

**CONFIDENTIAL**

# **C Programming Basic – week 8**

*Gdb – Make*

*Tree*

**Lecturers :**

**Cao Tuan Dung**

**Le Duc Trung**

**Dept of Software Engineering**

**Hanoi University of Technology**



# Chủ đề của tuần

- Công cụ debug với GDB
- Cấu trúc dữ liệu cây (Tree)
  - Binary Tree
  - Binary Search Tree
- Xử lý đệ quy trên Tree



# GDB

- Chi tiết xem slide tiếng Anh
- Mục đích:
  - Là chương trình debug (gỡ lỗi) chương trình trong Linux
  - Debug: tiến hành kiểm tra, theo dõi sự thực thi của chương trình => tìm ra lỗi



# Cách sử dụng GDB

- Trước khi sử dụng **`gdb`**:
  - dịch mã nguồn C với cờ `-g`
- Thực thi:
  - `gdb tên_file biên_dịch`



# Cách sử dụng GDB

- Tạo điểm dừng (break point): khi đến điểm này, chương trình tạm dừng lại
  - `gdb break` số\_dòng hoặc
  - `gdb break` tên\_hàm
- Để xóa điểm dừng
  - `gdb break` số\_thứ\_tự\_break\_point



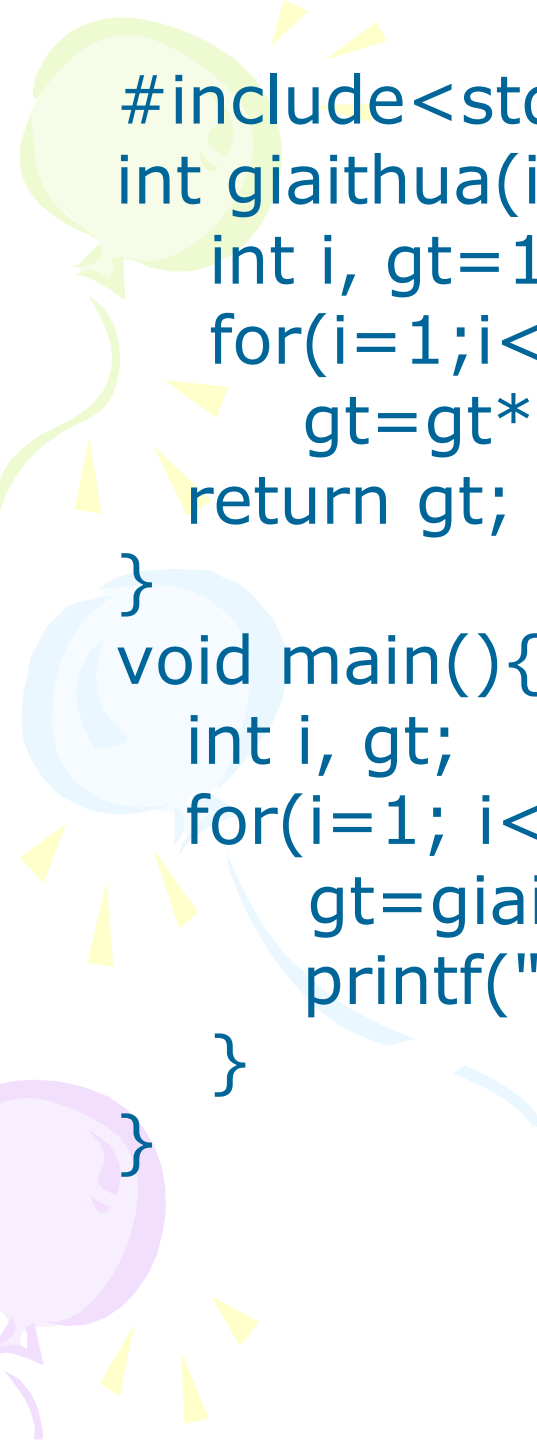
# Cách sử dụng GDB

- Tại điểm dừng
  - **gdb next** [số\_dòng]
  - nếu không có số\_dòng thì lệnh kế tiếp được thực thi
  - ngược lại, chương trình chạy từ lệnh hiện tại tới dòng lệnh [số\_dòng]
  - chạy tiếp đến điểm dừng tiếp theo hoặc tới hết chương trình: **gdb continue**
  - chạy vào trong thân hàm **gdb step**



