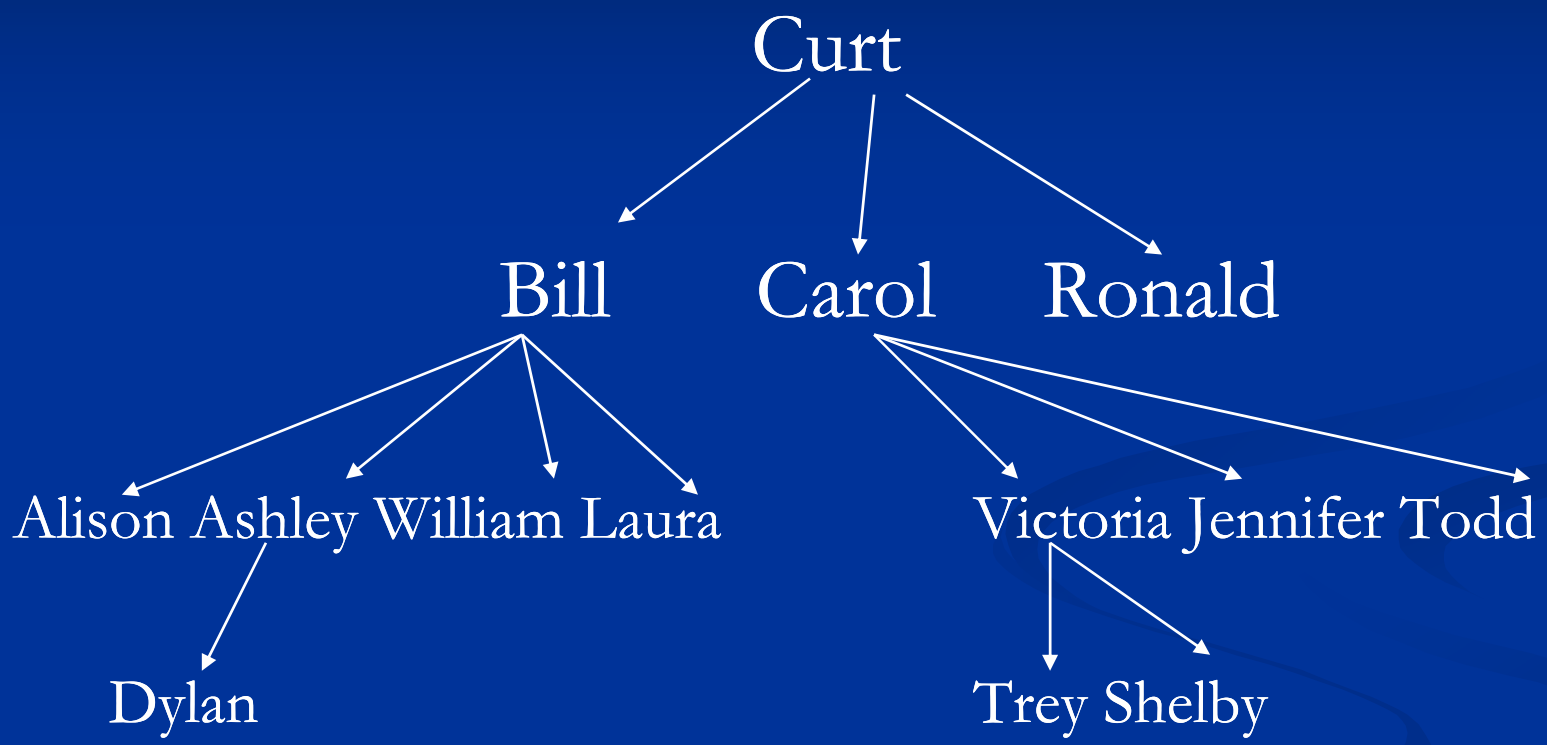
مثال


Curt
Bill
Carol
Ronald
Alison
Ashley
William
Laura
Victoria
Jennifer
Todd
Dylan
Trey
Shelby

تعریف درخت

- درخت یک مجموعه محدود از نودها است که:
  - یکی از نودها ریشه درخت است
  - باقی نودها به $n >= 0$ مجموعه مجزای T1, T2, .. Tn تقسیم می شوند بطوریکه هر کدام یک درخت هستند. T1, T2, … Tn زیر درختهای ریشه نامیده میشوند.

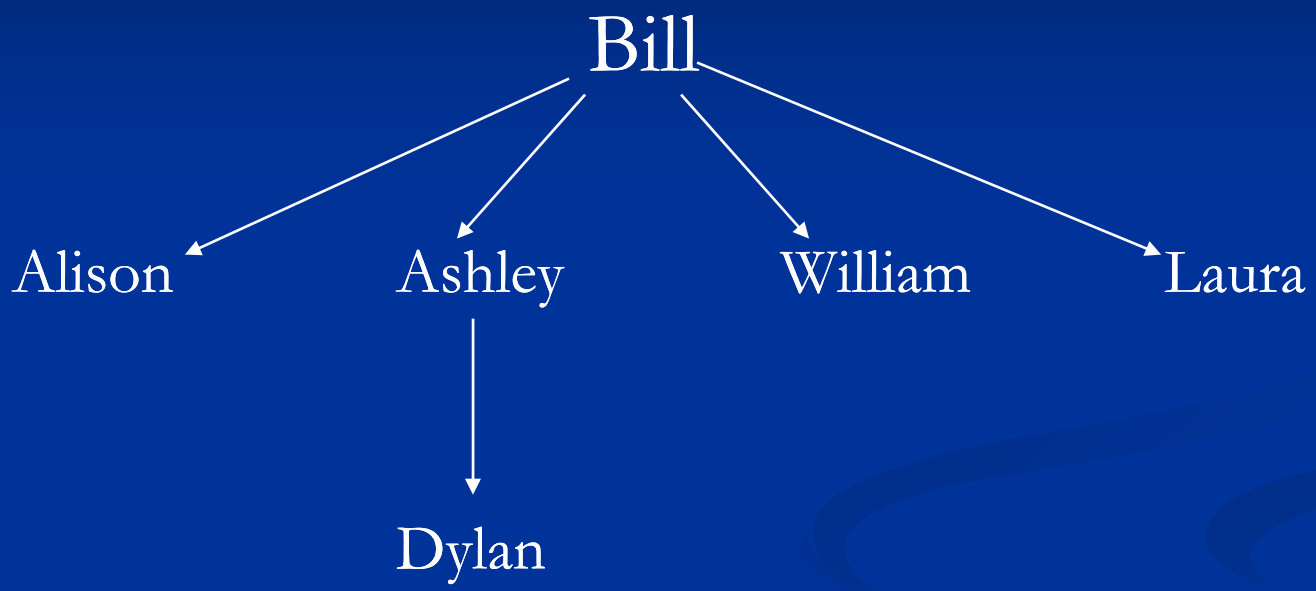
مثال



ریشه: Curt

Bill یک زیر درخت Court است که خود نیز یک درخت هست.

ادامه مثال



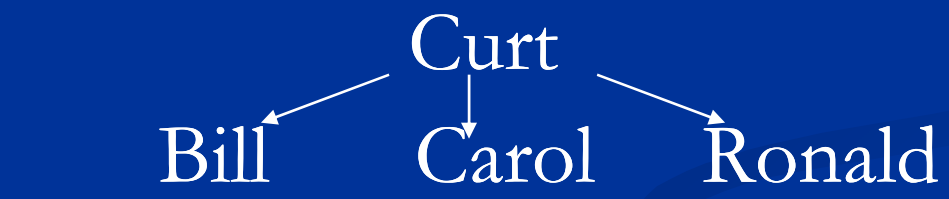
ریشه: Bill

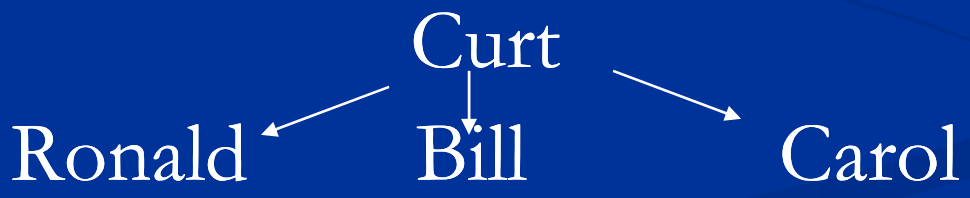
Ashleyیک زیر درخت Bill است که خود نیز یک درخت هست.

