ต้นไม้แบบทวิภาค

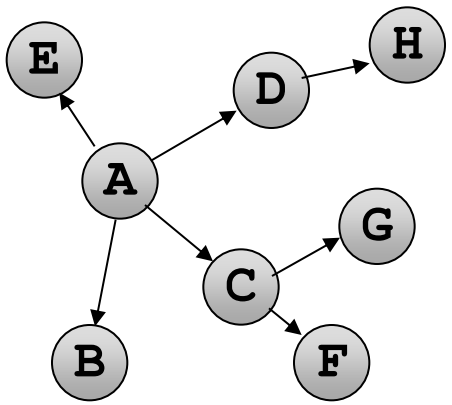
(Binary Trees)

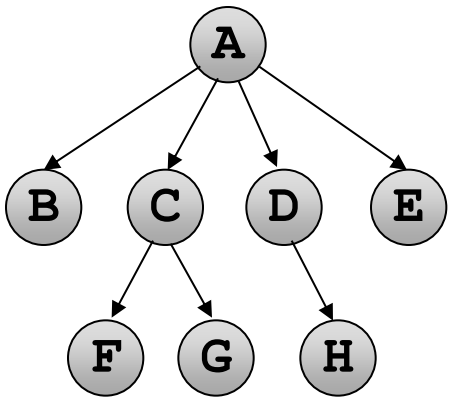
หัวข้อ

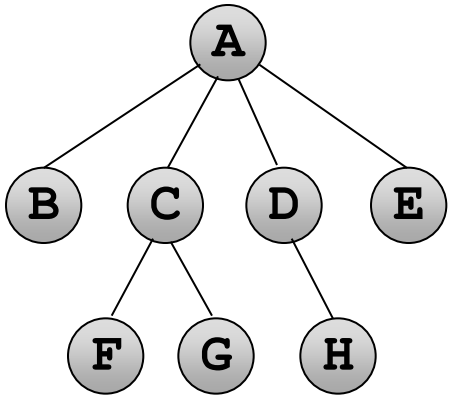
- นิยามต้นไม้
- การสร้างต้นไม้
- ต้นไม้แบบทวิภาค
  - ต้นไม้รหัสฮัฟฟ์แมน
  - ต้นไม้นิพจน์
  - การแวะผ่าน
  - การคำนวณค่าของต้นไม้นิพจน์
  - การหาอนุพันธ์ของฟังก์ชันตัวแปรเดียว

ต้นไม้ (Tree)

- ต้นไม้ประกอบด้วยปม (nodes) กับเส้นเชื่อม (edges)
- เส้นเชื่อมมีทิศทาง
- A เป็นปมพ่อของ B เมื่อมีเส้นเชื่อมจาก A ไปยัง B
- แต่ละปมมีปมพ่อได้เพียงปมเดียว (ยกเว้นปมพิเศษคือรากไม่มีพ่อ)
- ต้นไม้ที่มีปม v ปม ย่อมมี v – 1 เส้นเชื่อม

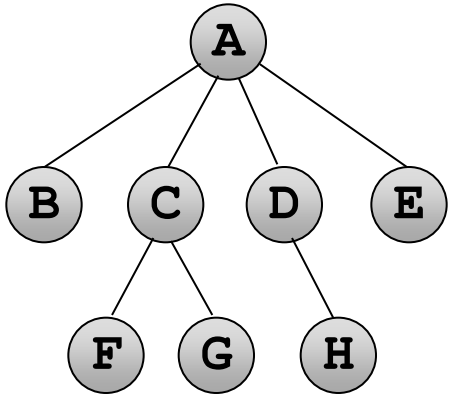


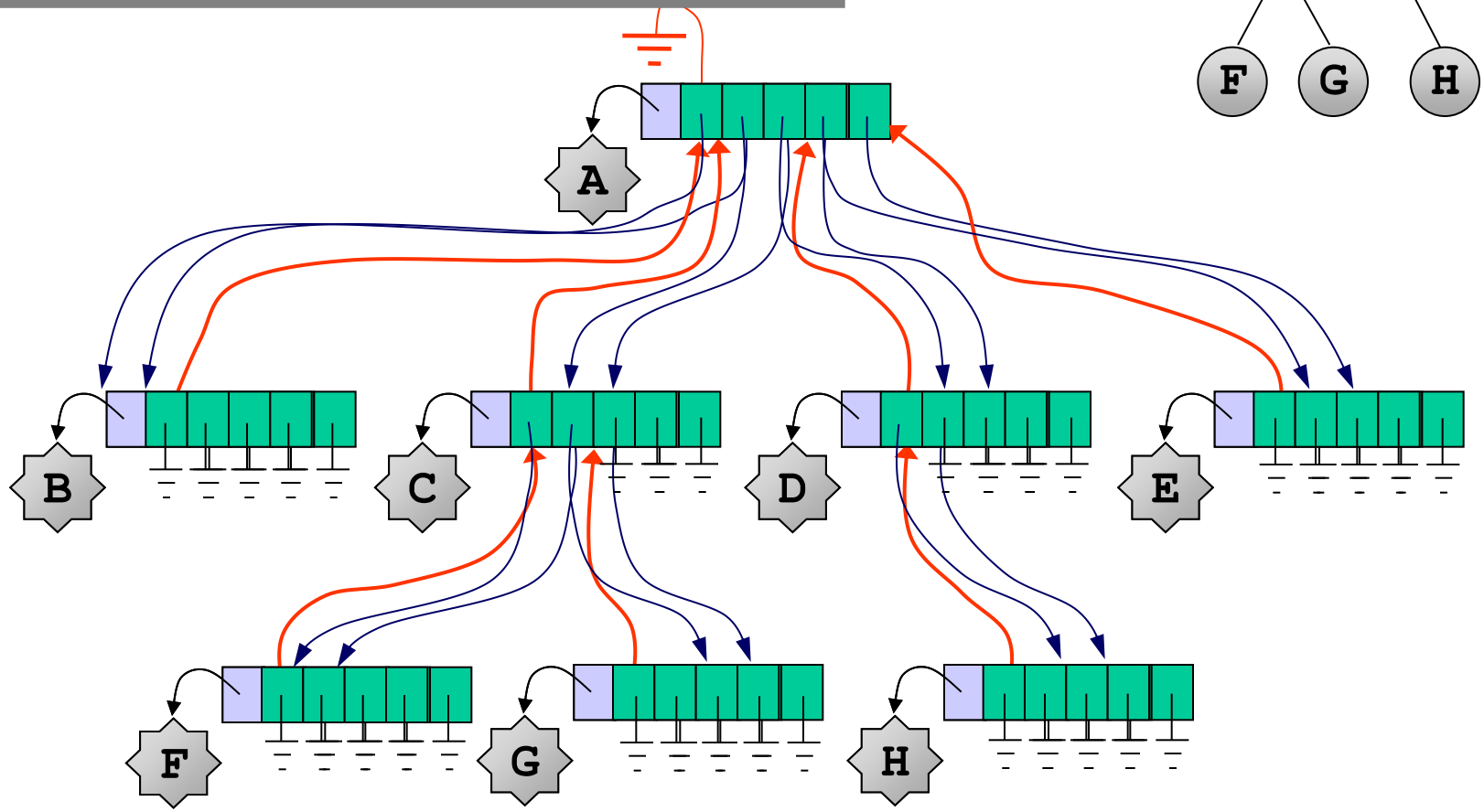




การสร้างต้นไม้ : ใช้อาเรย์เก็บลูก ๆ

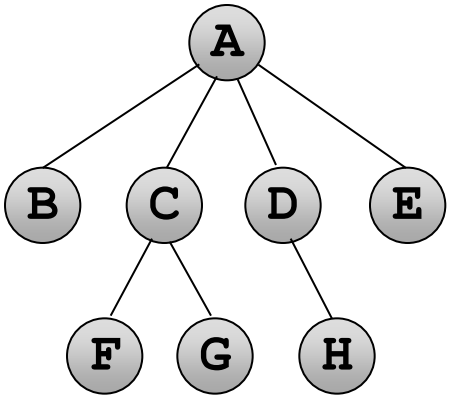
```
class node {
  T     data;
  node *children[4];
};
```