# Cách sử dụng GDB

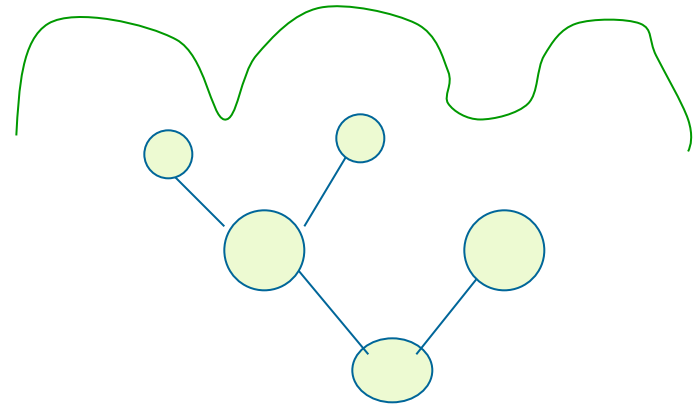
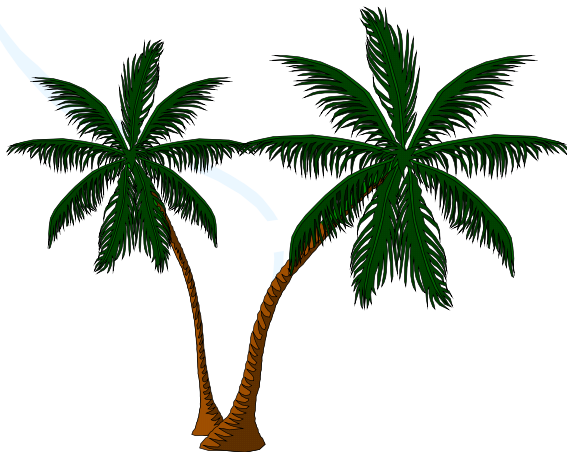
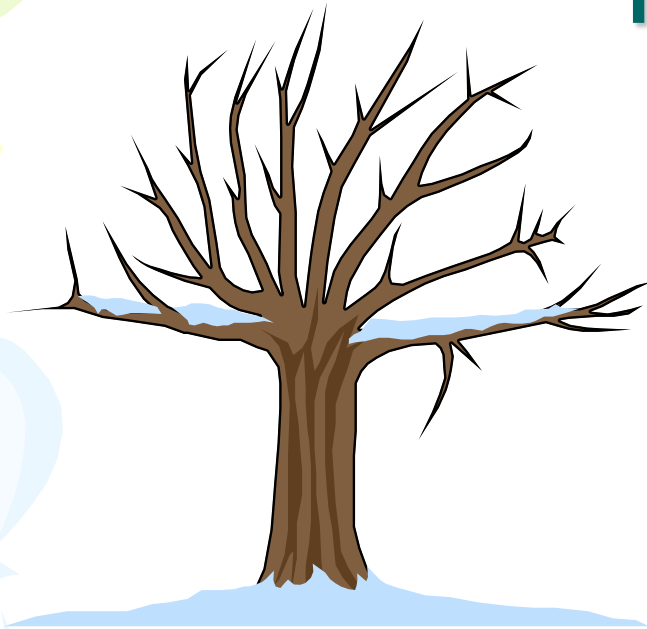
- Chạy chương trình bằng lệnh: **gdb run**
  - chương trình không có lỗi => thực thi bình thường
  - ngược lại => thông báo lỗi. Sử dụng **gdb where** để xác định vị trí lỗi
  - Thoát: **gdb quit**
- Để xem giá trị của một biến
  - **gdb display** tên\_biến (in giá trị biến mỗi lần thực hiện lệnh) hoặc
  - **gdb print** tên\_biến



```
#include<stdio.h>
int giaithua(int n){
    int i, gt=1;
    for(i=1;i<=n;i++)
        gt=gt*i;
    return gt;
}
void main(){
    int i, gt;
    for(i=1; i<=5; i++){
        gt=giaithua(i);
        printf("%3d\n",gt);
    }
}
```