Dylanیک زیر درخت Ashley است که خود نیز یک درخت هست.

تعریف درخت

- ترتیب زیر درختها اهمیتی ندارد
- دو درخت زیر هم ارزند.



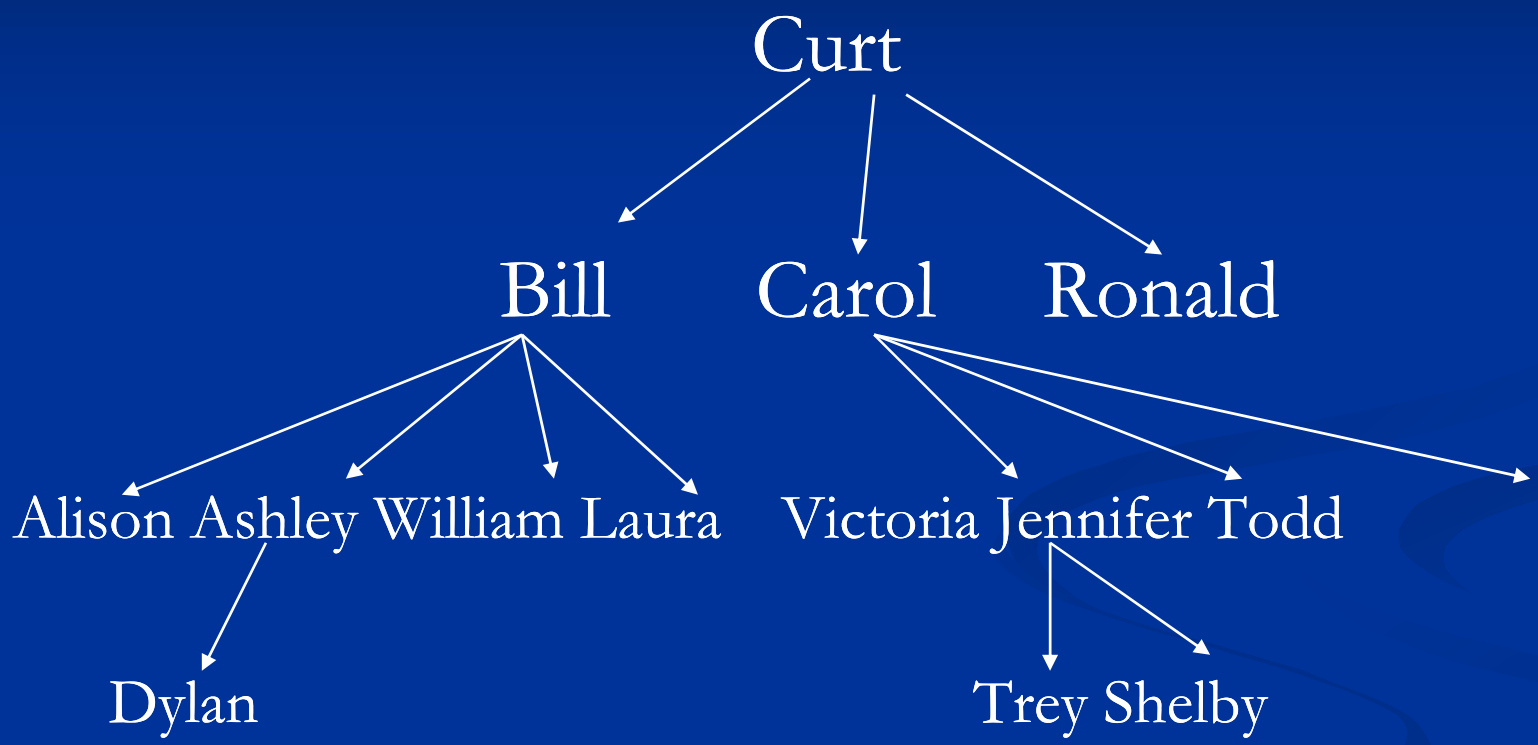


تعاریف درخت

- تعریف درخت با تعریف لیستهای پیوندی مشابه است

- ***درجه*** هر نود تعداد زیر درختهای آن نود است
- به نودهای که دارای درجه صفر هستند ***برگ*** گفته می شود
- به نودهای دارای درجه بزرگتر از صفر، ***نودهای غیر انتهایی*** گفته می شود.

مثال



درجه(Curt)=3 درجه(Bill)= 4 درجه(Ashley)= 1 درجه(William)= 0

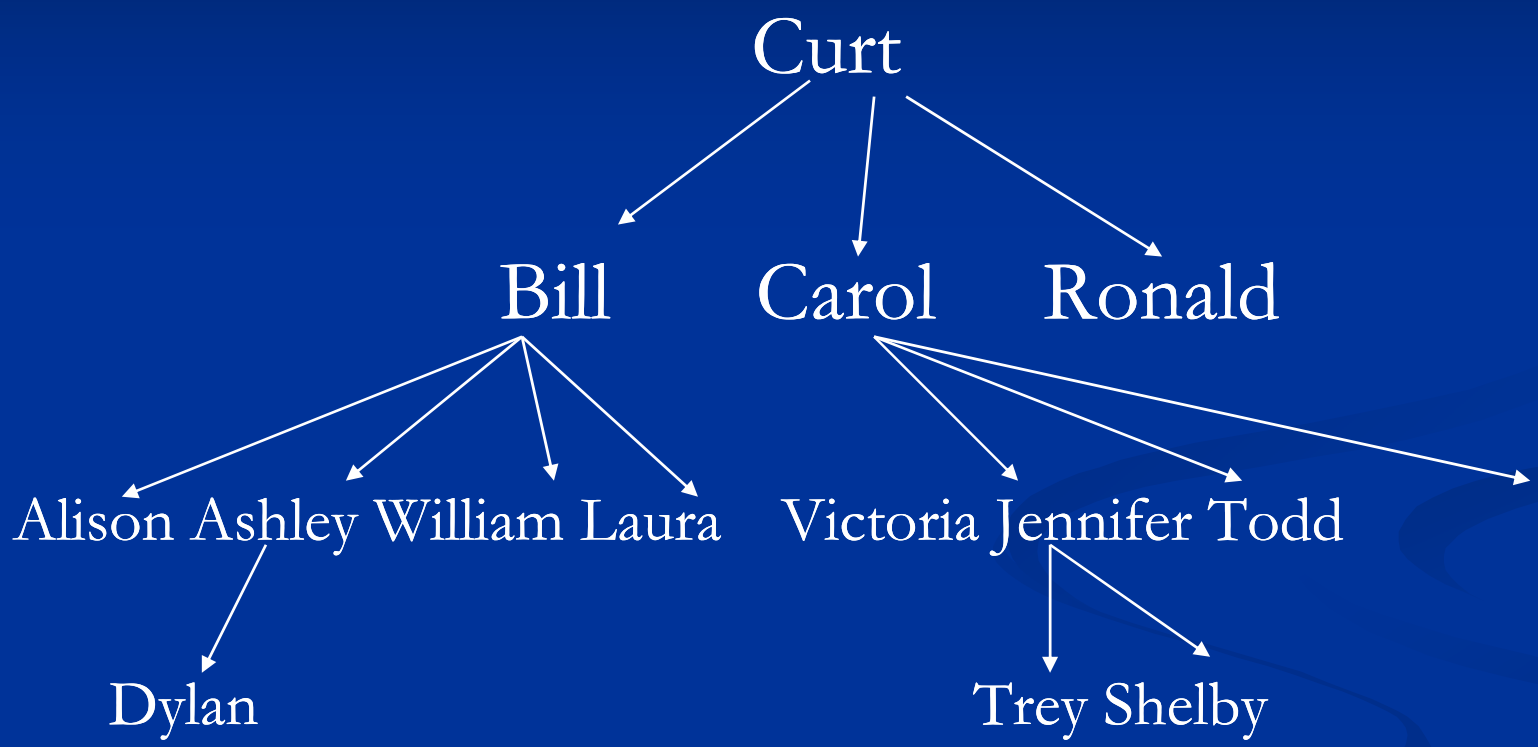
نودهای انتهایی: Alison, Dylan, William, Laura, Trey, Shelby,

Jennifer, Todd, Ronald

تعاریف درخت

- به ریشه های زیر درختان ***فرزند*** گفته می شود
- ***والد*** نودی است که یک یا چند فرزند دارد
- فرزندان یک والد ***همزاد*** هستند.