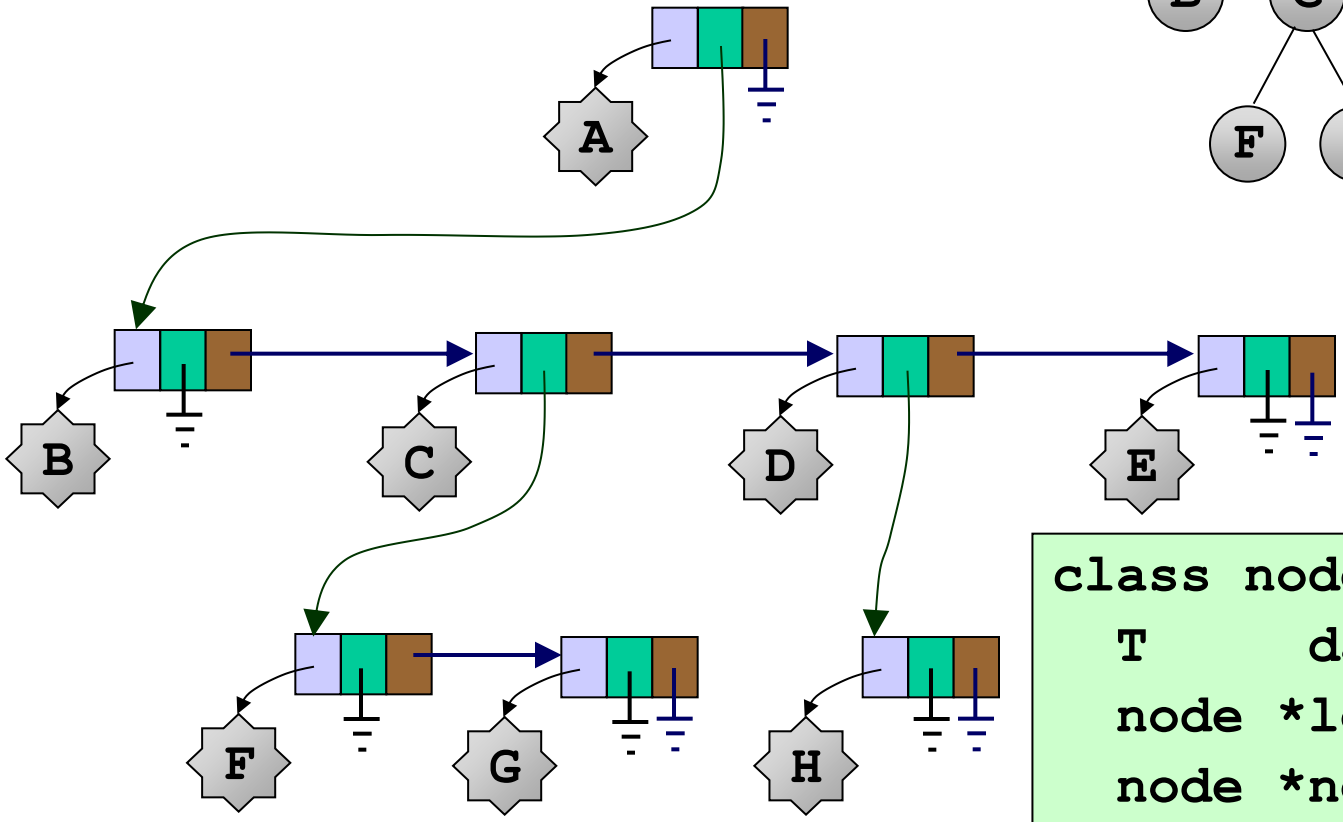




การสร้างต้นไม้ : ใช้รายการเก็บลูก ๆ

- มีตัวเชื่อมไปยังลูกคนโต
- มีตัวเชื่อมไปยังน้องคนถัดไป

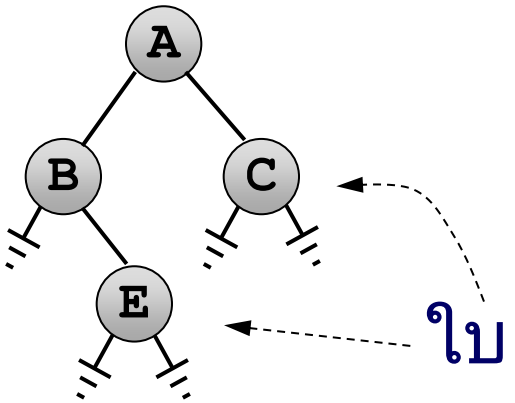


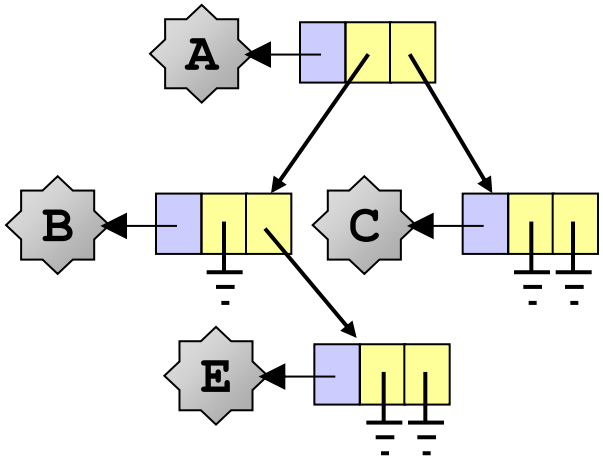



```
class node{
   T      data;
   node *leftChild;
   node *nextSibling;
};
```

ต้นไม้แบบทวิภาค (Binary Tree)

- ทุกปมมีสองลูก : ลูกซ้าย และลูกขวา






```
class node {
  T     data;
  node *left, *right;

  node(T data, node *left, node *right) :
    data(data), left(left), right(right)
  {}

  bool isLeaf() {
    return left==NULL && right==NULL;
};
```

รหัส Huffman

งึกงึกงักงักมันเป็นงึกงึกงักงัก
งึกงึกงักงักมันเป็นงึกงึกงักงัก
มันเป็นกะอึกกะอักมันเป็นจึกจึกจั๊กจั๊ก

ก	ง	◌ั	◌ึ	น	จ	ป	ม	เ	◌็	อ	ะ	◌ั๊
24	16	15	11	8	4	4	4	4	4	2	2	2
0000	0001	0010	1010	0011	0100	0101	0110	0111	1000	1001	1011	1100
11	010	000	001	0110	01110	01111	1000	1001	10100	10101	10110	10111