# Tree

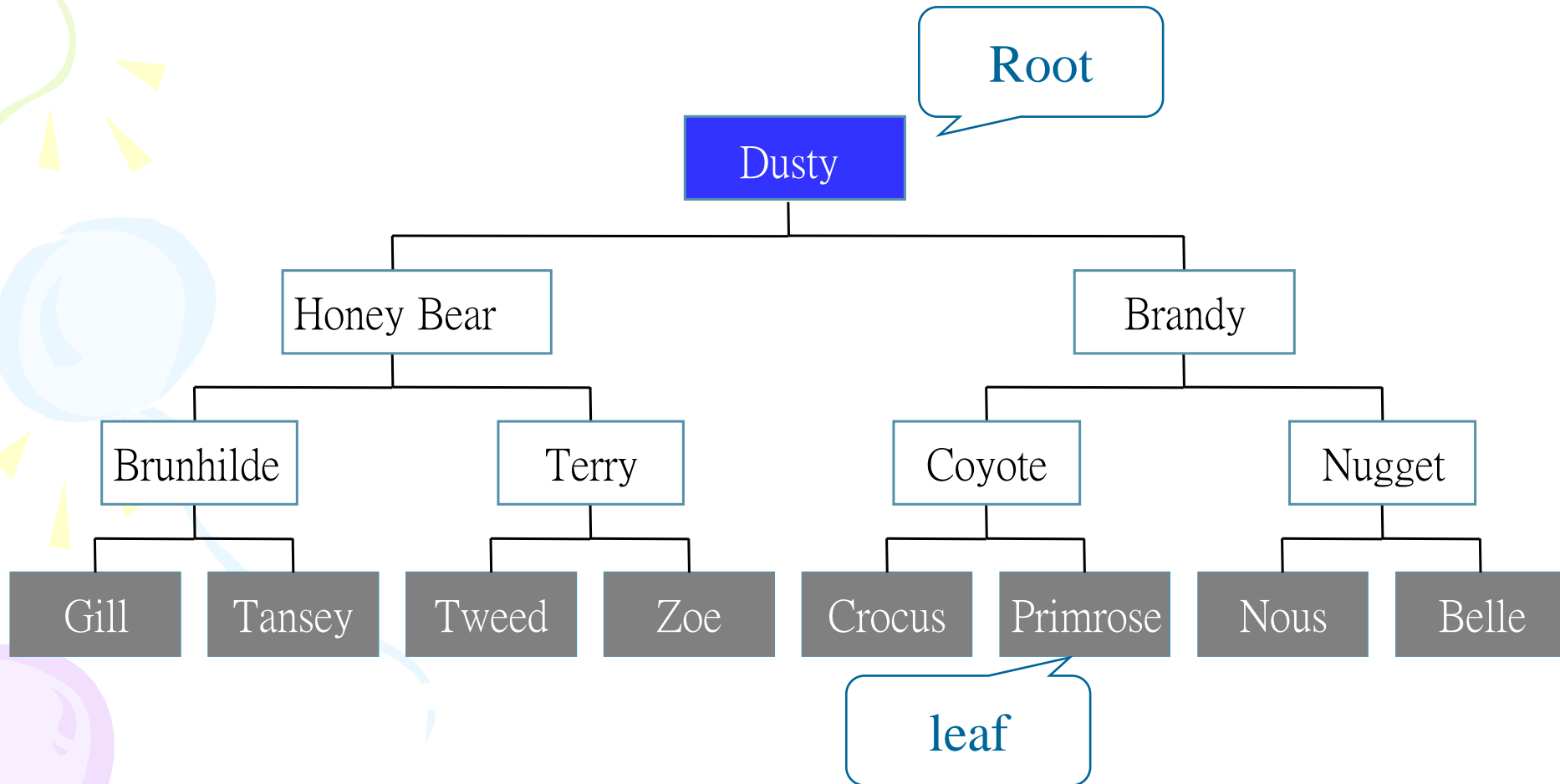




# Trees, Binary Trees, and Binary Search Trees

- Linked list: cấu trúc tuyến tính, khó thể hiện được sự thứ bậc (hierarchi)
- Stack, Queue: thể hiện được một phần thứ bậc nhưng chỉ 1 chiều
- Tree: khắc phục những hạn chế trên.
  - bao gồm các nút và cạnh.
  - Ngược với cây tự nhiên: gốc ở trên và các lá ở dưới