مثال



Bill, Carol, Ronald :Court فرزندان

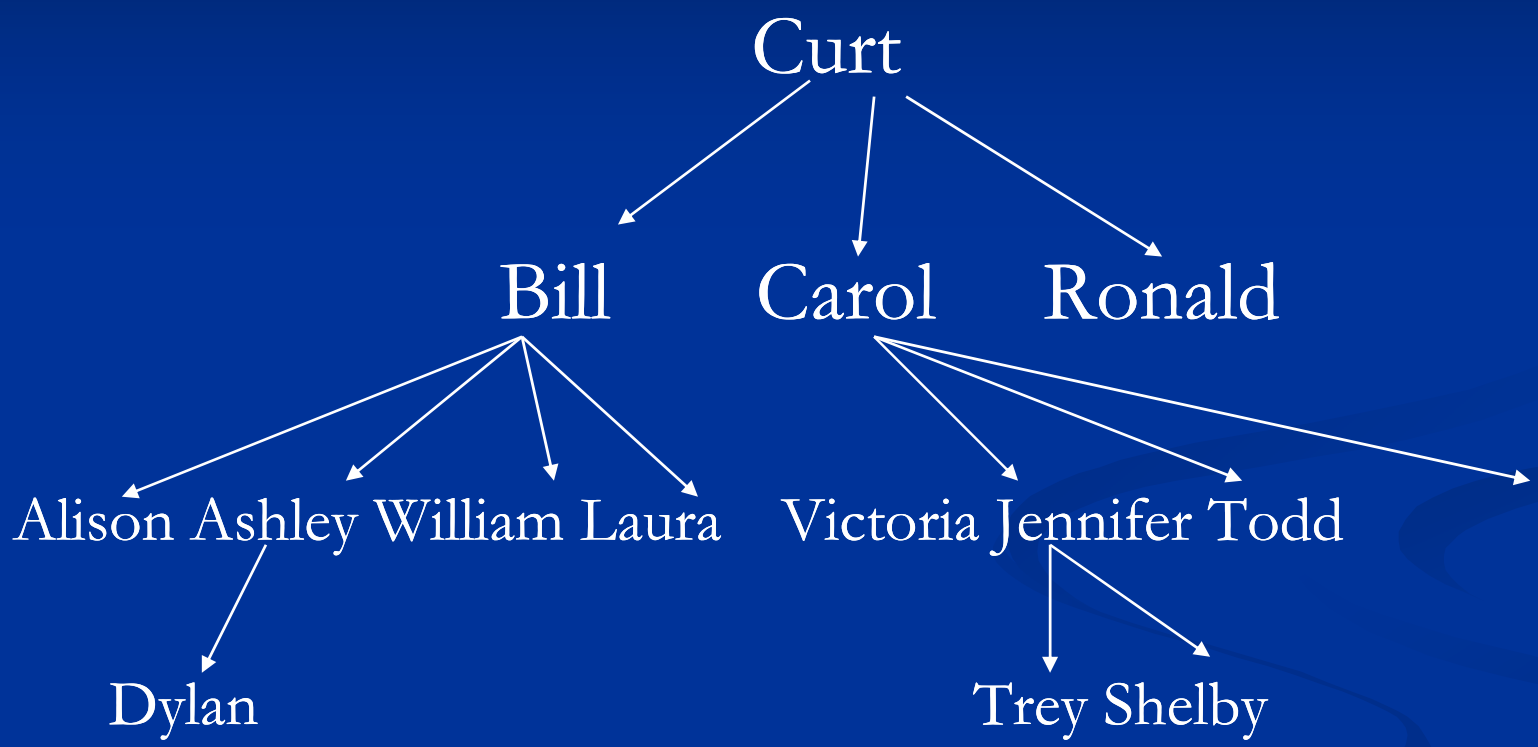
Carol :Jennifer والد

Alison, William, Laura :Ashley همزادهای

تعاریف درخت

- به والد والد، ***جد*** گفته می شود.
- تمام نودهایی که روی مسیر ریشه به سمت یک نود قرار دارند، ***نیاکان*** آن نود هستند.
- به تمام نودهایی که در زیر درخت مربوط به یک نود قرار دارند، ***اعقاب*** آن نود گفته می شود.

مثال



Ashley, Bill, Curt = (Dylan) نیاکان

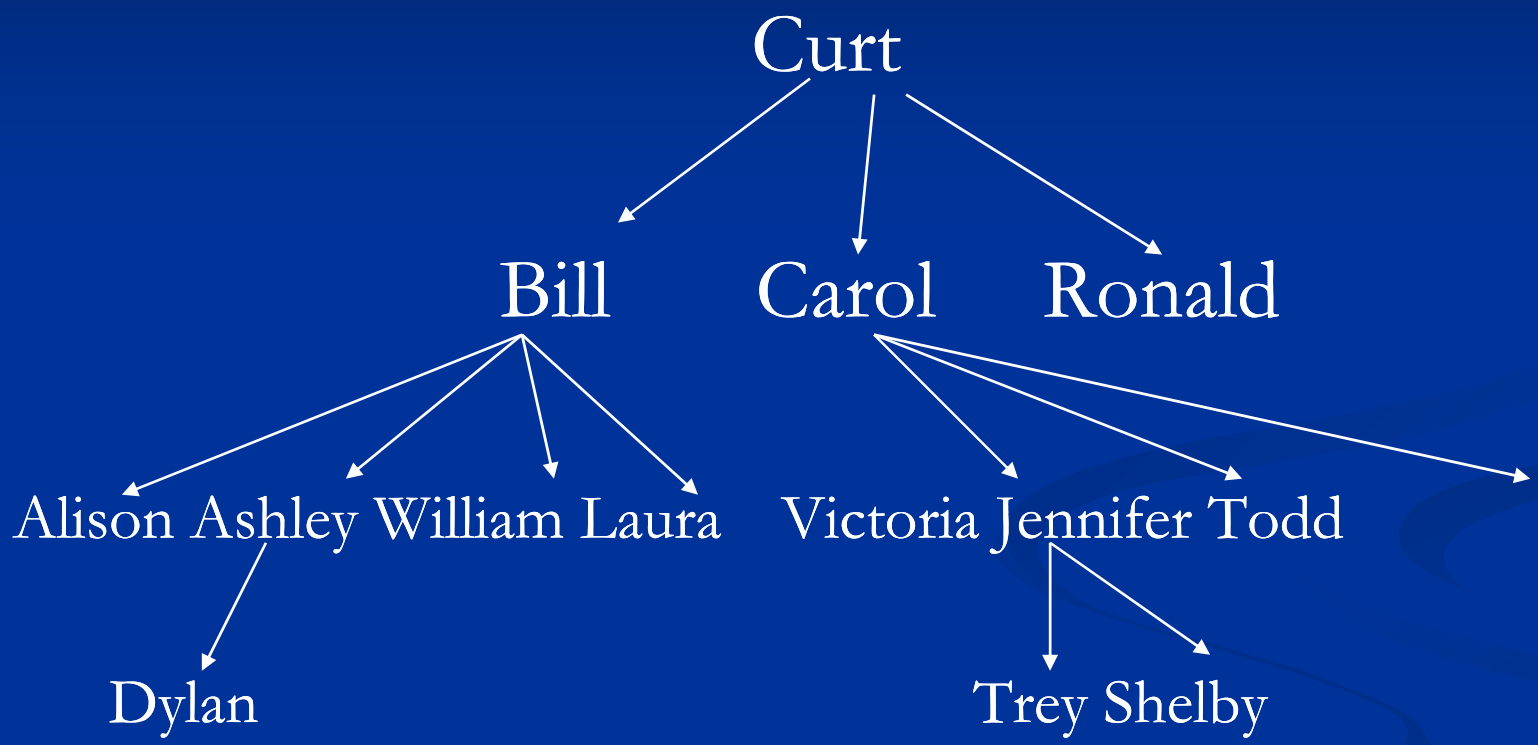
Bill = (Dylan) جد

Alison, Ashley, William, Laura, Dylan = (Bill) اعقاب

تعاریف درخت

- ***درجه درخت:*** حداکثر درجه نودهای درخت
- ***سطح نود:***
  - سطح ریشه = ۱
  - سطح نودهای غیر ریشه= تعداد لینکهای بین نود و ریشه + ۱
- ***ارتفاع*** یا ***عمق*** درخت: حداکثر سطح نودهای درخت

مثال



درجه درخت = 4(به خاطر Bill)

سطح Curt = 1 ، سطح Dylan = 4 ، سطح Ashley = 3

ارتفاع درخت = 4(به خاطر Dylan)

Recursion and Trees

- دقت کنید که تعریف درخت بازگشتی است.
  - فرزندان یک نود ریشه های زیردرخت خودشان هستند.
- اکثر اعمالی که روی درختان انجام می شوند نیز بازگشتی هستند.
  - محاسبه ارتفاع
  - پیمایش درخت
  - جستجوی یک نود خاص

Tree Representations

- ایجاد یک ساختار داده برای درخت:

- راه حل اول:
  - استفاده از نود لیست پیوندی: هر نود برای هر فرزند دارای یک اشاره گر باشد.