$4 \times (24 + 16 + 15 + 11 + 8 + 4 + 4 + 4 + 4 + 4 + 2 + 2 + 2) = 400$ บิต

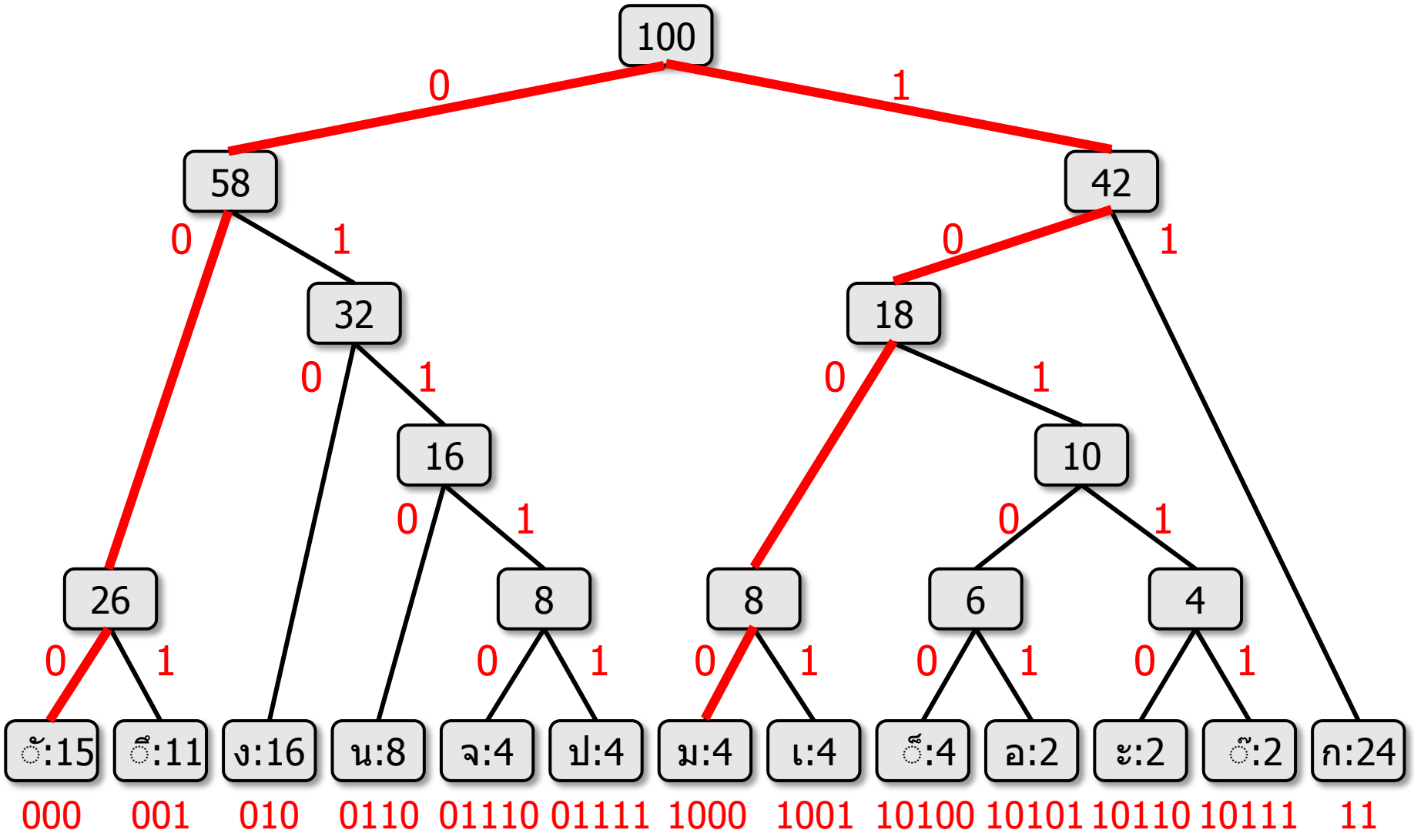
variable-length code

prefix-free code

$$\left.\begin{array}{l} 2 \times 24 + \\ 3 \times (16 + 15 + 11) + \\ 4 \times (8 + 4 + 4) + \\ 5 \times (4 + 4 + 4 + 2 + 2 + 2) \end{array}\right\} = 328 \text{ บิต}$$

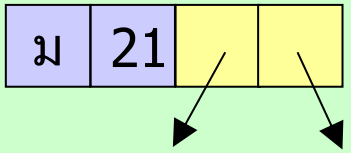
ไม่มีรหัสใดเป็นส่วนนำหน้าของรหัสอื่น

การหารหัส Huffman



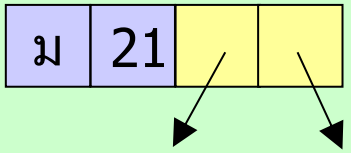
huffman_tree

```
class huffman_tree {
  class node {
    public:
      char c;
      int  freq;
      node *left, *right;

      node(char c, int freq, node *left, node *right) :
        c(c), freq(freq), left(left), right(right)
      {}
  };

  node *root;
```



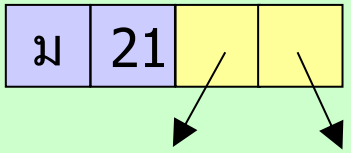
huffman_tree: ctor

```
class huffman_tree {
  class node { ... }
  node *root;
public:
  huffman_tree(char c[], int f[], size_t n) {
    class freq_comp {
      public:
        bool operator()(node *a, node *b) {
            return a->freq > b->freq;
        }
    };
    priority_queue<node*,vector<node*>, freq_comp > h;
    for (size_t i=0; i<n; ++i) {
       h.push(new node(c[i], f[i], NULL, NULL));
    }
    for (size_t i=0; i<n-1; ++i) {
       node *n1 = h.top(); h.pop();
       node *n2 = h.top(); h.pop();
       h.push(new node('*', n1->freq+n2->freq, n1, n2));
    }
    root = h.top();
  }
```



huffman_tree: dtor

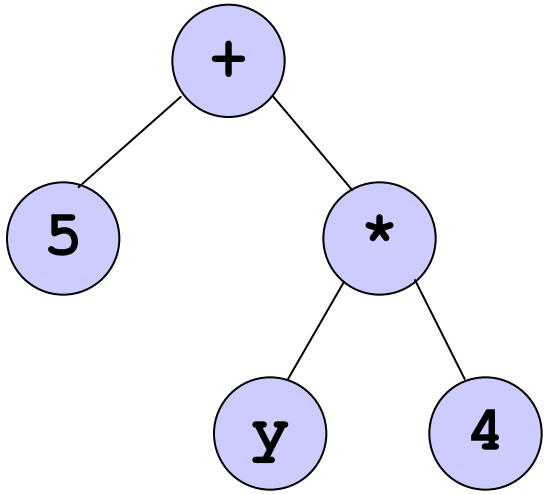
```
class huffman_tree {
  ...
  node *root;

public:
  huffman_tree(char c[], int f[], int n) { ... }

 ~huffman_tree() {
    delete_all_node( root );
  }
  void delete_all_nodes(hnode *r) {
    if (r == NULL) return;
    delete_all_nodes(r->left);
    delete_all_nodes(r->right);
    delete r;
  }
```

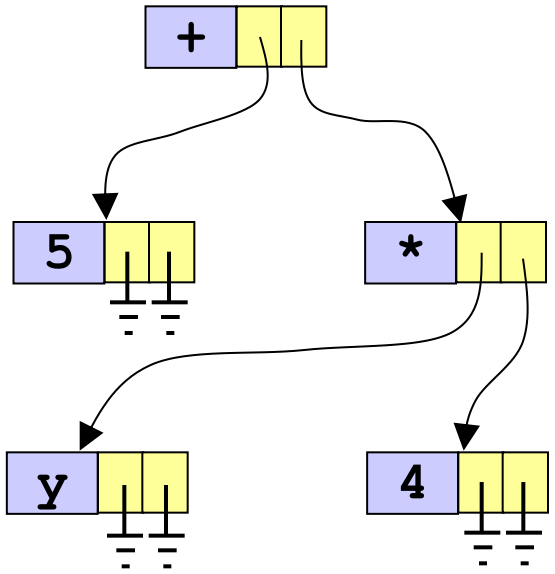


ต้นไม้นิพจน์ (Expression Tree)

- แทนนิพจน์ได้ด้วยต้นไม้
- ใบ : ตัวถูกดำเนินการ (operand)
- ปม : ตัวดำเนินการ (operator)

```
5 + y * 4
```

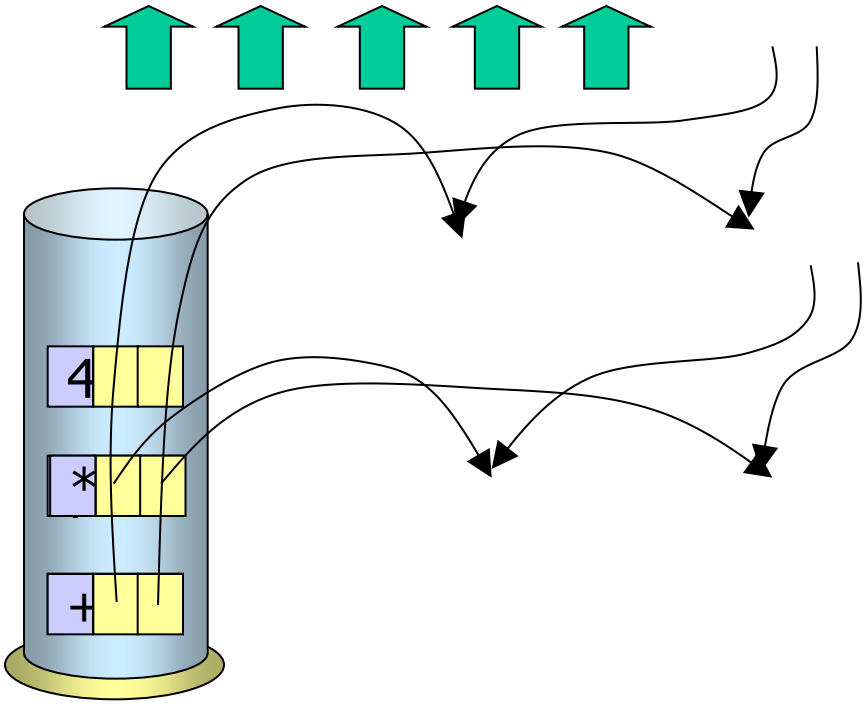




การสร้างต้นไม้นิพจน์

- พบ operand ให้ push ลงกองซ้อน
- พบ operator ให้ pop ออกมาเป็นลูกของปมใหม่ เพื่อ push ลงกองซ้อน

```
infix : 5  +  y  *  4
postfix : 5  y  4  *  +
```