# Family Tree

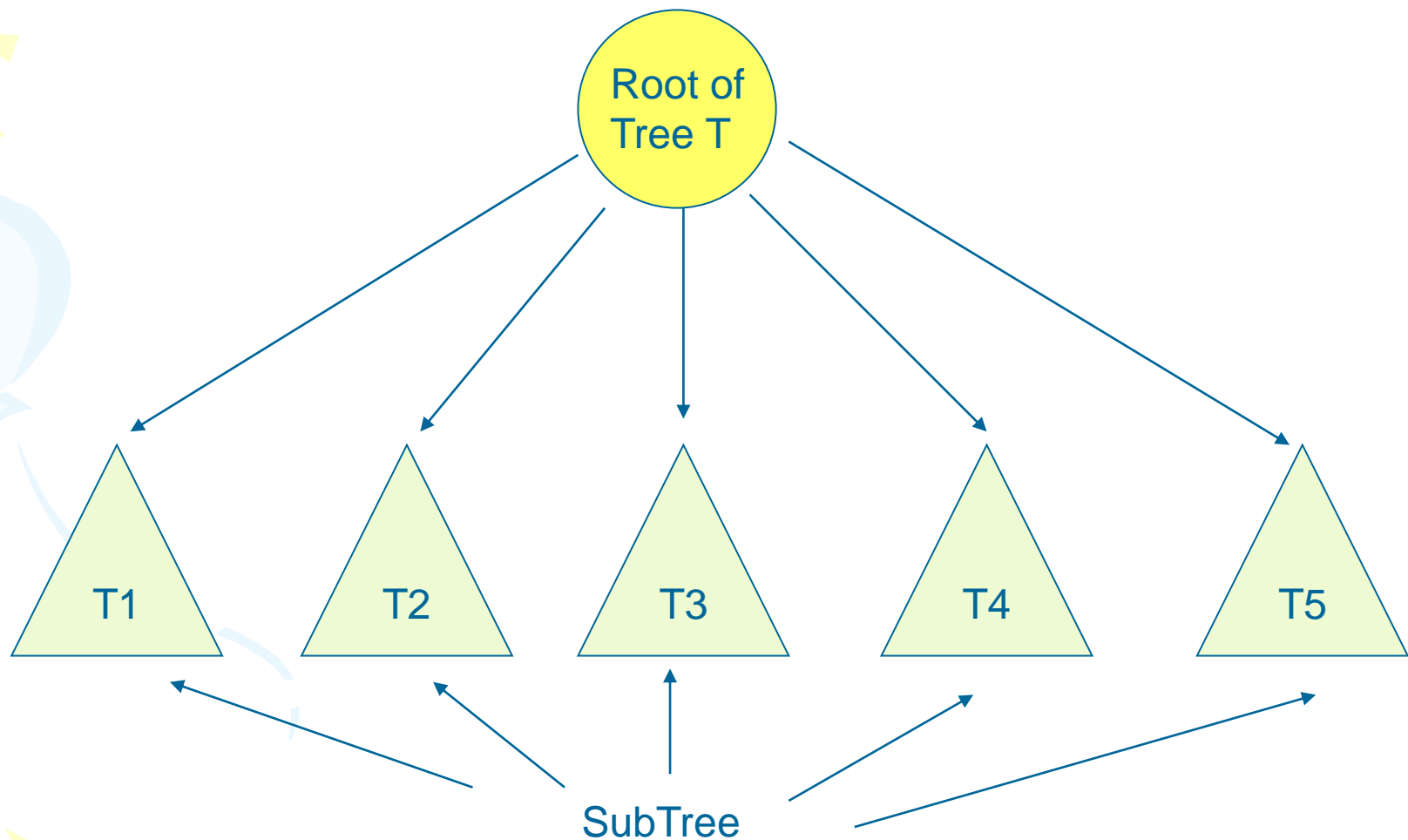




# Định nghĩa cây

- Cây là tập hợp hữu hạn của một hoặc nhiều nút trong đó:
  - Có 1 nút đặc biệt gọi là nút gốc: root
  - Các nút còn lại được phân chi thành các cây con không giao nhau  $T_1, T_2, \dots, T_n$

# Định nghĩa đệ quy





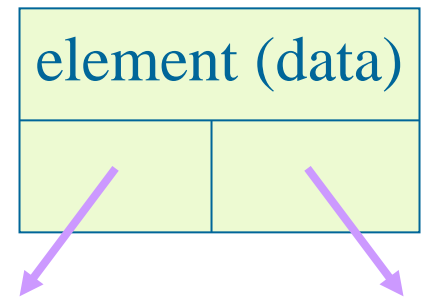
# Binary Tree

- Binary tree (cây nhị phân): là một cây trong đó mỗi nút không có quá 2 nút con  
=> Mỗi nút chỉ có 0, 1, hoặc 2 nút con

# Biểu diễn liên kết

- Mỗi nút gồm có: dữ liệu, liên kết đến các con của nó (tối đa 2 con) => gồm các trường: dữ liệu, con trái và con phải

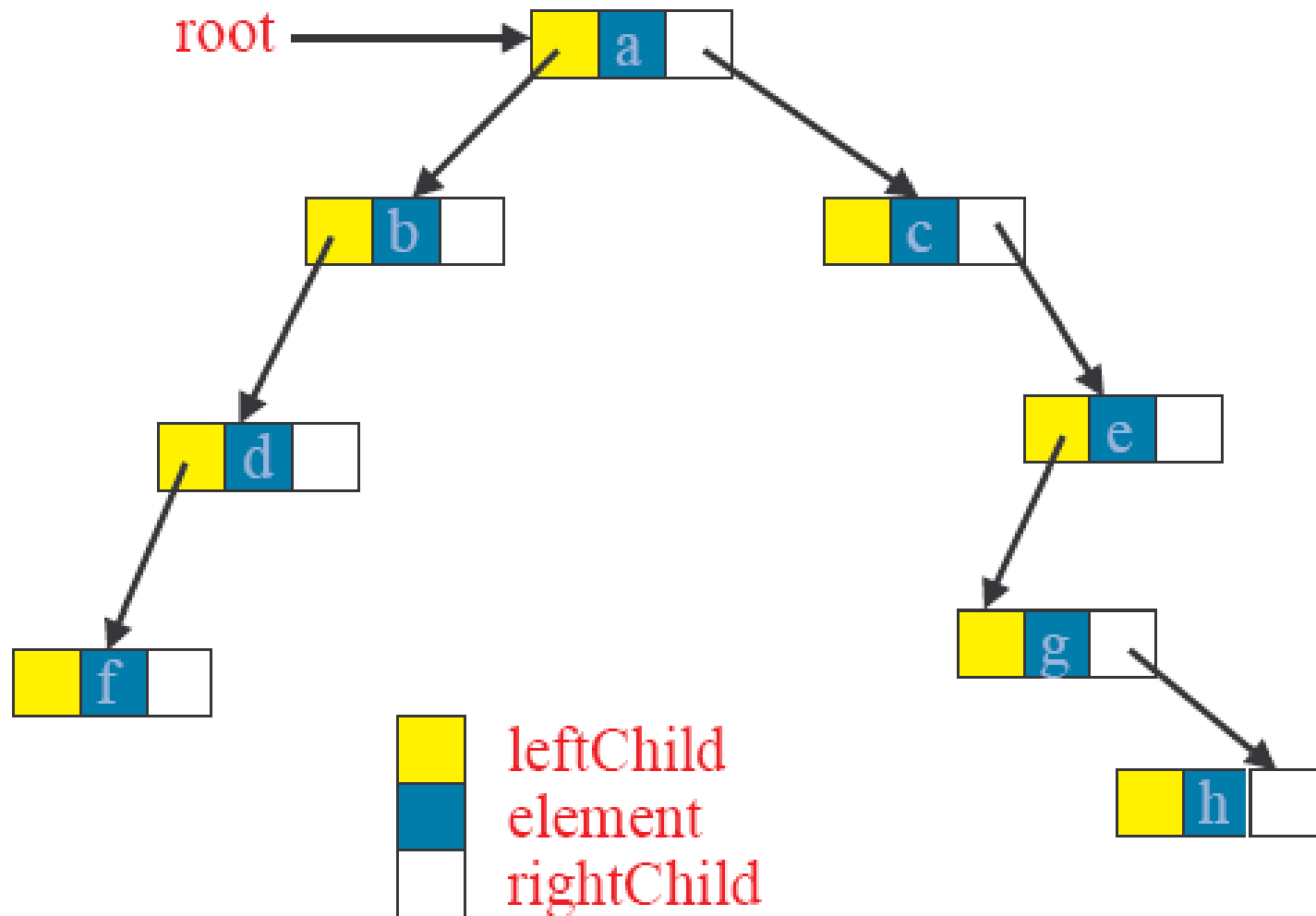
```
typedef ... elmType;  
//Cấu trúc 1 nút  
typedef struct nodeType {  
    elmType element;  
    struct nodeType *left, *right;  
};
```