- ملاحظات:
  - برای تمام فرزندان به اشاره گر نیازمندیم.
  - اگر تعداد فرزندان نودها مثل هم نباشد چکار کنیم؟

Tree Representations

- اگر ماکسیمم تعداد فرزندان نودهای درخت برابر k باشد، هر نود باید دارای k اشاره گر باشد.

Node for tree with max degree 4

DATA	CHILD 1	CHILD 2	CHILD 3	CHILD 4

- تعداد زیادی از این اشاره گر ها هدر خواهند رفت:
  - $nk - n + 1$ اشاره گر بدون استفاده
  - اگر درخت پر باشد این روش مناسب است.

Tree Representation

- اثبات:
  - n نود که هر کدام k اشاره گر دارند، در نتیجه nk اشاره گر داریم.
  - به سمت هر نود غیر از ریشه دقیقا یک اشاره گر داریم. پس n-1 اشاره گر را استفاده می کنیم.
  - تعداد اشاره گر های بلا استفاده: $nk - (n - 1) = nk - n + 1$

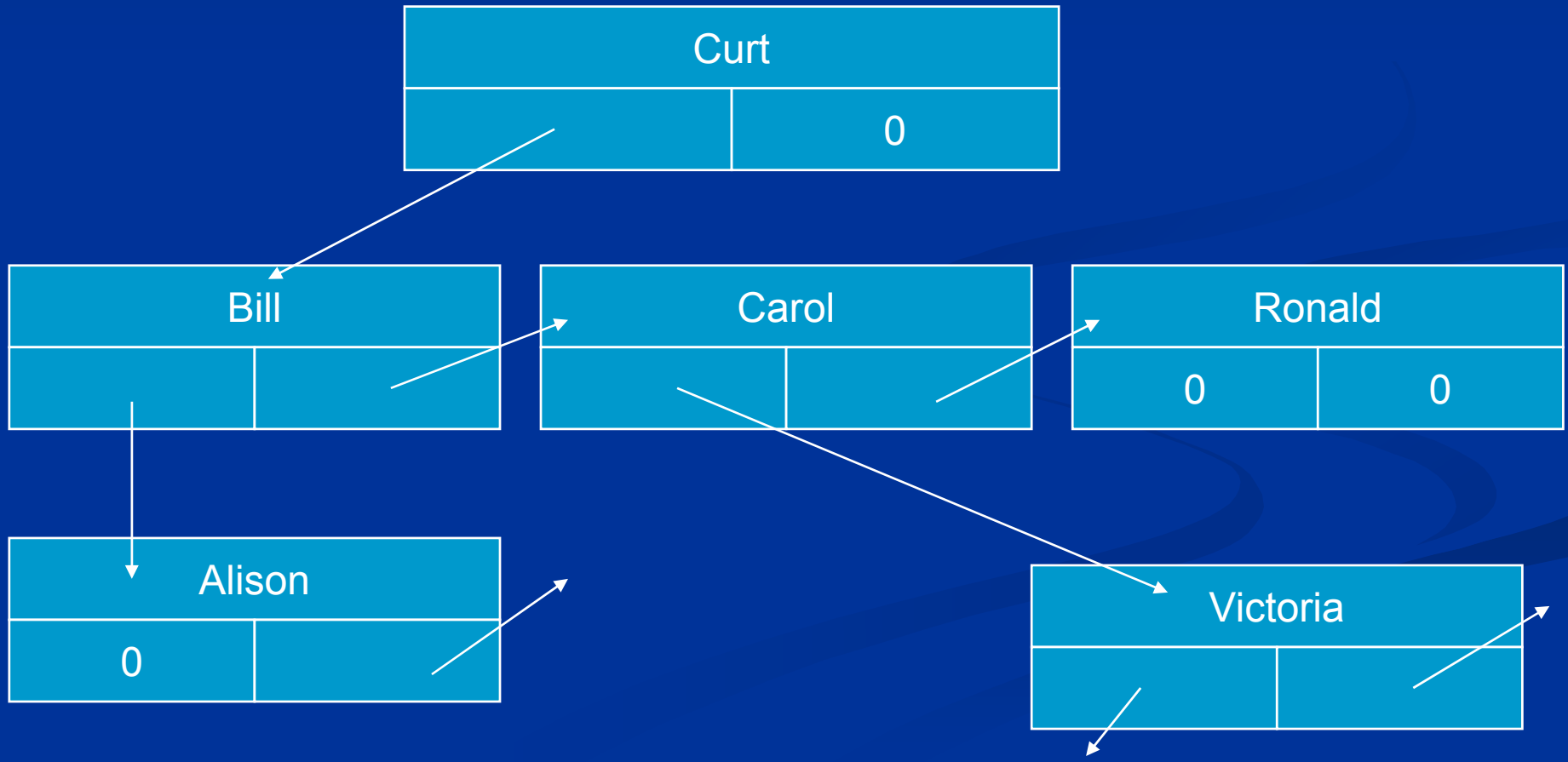
- مثال:
  - درخت فامیلی: 4-ary, 14 nodes
  - $56 - 14 + 1 = 43$ unused pointers
    (172 bytes wasted)
  - اکثر اشاره گر های اتلاف شده مربوط به برگها است.

Tree Representation

- نمایش درخت با دو اشاره گر برای هر نود
- فرزند سمت چپ و همزاد سمت راست

Tree Representation

Left Child, Right Sibling Family Tree

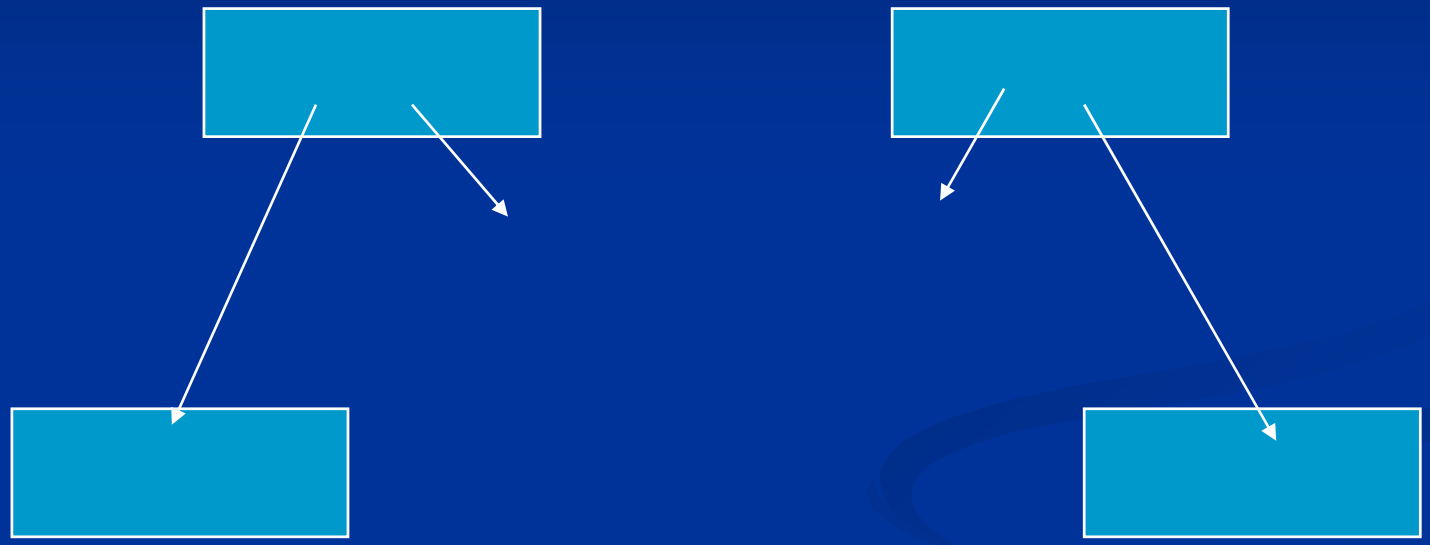


Specialized Trees

- درخت دودویی:
  - در این درخت درجه هر نود حداکثر دو است.
  - ترتیب نودها اهمیت دارد.
  - ممکن است دارای صفر نود باشد.