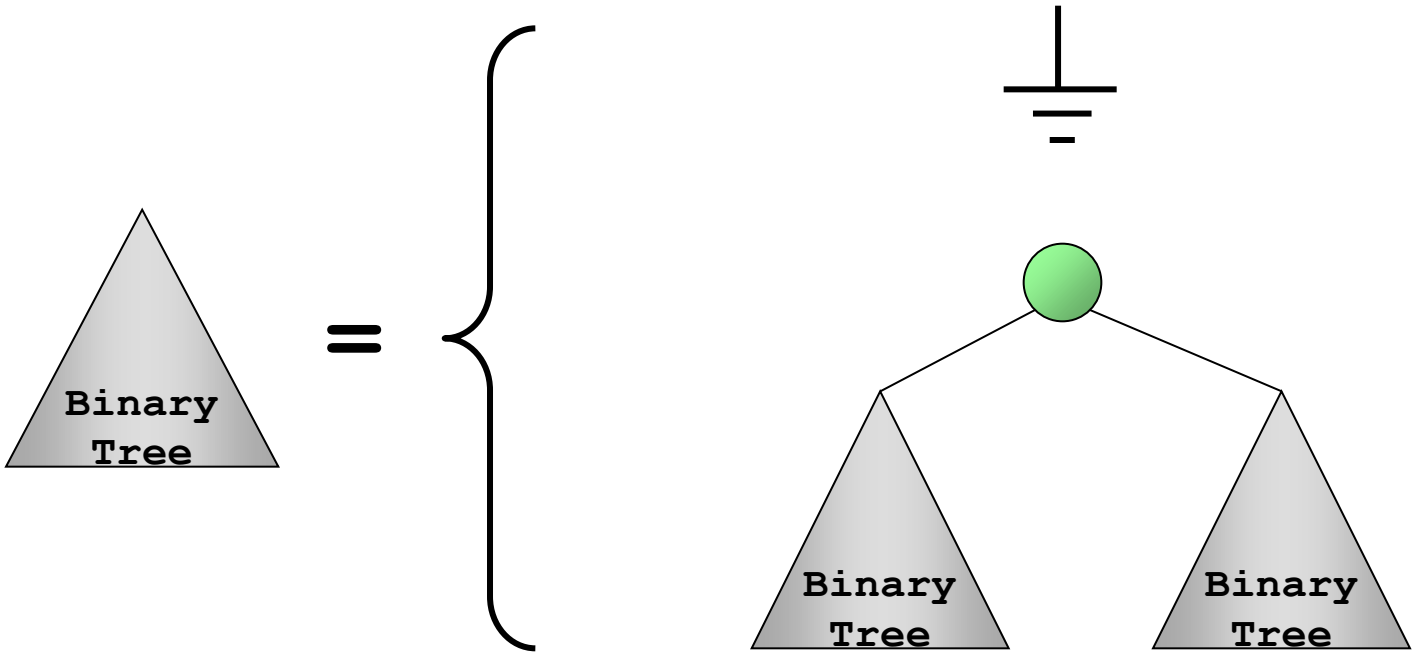


expression_tree

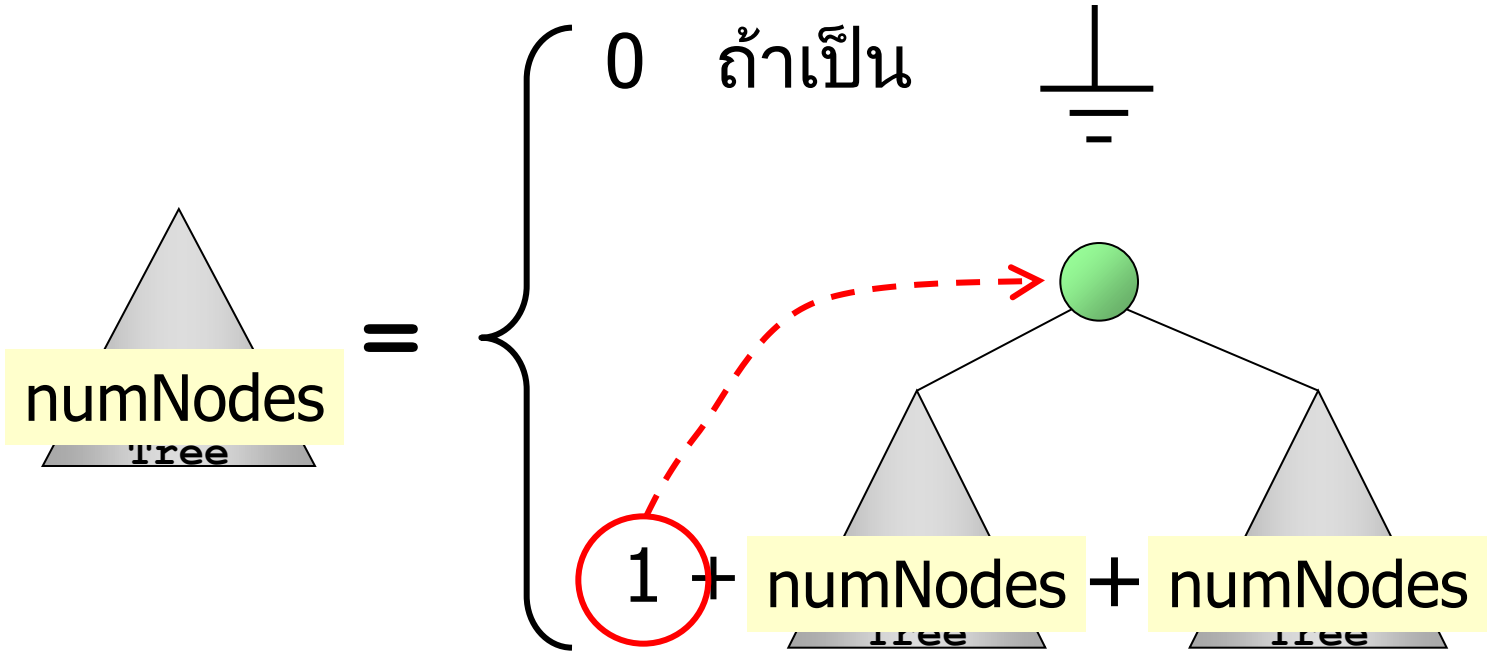
```
node *expression_tree(string &infix) {
    string postfix = infix2postfix(infix);
    stack<node*> s;
    for (size_t i=0; i<postfix.size(); i++) {
      char token = postfix[i];
      if (!isOperator(token)) {
        s.push(new node(token, NULL, NULL));
      } else {
        node *right = s.top(); s.pop();
        node *left  = s.top(); s.pop();
        s.push(new node(token, left, right));
      }
    }
    return s.top();
}
```

มองต้นไม้แบบทวิภาคแบบเวียนเกิด

- Binary tree คือ
  - ต้นไม้ว่าง (NULL) หรือ
  - หนึ่งปม และลูกต้นซ้ายกับลูกต้นขวา