left child   right child

```
typedef struct nodeType *treetype;
```

# Ví dụ về cây nhị phân





Three balloons (green, blue, and purple) with yellow streamers are positioned on the left side of the slide.

# Một số hàm

- `makenullTree(treetype *t)`
- `creatnewNode()`
- `isEmpty()`



# Khởi tạo và kiểm tra cây

```
typedef ... elmType;
typedef struct nodeType {
    elmType element;
    struct nodeType *left, *right;
} node_Type;

typedef struct node_Type* treetype;

void MakeNullTree(treetype *T) {
    (*T)=NULL;
}

int EmptyTree(treetype T) {
    return T==NULL;
}
```



# Truy cập con trái, phải

```
treetype LeftChild(treetype n)
```

```
{
```

```
    if (n!=NULL) return n->left;
```

```
    else return NULL;
```

```
}
```

```
treetype RightChild(treetype n)
```

```
{
```

```
    if (n!=NULL) return n->right;
```

```
    else return NULL;
```

```
}
```



# Tạo nút mới

```
node_type *create_node(elmtype NewData)
{
    node_type *N;
    N=(node_type*)malloc(sizeof(node_type)) ;
    if (N != NULL) {
        N->left = NULL;
        N->right = NULL;
        N->element = NewData;
    }
    return N;
}
```



# Kiểm tra nút lá?

```
int IsLeaf(treetype n) {  
    if (n!=NULL)  
        return (LeftChild(n)==NULL) && (Right  
            Child(n)==NULL) ;  
    else return -1;  
}
```



# Xử lý đệ quy: tìm số nút trên cây

- Vì cây là một cấu trúc dữ liệu đệ quy (cây gồm các cây con) => có thể áp dụng giải thuật đệ quy
- Số nút = 1 (nút gốc) + Số nút cây con trái + Số nút cây con phải

```
int nb_nodes(treetype T) {  
    if (EmptyTree(T)) return 0;  
    else return 1+nb_nodes(LeftChild(T)) +  
        nb_nodes(RightChild(T));  
}
```



# Tạo 1 cây từ 2 cây con

```
treetype createfrom2(elmttype v,  
    treetype l, treetype r){  
    treetype N;  
    N=(node_type*)malloc(sizeof(node_type)  
        e) );  
    N->element=v;  
    N->left=l;  
    N->right=r;  
    return N;  
}
```