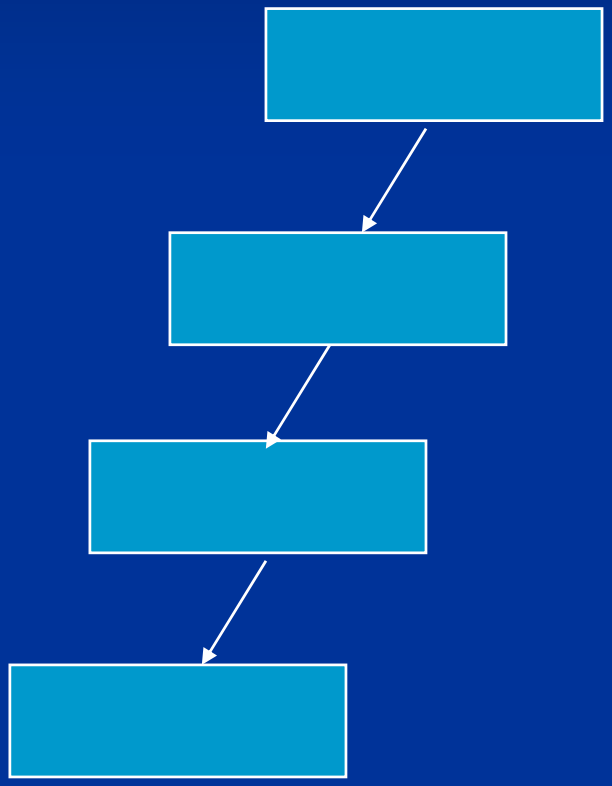
- تعریف رسمی:
  - یک درخت باینری مجموعه محدودی از نودها است که یا خالی است یا شامل ریشه و دو زیر درخت متمایز دودویی است که به آنها زیر درختان سمت راست و سمت چپ گفته می شود.

Specialized Trees

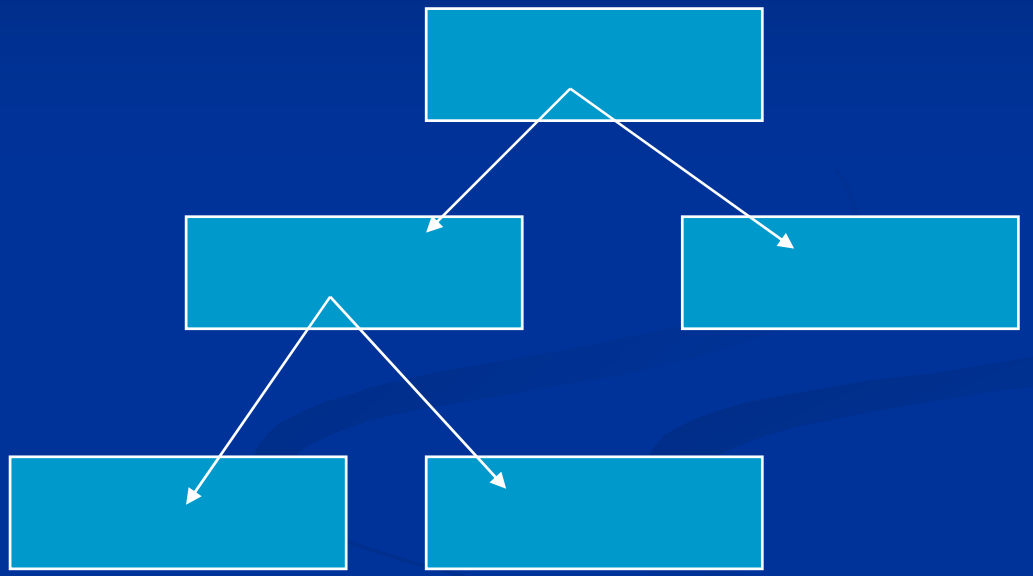


اینها دو درخت معادل هستند. اما دو درخت دودویی معادل نیستند.

Types of Binary Trees



خطی – نامتعادل



کامل – متعادل تر

Properties of Binary Trees

- خواص جالب درخت دودویی:
  - حداکثر تعداد نودها در سطح l برابر 2^{l-1} است.
  - حداکثر تعداد نودها در درخت باینری با ارتفاع k ، برابر $2^k - 1$ است.

Proofs

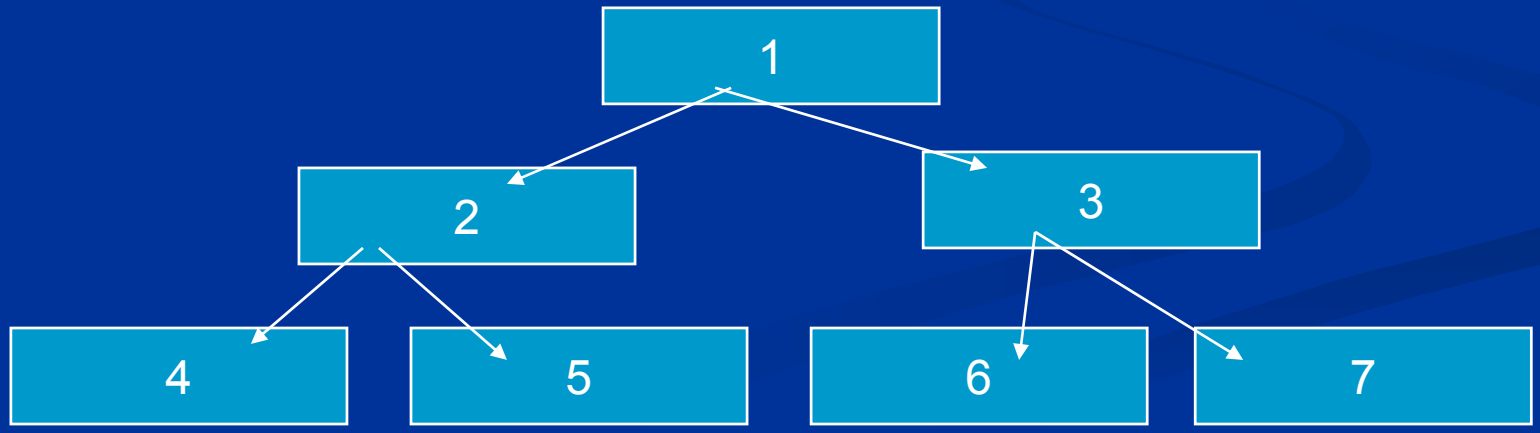
- حداکثر تعداد نودها در سطح l برابر 2^{l-1} است.
- اثبات با استفاده از استقراء:
  - حالت پایه:
    - Level 1 = $2^{1-1} = 2^0 = 1$
  - برای یک $n > 1$
    - اگر سطح n دارای 2^{n-1} نود باشد.
  - برای n+1 :
    - هرنود در سطح n ، حداکثر دو فرزند در سطح n+1 دارد. پس تعداد نودهای سطح n+1 حداکثر برابر $2^{n-1} * 2 = 2^n$ است.

Proofs

- حداکثر تعداد نودها در درخت باینری با ارتفاع k ، برابر $2^k - 1$ است.
- می توان ماکسیمم تعداد نودهای هر سطح را با هم جمع زد.
- باید حاصل جمع $1+2+4+\ldots+2^{k-1}$ را بدست آوریم.
- این جمع برابر است با: $2^k - 1$

Properties of Binary Trees

- یک درخت باینری پر با عمق K ، دارای $2^k - 1$ نود هست
  - تمام نودهای غیر انتهایی دارای دو فرزند هستند.
  - نودهای انتهایی در آخرین سطح قرار دارند.