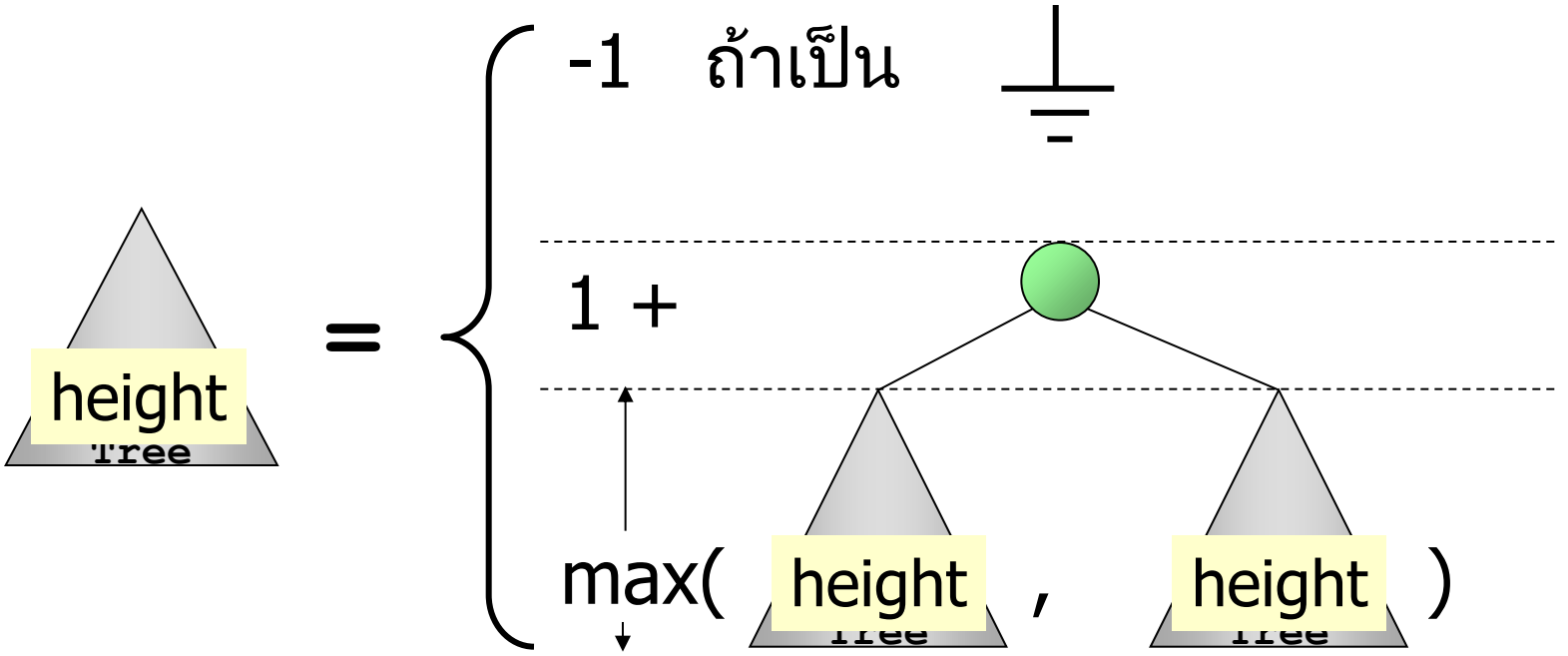


การหาจำนวนปมทั้งหมดของต้นไม้



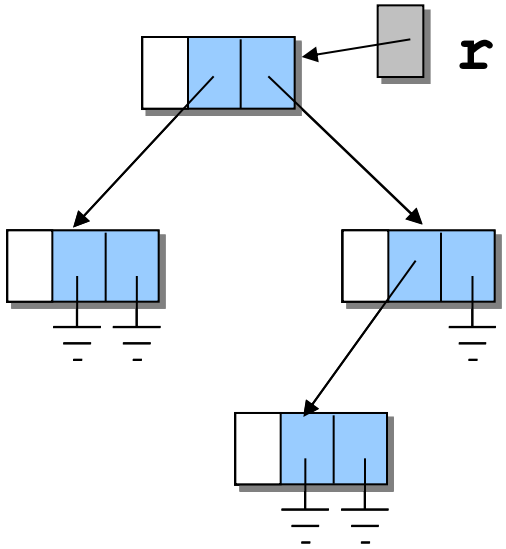

```
int numNodes(node *r) {
   if (r == NULL) return 0;
   return 1 + numNodes(r->left)
            + numNodes(r->right);
}
```

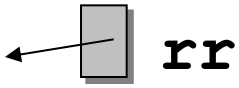
การหาความสูงของต้นไม้




```
int height(node *r) {
  if (r == NULL) return -1;
  return 1 + max(height(r->left),
                 height(r->right));
}
```

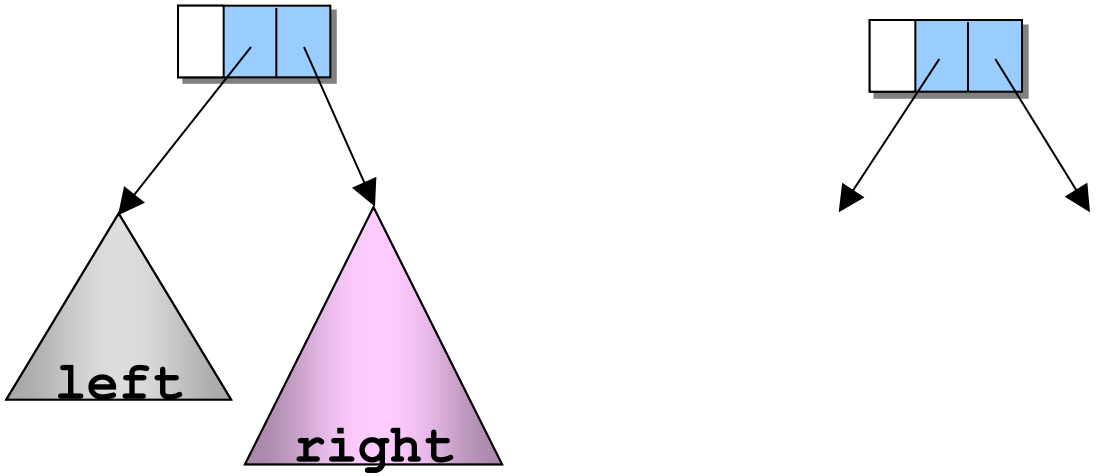
การสำเนาต้นไม้






```
node *rr = copy(r);
```

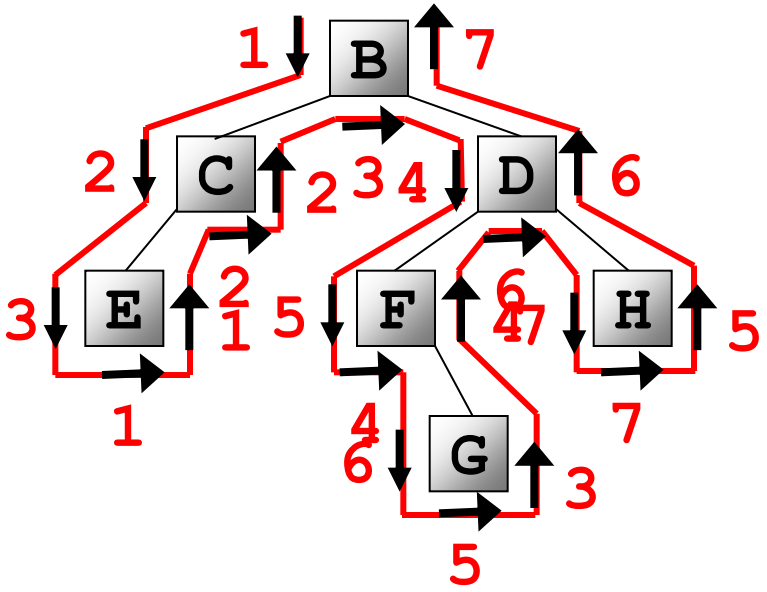
การสำเนาต้นไม้ (มองแบบเวียนเกิด)




```
node *copy(node *r) {
    if (r == NULL) return NULL;
    node *rL = copy(r->left);
    node *rR = copy(r->right);
    return new node(r->data, rL, rR);
}
```