# Thêm một nút vào vị trí trái nhất trên cây

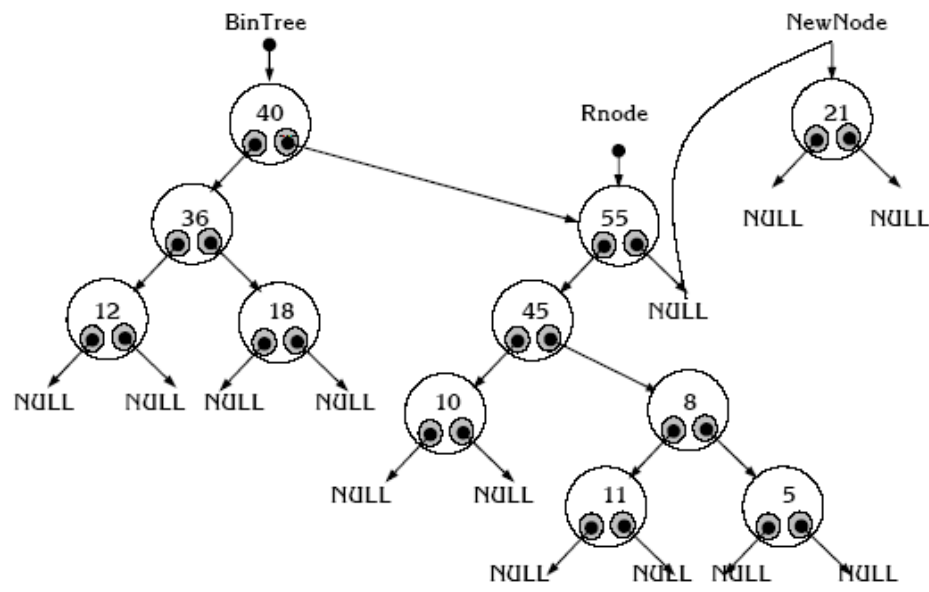
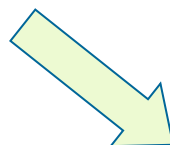
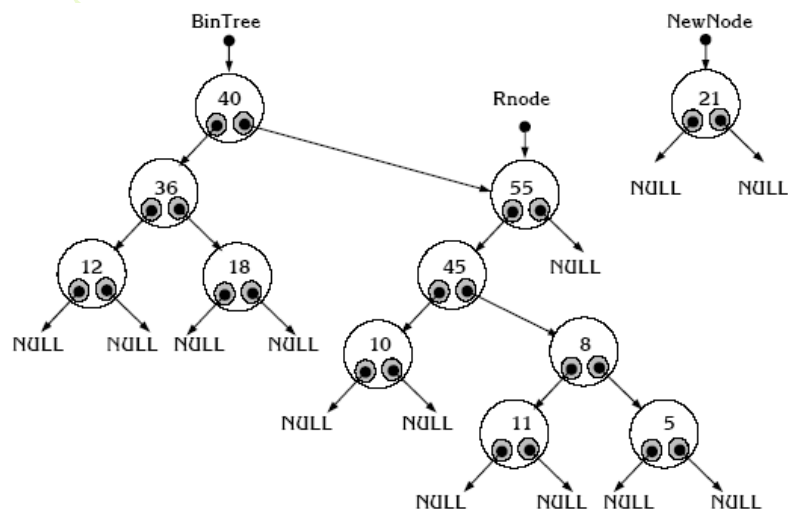
```
treetype Add_Left(treetype *Tree, elmtyp NewData){
    node_type *NewNode = Create_Node(NewData);
    if (NewNode == NULL) return (NewNode);
    if (*Tree == NULL)
        *Tree = NewNode;
    else{
        node_type *Lnode = *Tree;
        while (Lnode->left != NULL)
            Lnode = Lnode->left;
        Lnode->left = NewNode;
    }
    return (NewNode);
}
```



# Thêm một nút vào vị trí phải nhất trên cây

```
treetype Add_Right(treetype *Tree, elmtype NewData){
    node_type *NewNode = Create_Node(NewData);
    if (NewNode == NULL) return (NewNode);
    if (*Tree == NULL)
        *Tree = NewNode;
    else{
        node_type *Rnode = *Tree;
        while (Rnode->right != NULL)
            Rnode = Rnode->right;
        Rnode->right = NewNode;
    }
    return (NewNode);
}
```

# Ví dụ: phải nhất



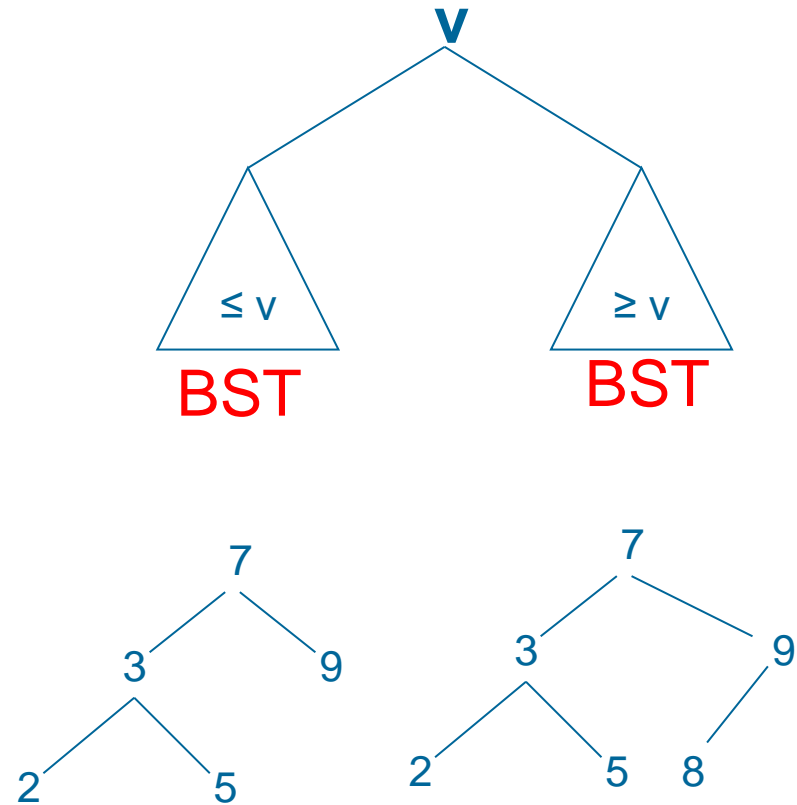


# Bài tập

- Viết chương trình thực hiện
  - Cài đặt một cấu trúc cây với kiểu dữ liệu `elemType` là `int`
  - Nhập từ bàn phím số nguyên  $n$  và  $m$ . Sau đó nhập lần lượt  $n$  nút trái nhất và  $m$  nút phải nhất trên cây
  - Cho biết số lượng nút
  - Cho biết số lượng nút lá
  - Cho biết độ cao của cây