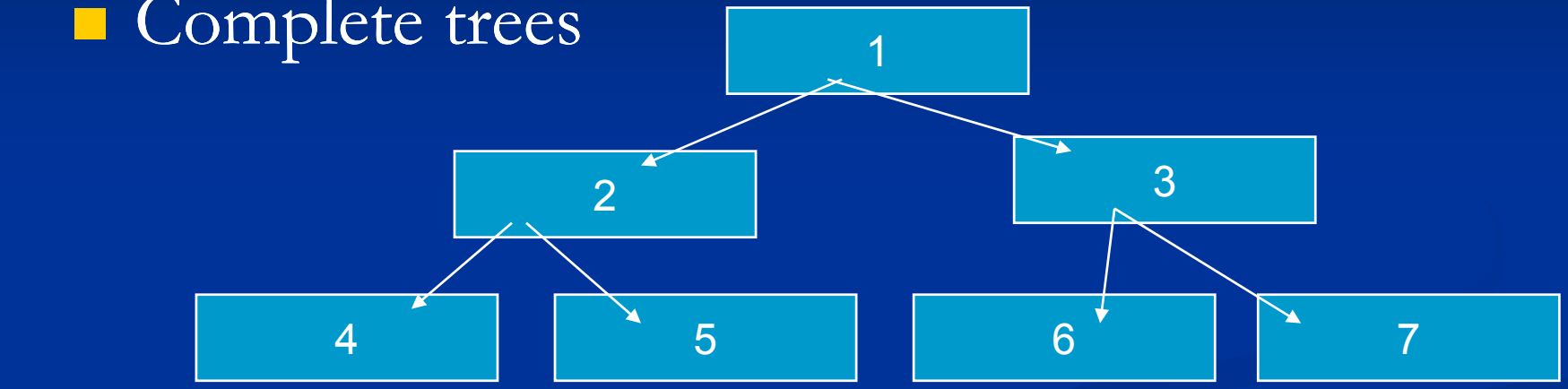
Properties of Binary Trees

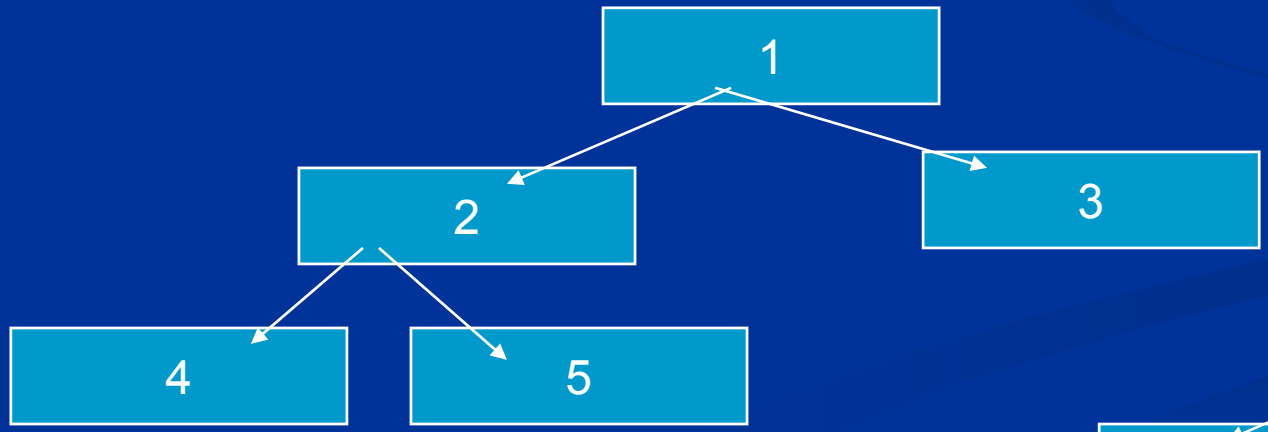
- می گوییم یک درخت باینری با n نود یک درخت کامل است اگر نودهای آن با درخت پری که نودهای آن از ۱ تا n شماره گذاری شده اند، متناظر باشد.

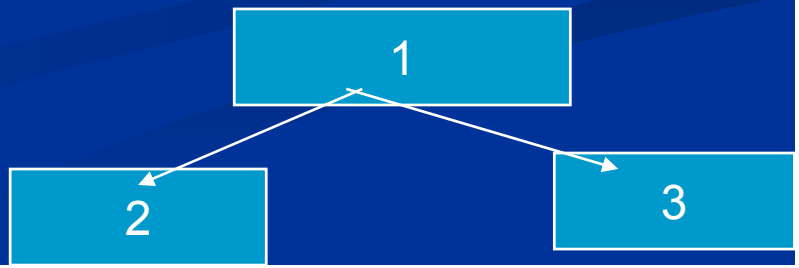
- ارتفاع یک درخت دودویی کامل با n نود برابر است با:
$(\log_2(n+1))$

Properties of Binary Trees

- Complete trees

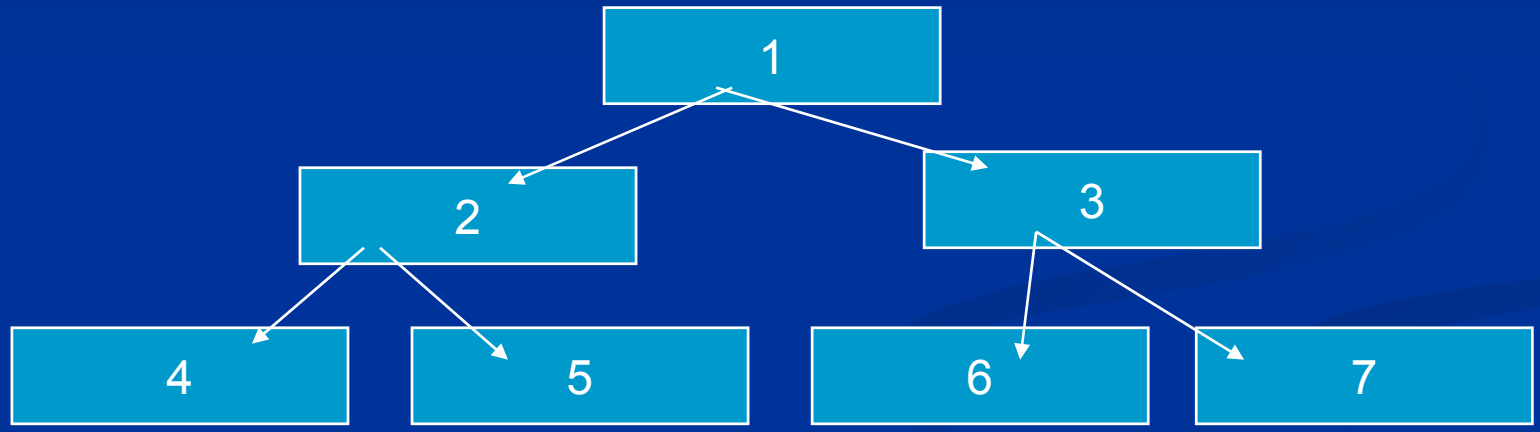






Binary Tree Implementation

- پیاده سازی آرایه ای



- هر نود با یک عنصر از آرایه متناظر می گردد

Binary Tree Implementation

- پیاده سازی آرایه ای:
  - parent(i) در محل i/2 قرار دارد (برای i!=1).
  - فرزند سمت چپ i در محل 2i و فرزند سمت راست در محل 2i+1 قرار دارد.
  - اگر 2i>n باشد، i فرزند سمت چپ ندارد.
  - اگر 2i+1>n باشد، i فرزند سمت راست ندارد.

Array Index: 0 1 2 3 4 5 6 7

خالي	Data1	Data2	Data3	Data4	Data5	Data6	Data7

Binary Tree Implementation

- مثل قبل، نمایش آرایه ای بهترین راه حل نیست.
  - اندازه حافظه ثابت است:
    - براحتی قابل گسترش نیست.
    - اگر درخت بالانس نباشد، حافظه هدر می رود.