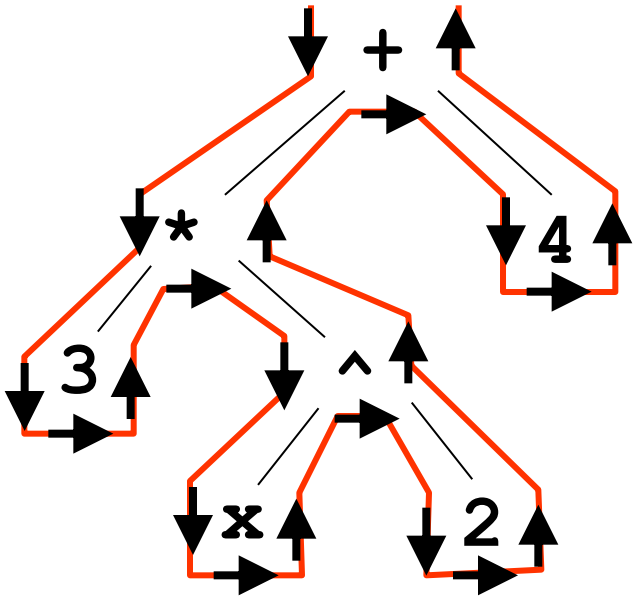
การแวะผ่านต้นไม้

- Tree traversal เป็นกระบวนการเข้าถึงปมต่าง ๆ ในต้นไม้ ปมละหนึ่งครั้งอย่างมีระเบียบ
  - แบบก่อนลำดับ (preorder) B, C, E, D, F, G, H
  - แบบตามลำดับ (inorder) E, C, B, F, G, D, H
  - แบบหลังลำดับ (postorder) E, C, G, F, H, D, B



การแวะผ่านต้นไม้นิพจน์

- แวะผ่านแบบก่อนลำดับ ได้นิพจน์แบบ prefix
- แวะผ่านแบบตามลำดับ ได้นิพจน์แบบ infix
- แวะผ่านแบบหลังลำดับ ได้นิพจน์แบบ postfix



+ * 3 ^ x 2 4

3 * x ^ 2 + 4

3 x 2 ^ * 4 +

การแวะผ่านต้นไม้ที่มี x เป็นราก

- แบบก่อนลำดับ
  - แวะ x
  - แวะผ่าน left
  - แวะผ่าน right
- แบบตามลำดับ
  - แวะผ่าน left
  - แวะ x
  - แวะผ่าน right
- แบบหลังลำดับ
  - แวะผ่าน left
  - แวะผ่าน right
  - แวะ x

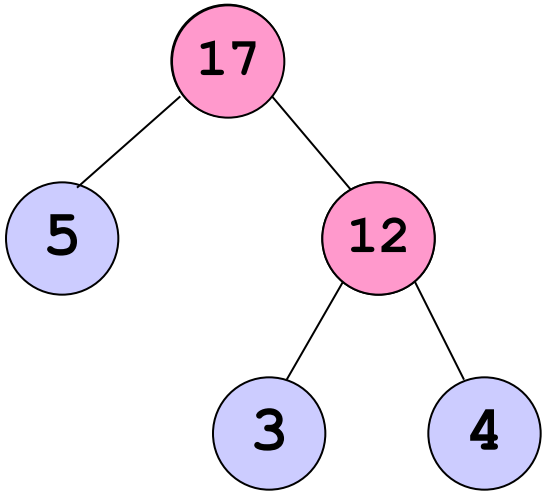
```
void preorder(node *x) {
  if (x == NULL) return;
  visit(x->data);
  preorder(x->left);
  preorder(x->right);
}
```

```
void inorder(node *x) {
  if (x == NULL) return;
  inorder(x->left);
  visit(x->data);
  inorder(x->right);
}
```

```
void postorder(node *x) {
  if (x == NULL) return;
  postorder(x->left);
  postorder(x->right);
  visit(x->data);
}
```

การคำนวณค่าของต้นไม้นิพจน์

- ต้องรู้ค่าของลูก ๆ ก่อน จึงคำนวณได้
- มีลักษณะคล้ายการแวะผ่านแบบหลังลำดับ



การคำนวณค่าของต้นไม้นิพจน์

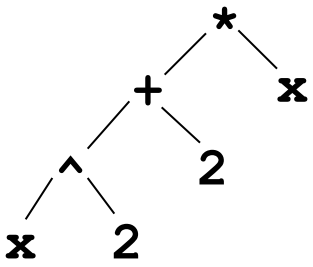
```
float eval(node *r) {
    if (r == NULL) return 0;
    if (r->isLeaf()) return (r->data - '0');
    float vLeft = eval(r->left);
    float vRight = eval(r->right);
    if (r->data == '+') return vLeft + vRight;
    if (r->data == '-') return vLeft - vRight;
    if (r->data == '*') return vLeft * vRight;
    if (r->data == '/') return vLeft / vRight;
    if (r->data == '^') return pow(vLeft, vRight);
    exit(-9); // ไม่น่ามาถึงตรงนี้
}
```

การหาอนุพันธ์ฟังก์ชันตัวแปรเดียว

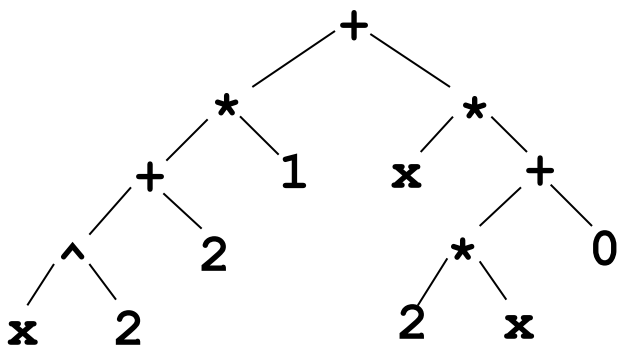
$$f(x) = (x^2 + 2)x$$

$$f'(x) = (x^2 + 2)\cdot 1 + x \cdot (2x + 0)$$

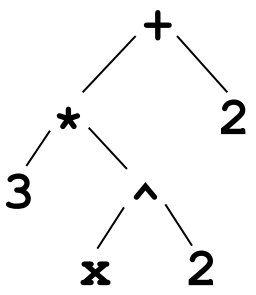
$$= 3x^2 + 2$$



f



diff(f)



simplify(f)

สูตรต่าง ๆ

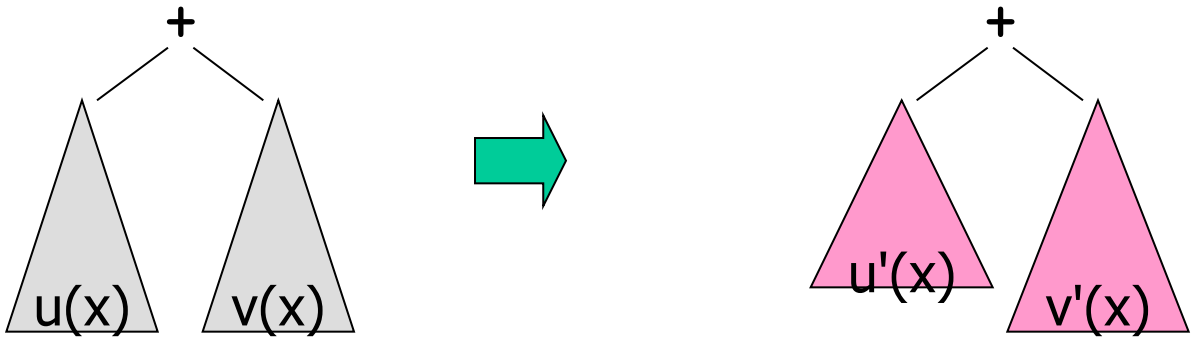
$$\left(u(x)+v(x)\right)' = u'(x)+v'(x)$$