# Binary Search Tree

- Mỗi nút có một khóa (key) duy nhất
- Mọi key ở nút con trái (phải) thì nhỏ hơn (lớn hơn) key ở nút gốc.
- Các cây con trái, phải cũng là các cây nhị phân tìm kiếm



# Cài đặt Binary Search Tree

```
#include <stdio.h>
#include <stdlib.h>
typedef . . . KeyType; // Loại dữ liệu của Key
typedef struct Node{
    KeyType key;
    struct Node* left,right;
} NodeType;
typedef Node* TreeType;
```



# Tìm kiếm trên BST

```
TreeType Search(KeyType x, TreeType Root) {  
    if (Root == NULL) return NULL; // not found  
    else if (Root->key == x) /* found x */  
        return Root;  
    else if (Root->key < x)  
        //continue searching in the right sub tree  
        return Search(x, Root->right);  
    else {  
        // continue searching in the left sub tree  
        return Search(x, Root->left);  
    }  
}
```

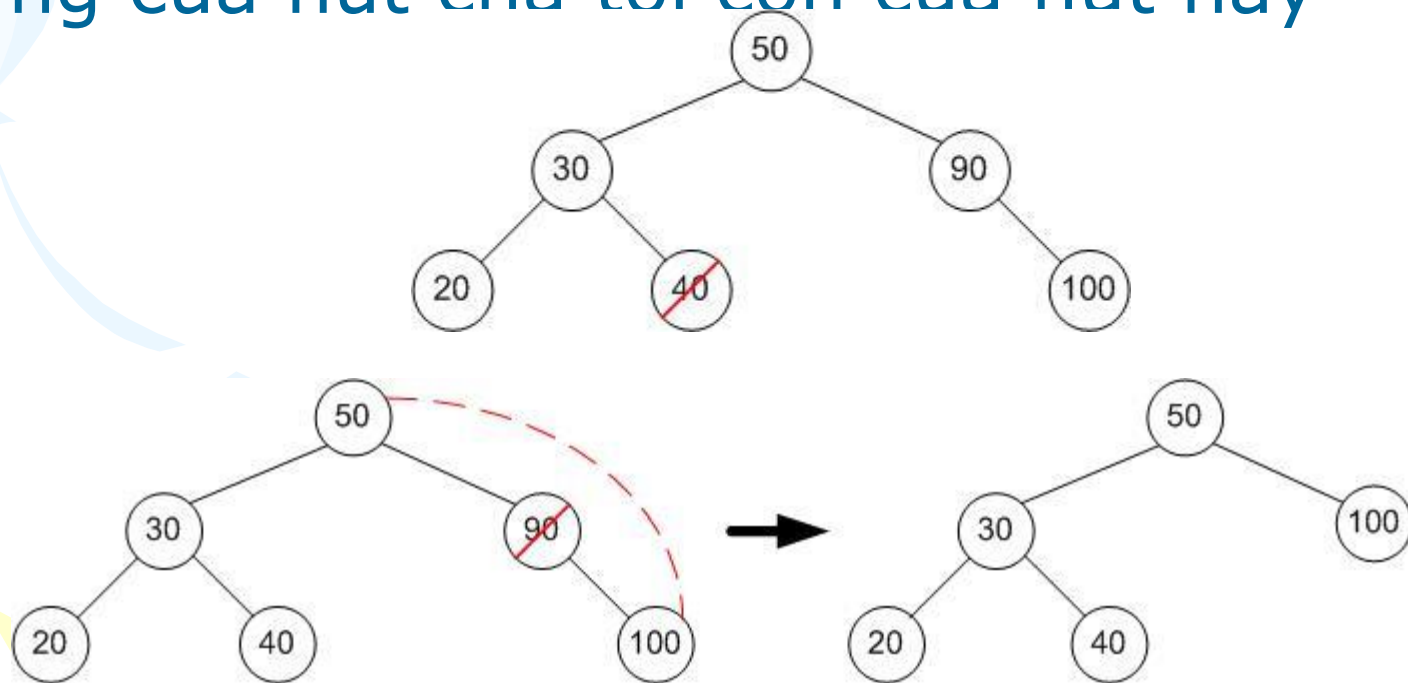
# Chèn một nút vào BST

- Lưu ý: Trong BST, không có 2 nút nào có cùng key

```
void InsertNode(KeyType x, Node **Root ) {
    if (*Root == NULL) {
        /* Create a new node for key x */
        *Root = (Node*) malloc(sizeof(Node));
        (*Root) -> key = x;
        (*Root) -> left = NULL;
        (*Root) -> right = NULL;
    }
    else if (x < (*Root) -> key) InsertNode(x, &(*Root) -
        > left);
    else if (x > (*Root) -> key) InsertNode(x, &(*Root) -
        > right);
}
```