- راه بهتر: استفاده از ایده لینک پیوندی.
  - می شود از راه حل استفاده از همزاد دوری کرد، چون تعداد فرزندان ثابت است و فقط به دو اشاره گر نیازمندیم.

Binary Tree Node

```
class BinaryTree; // اعلان پیش از موقع
class BinaryTreeNode
{
   friend class BinaryTree;
   private:
        char data;
        BinaryTreeNode* leftChild;
        BinaryTreeNode* rightChild;
};
```

Binary Tree Class

```
class BinaryTree
{
   public:
        // public member methods
   private:
        BinaryTreeNode* root;
};
```

Binary Tree Node

- تنها مشکل نمایش پیوندی این است که به پدر دسترسی نداریم.

- اشکالی ندارد:
  - بسیاری از الگوریتمها نیازی به دانستن پدر ندارند.
  - اگر هم مهم باشد، وقتی که درخت را پیمایش می کنیم، یک اشاره گر به پدر را نگهداری می کنیم.
  - اگر خیلی مهم باشد، می توان به تعریف نود اشاره گر پدر را اضافه نمود.

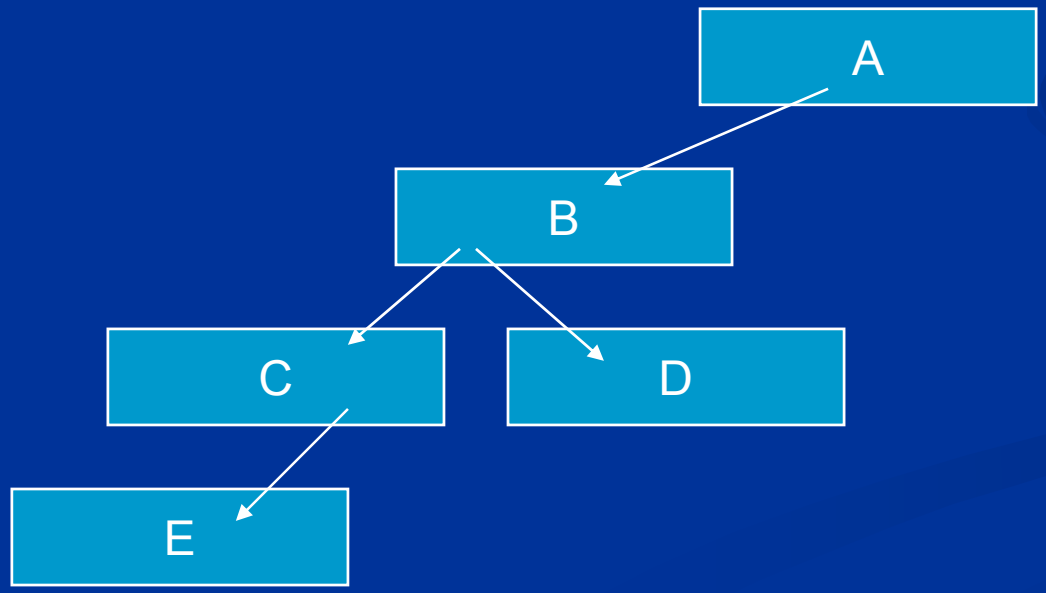
Questions

- در درخت باینری زیر:
  - نودهای انتهایی کدامند؟
  - نودهای غیر انتهایی کدامند؟
  - سطح هر نود چند است؟

D,E

A,B,C

A=1,B=2,CD=3,E=4



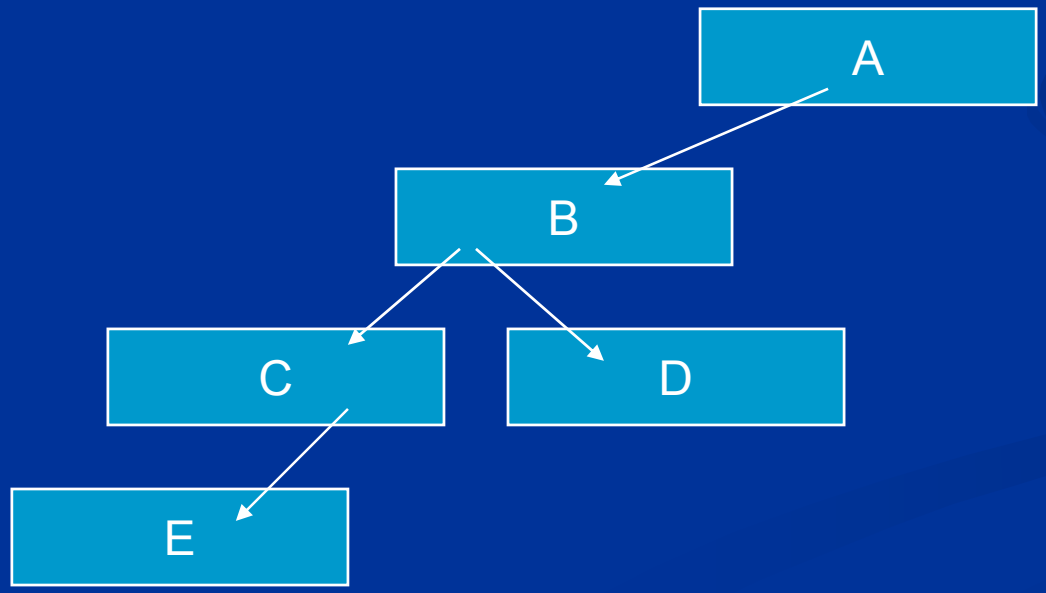
Questions

- حداکثر تعداد نودها در یک درخت k-ary با ارتفاع h چند است؟
  - برای k=3 داریم : $1 + 3 + 9 + 27 + 81 \ldots$
  - برای k=4 داریم :$1 + 4 + 16 + 64 \ldots$
  - برای k=k داریم :$1 + k + k^2 + k^3 \ldots$
- مجموع فوق برابر است با: $(k^h\text{-}1) / (k\text{-}1)$
- اگر h=4 و k=3 :
  - $(3^4 - 1) / (3\text{-}1) = 80/2 = 40$

Questions

نمایش آرایه ای درخت زیر چیست؟

خالی	A	B	خالی	C	D	خالی	خالی	E	خالی



Binary Tree Class

```
class BinaryTree
{
   public:
        BinaryTree(); // create empty tree
        BinaryTree(char data, BinaryTree bt1, BinaryTree bt2);
        //construct a new tree , setting root data to data and links to
        //other trees
        bool isEmpty(); // test whether empty
        BinaryTree leftChild(); // get left child of *this
        BinaryTree rightChild(); // get right child of *this
        char Data(); // return data in root node of *this
};
```

Binary Tree Functions

- تمام عملگرهای که روی درخت باینری کار می کنند نیازمند حرکت روی درخت هستند:
  - دیدن نودها
  - اضافه کردن نود
  - حذف یک نود
  - محاسبه ارتفاع بصورت غیر بازگشتی
- لذا داشتن یک مکانیسم برای پیمایش درخت بسیار مفید خواهد بود.

Binary Tree Traversal

- پیمایش درخت
  - هر نود فقط یک بار دیده شود.
  - تمام نودها دیده شوند.
  - یک یا چند عملگر روی درخت اجرا شود:
    - چاپ داده
    - جمع با حاصل جمع
    - چک کردن حداکثر ارتفاع
  - هر پیمایش یک ترتیب خطی از همه نودها را تولید خواهد کرد.

Binary Tree Traversal

- با درختان و زیر درختان به یک صورت برخورد می کنیم:
  - با ترتیب مساوی پیمایش را انجام می دهیم
  - از بازگشت استفاده می کنیم.