$$\left(u(x)v(x)\right)' = v(x)u'(x)+u(x)v'(x)$$

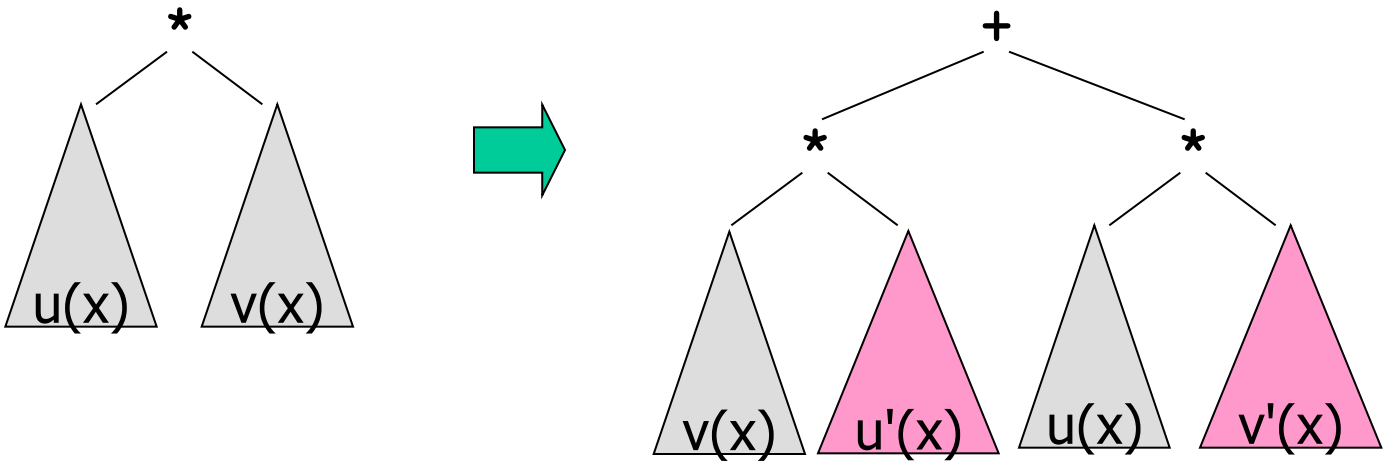
$$\left(\frac{u(x)}{v(x)}\right)' = \frac{v(x)u'(x)-u(x)v'(x)}{\left(v(x)\right)^2}$$

$$\left(\left(u(x)\right)^C\right)' = C\left(u(x)\right)^{C-1}u'(x)$$

อนุพันธ์ของต้นไม้นิพจน์ : +, *

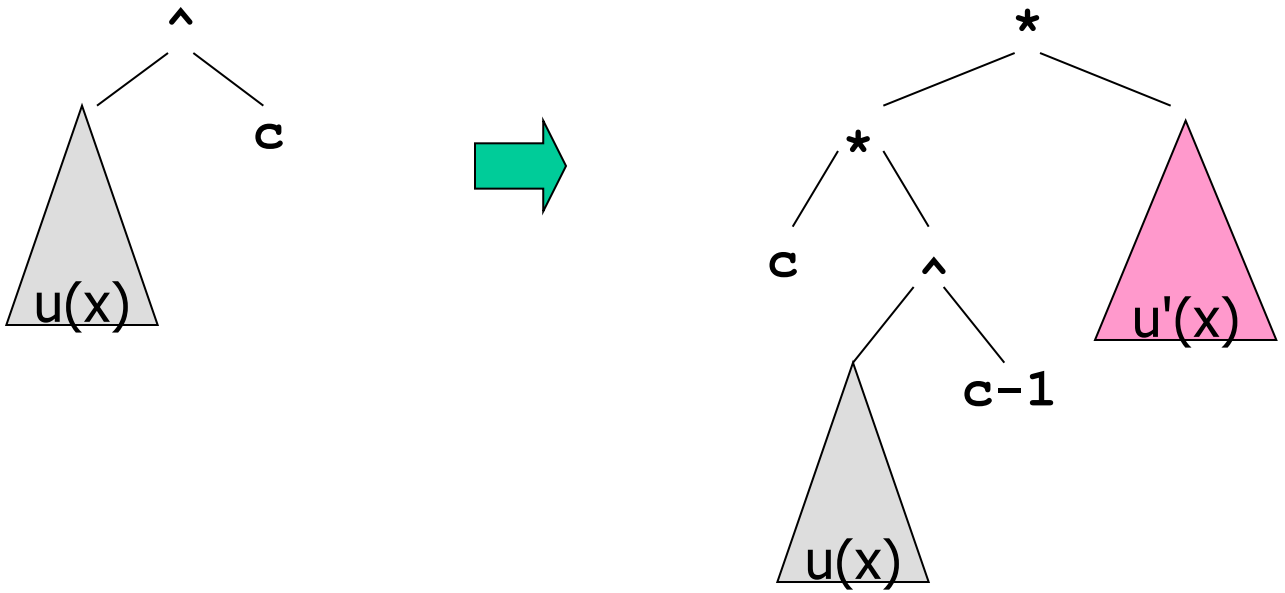


$$\left(u(x)+v(x)\right)' = u'(x)+v'(x)$$



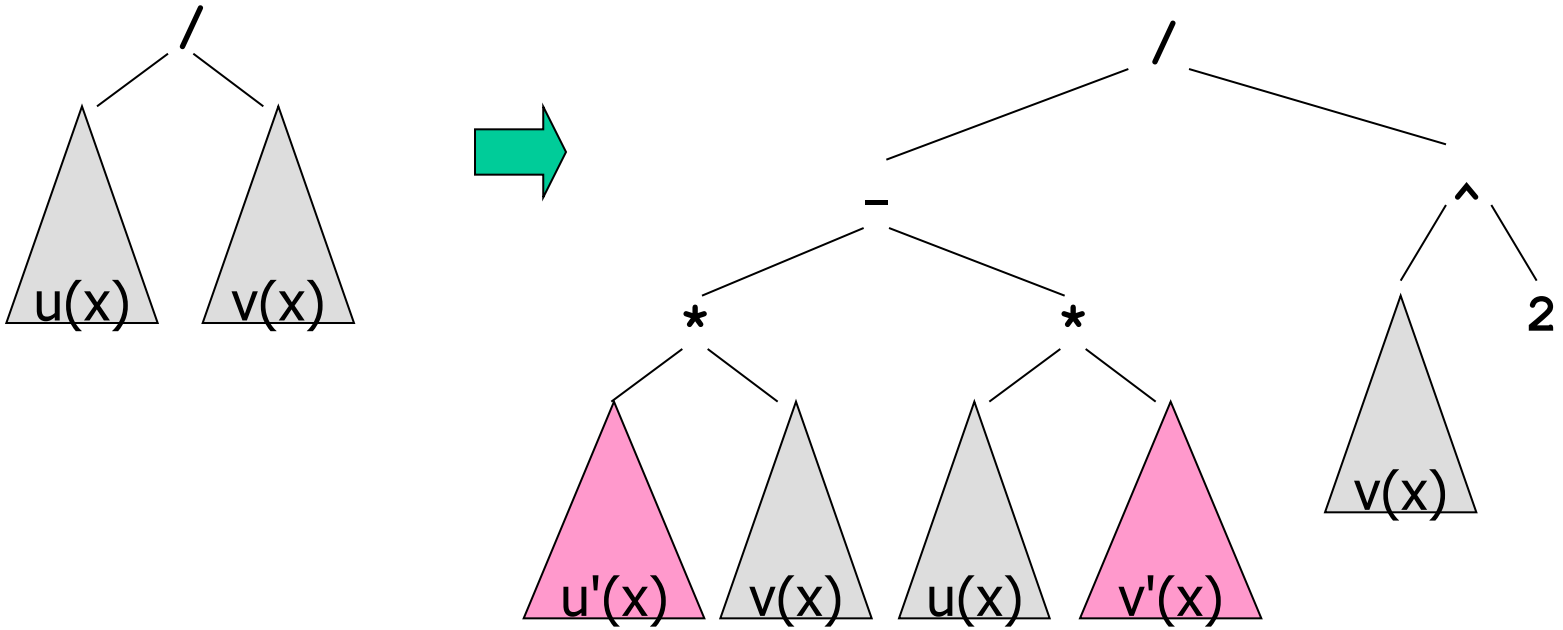
$$\left(u(x)v(x)\right)' = v(x)u'(x)+u(x)v'(x)$$