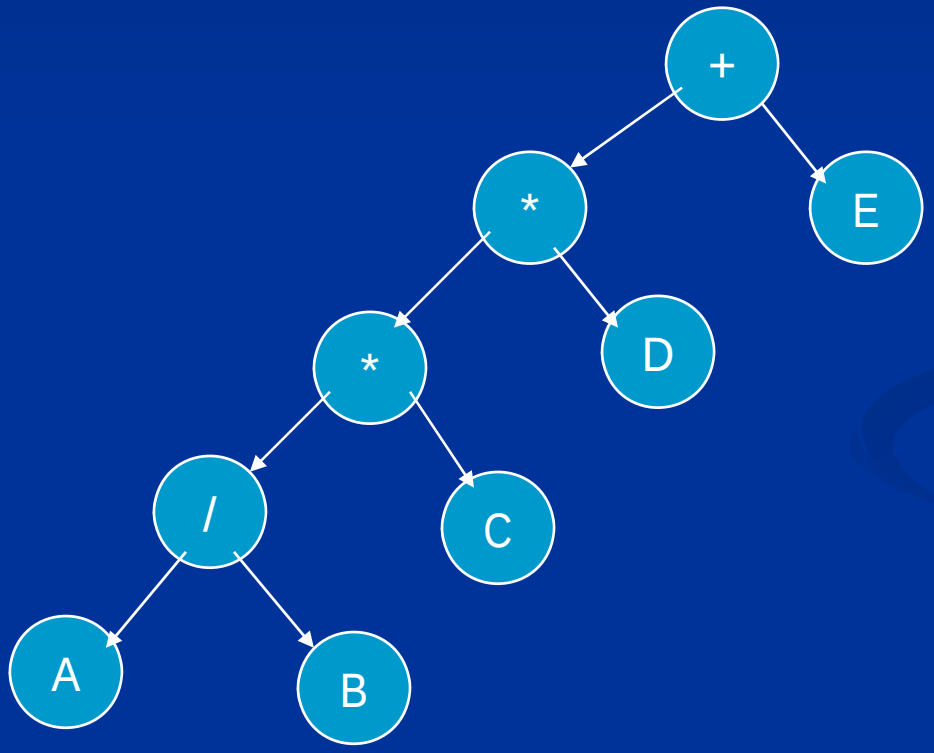
- فرض کنید برای پیمایش از حروف زیر استفاده کنیم:
  - L یعنی حرکت به فرزند سمت چپ
  - R یعنی حرکت به فرزند سمت راست
  - V یعنی دیدن نود (یا انجام عمل مورد نظر)

Binary Tree Traversal

- شش حالت امکان پذیر است:
  - LVR, LRV, VLR, VRL, RVL, RLV
- ما فقط حالتهایی را که L قبل از R آمده است را مورد توجه قرار می دهیم:

LVR	LRV	VLR
Inorder	Postorder	Preorder
به ترتیب	پس ترتیب	پیش ترتیب

Binary Tree Traversal



Inorder: LVR

A / B * C * D + E

عبارت ميانوندي
ديدن سمت چپ پيش از پدر

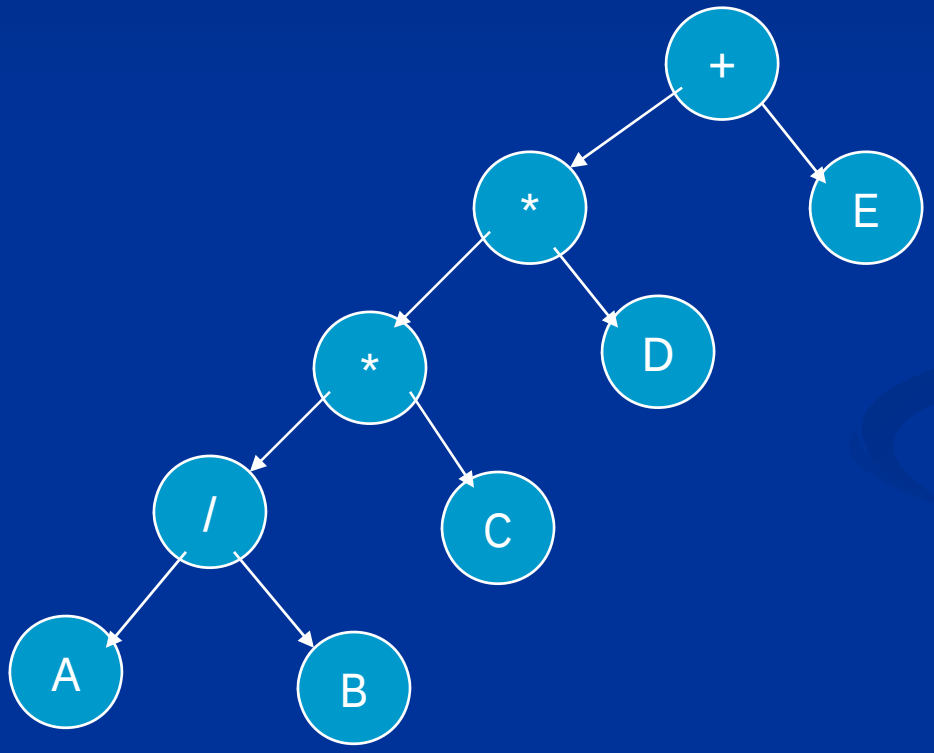
Binary Tree Traversal

- پیاده سازی Inorder:

```
void BinaryTree::inorder()
{
    inorder(root);
}

Void BinaryTree::inorder(BinaryTreeNode* node)
{
    if (node)
    {
        inorder(node->leftChild);
        cout << node->data;                 // replace cout with
        inorder(node->rightChild);          //arbitrary processing
    }
}
```

Binary Tree Traversal



Postorder: LRV

A B / C * D * E +

عبارت پسوندی

دیدن سمت چپ و راست پیش از پدر

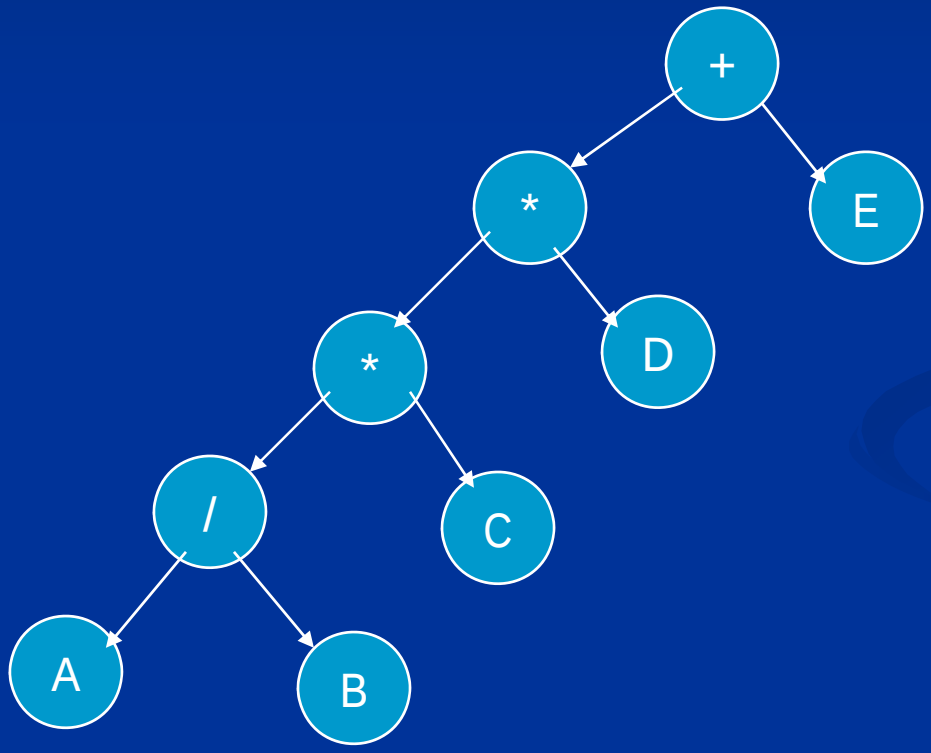
Binary Tree Traversal

- پیاده سازی postorder:

```
void BinaryTree::postorder()
{
    postorder(root);
}

void BinaryTree::postorder(BinaryTreeNode* node)
{
    if (node)
    {
        postorder(node->leftChild);
        postorder(node->rightChild);
        cout << node->data;
    }
}
```

Binary Tree Traversal



Preorder: VLR

+ * * / A B C D E

عبارت پیشوندی
دیدن پدر پیش از فرزندان

Binary Tree Traversal

- پیاده سازی Preorder :

```
void BinaryTree::preorder()
{
    preorder(root);
}

Void BinaryTree::preorder(BinaryTreeNode* node)
{
    if (node)
    {
        cout << node->data;
        preorder(node->leftChild);
        preorder(node->rightChild);
    }
}
```