อนุพันธ์ของต้นไม้นิพจน์ : ^



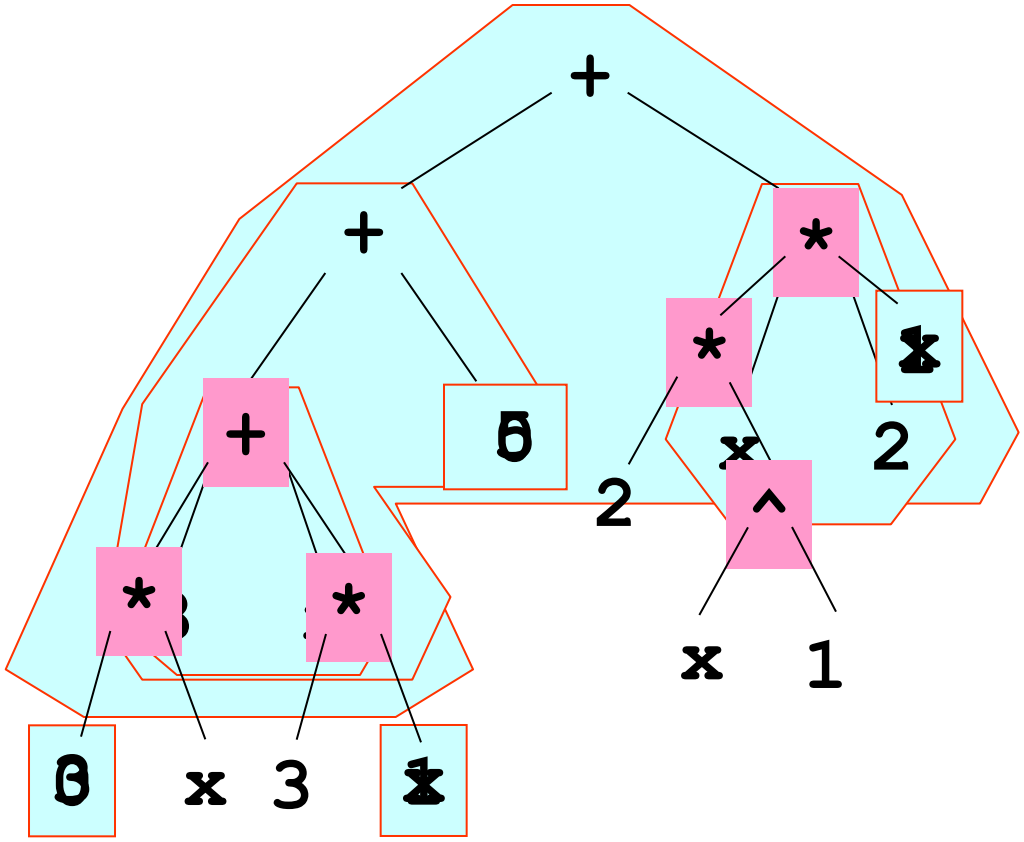
$$\left(\left(u(x) \right)^{C} \right)' = C \left(u(x) \right)^{C-1} u'(x)$$

อนุพันธ์ของต้นไม้นิพจน์ : /



$$\left(\frac{u(x)}{v(x)}\right)' = \frac{v(x)u'(x) - u(x)v'(x)}{\left(v(x)\right)^2}$$

ตัวอย่างการหาอนุพันธ์ของต้นไม้



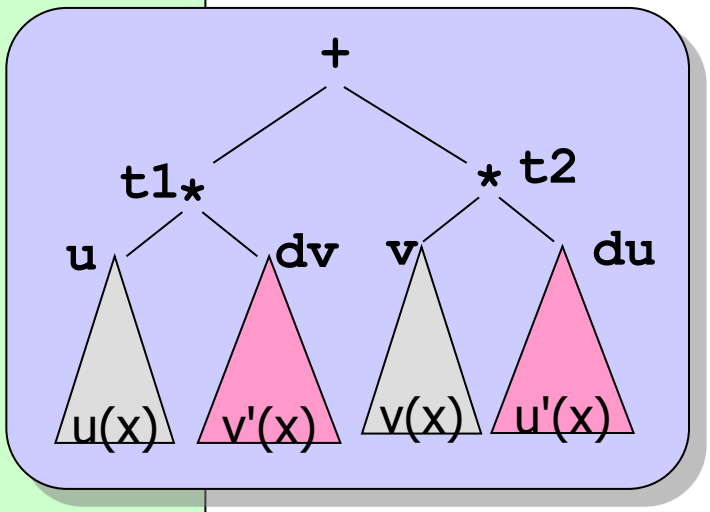
การหาอนุพันธ์

```
node *diff(node *r) {
    if (r == NULL) return NULL;
    char s = r->data;
    if ( r.isLeaf() ) {
      r->data = (s == 'x' ? '1' : '0');
    } else {
      if (s == '+')       r = diffSum(r);
      else if (s == '-') r = diffSum(r);
      else if (s == '^') r = diffPow(r);
      else if (s == '*') r = diffMult(r);
      else if (s == '/') r = diffDiv(r);
    }
    return r;
}
```

การหาอนุพันธ์ : +, *

```
node *diffSum(node *r) {
    r->left = diff(r->left);
    r->right = diff(r->right);
    return r;
}

node *diffMult(node *r) {
    node *u = copy(r->left);
    node *v = copy(r->right);
    node *du = diff(r->left);
    node *dv = diff(r->right);
    node *t1 = new node('*', u, dv);
    node *t2 = new node('*', v, du);
    return new node('+', t1, t2);
}
```

$$\big(u(x)+v(x)\big)' = u'(x)+v'(x)$$

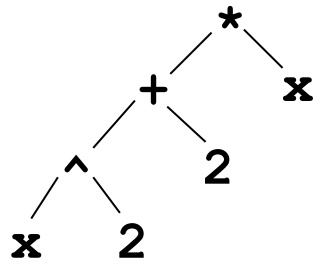


$$\big(u(x)v(x)\big)' = v(x)u'(x)+u(x)v'(x)$$

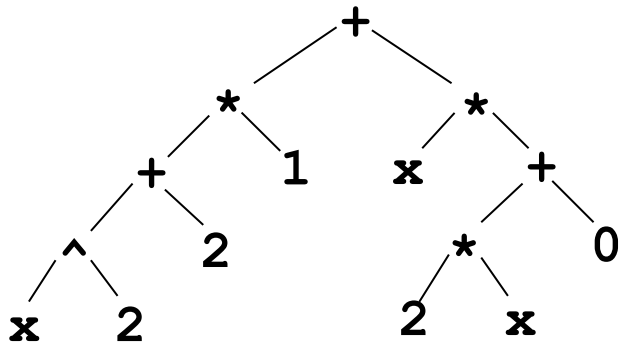
เขียนโปรแกรมทั้งหมดให้สมบูรณ์

```
char infix[100];
scanf("%s", infix);
node *r = newExpressionTree(infix);
printInorder(r); printPostorder(r);
r = diff(r);
r = simplify(r);
printInorder(r); printPostorder(r);
```

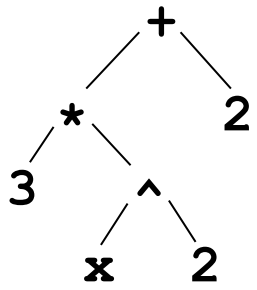
infix = "(x^2+2)*x"



r



diff(r)



simplify(r)

สรุป

- ต้นไม้เป็นโครงสร้างในการจัดเก็บข้อมูล
  - สร้างต้นไม้ได้ด้วยการโยงปมของต้นไม้
- ต้นไม้แบบทวิภาคเป็นต้นไม้ที่แต่ละปมมี 2 ลูก
  - มองต้นไม้ใหญ่ประกอบด้วยต้นไม้ย่อย
  - ทำให้เขียนฟังก์ชันได้แบบเวียนเกิด
  - การประมวลข้อมูลตามปม กระทำได้ด้วยการแวะผ่านแบบก่อนลำดับ ตามลำดับ หรือหลังลำดับ