# Xóa một nút khỏi BST

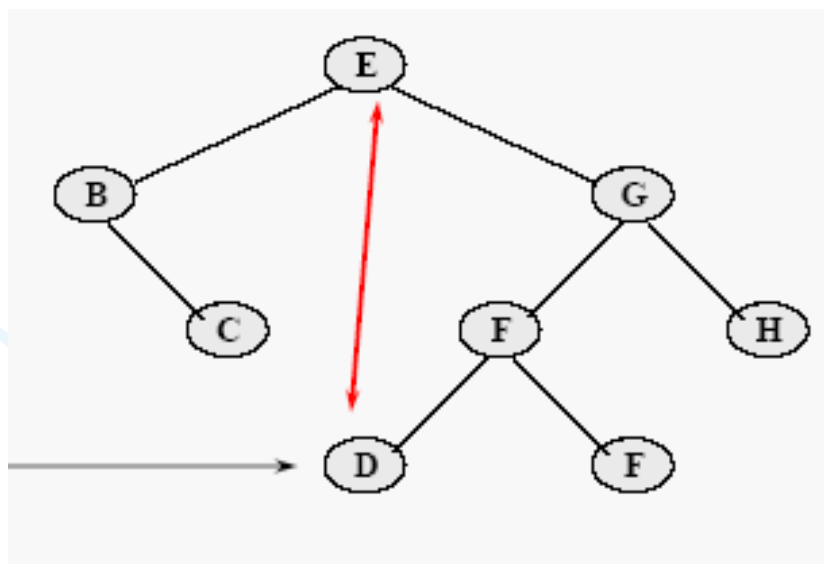
- Xóa một nút lá là công việc đơn giản: thiết lập con trỏ tương ứng của nút cha là NULL
- Xóa một nút trong chỉ có duy nhất 1 cây con cũng đơn giản: thiết lập con trỏ tương ứng của nút cha tới con của nút này





# Xóa một nút khỏi BST

- Xóa một nút con có 2 cây con: khó khăn hơn
  - Tìm nút trái nhất của cây con phải => đổi chỗ giá trị của nút này với nút cần xóa
  - Xóa nút này (nút trái nhất của cây con phải)



# Tìm nút trái nhất của cây con phải

- Tìm nút trái nhất của cây con phải và xóa

```
KeyType DeleteMin (TreeType *Root ) {
    KeyType k;
    if ((*Root)->left == NULL) {
        k=(*Root)->key;
        (*Root) = (*Root)->right;
        return k;
    }
    else return DeleteMin(&(*Root)->left) ;
}
```

# Xóa một nút từ BST

```
void DeleteNode(key X, TreeType *Root) {
    if (*Root != NULL)
        if (x < (*Root)->Key) DeleteNode(x, &(*Root)->left)
        else if (x > (*Root)->Key)
            DeleteNode(x, &(*Root)->right)
        else if
            (((*Root)->left == NULL) && ((*Root)->right == NULL))
                *Root = NULL;
        else if ((*Root)->left == NULL)
            *Root = (*Root)->right
        else if ((*Root)->right == NULL)
            *Root = (*Root)->left
        else (*Root)->Key = DeleteMin(&(*Root)->right);
}
```



# Exercise

- Viết hàm xóa toàn bộ nút trên cây.  
Hàm này được gọi trước khi kết thúc chương trình



# Solution

```
void freetree(TreeType tree)  
{  
    if (tree!=NULL)  
    {  
        freetree(tree->left);  
        freetree(tree->right);  
        free((void *) tree);  
    }  
}
```



# Exercise

- Tạo cây nhị phân tìm kiếm có 10 nút. Mỗi nút là một số nguyên được khởi tạo ngẫu nhiên
- Nhập từ bàn phím một số nguyên => tìm kiếm số nguyên này
- Gợi ý: tạo số ngẫu nhiên  
`srand (time (NULL) ) ;`  
`rand () % MAX ;`

# Solution

```
#include <stdio.h>
#include <stdlib.h>
#include <bsttree.h> // create by your self
#include <time.h>
int main() {
    TreeType p, tree = NULL;
    int i, n = 0;
    srand(time(NULL));
    for ( i = 0; i < 10; i++ )
        insert (rand() % 100, tree );
    printf("pretty print:\n");
    strcpy(prefix, "    ");
    prettyprint(tree, prefix);
    printf("\n");
    do {
        printf("Enter key to search (-1 to quit):");
        scanf("%d", &n);
        p = Search(n, tree);
        if (p != NULL) printf("Key %d found on the tree", n);
        else insert(n, tree);
        while (n != -1);
        return 0;
    }
```

# Exercise

We assume that you make a mobile phone's address book.

- Declare a structure which can store at least "name", "telephone number", "e-mail address."
  - Declare a structure for a binary tree which can stores the structure of an address book inside. Read data of about 10 from an input file to this binary tree as the following rules.
    - An address data which is smaller in the dictionary order for the **e-mail address** is stored to the left side of a node.
    - An address data which is larger in the dictionary order for the e-mail address is stored to the right side of a node.
- (1) Confirm the address data is organized in the binary tree structure with some methods (printing, debugger, etc).
  - (2) Find a specified e-mail address in the binary tree and output it to a file if found.
  - (3) Output all the data stored in the binary tree in ascending order for the e-mail address. (Reserve it for the next week)



# Solution

```
#include <stdio.h>
#define MAX 20
typedef struct phoneaddress_t {
    char name[20];
    char tel[11];
    char email[25];
}phoneaddress;

typedef struct Node{
    phoneaddress key;
    struct Node* Left,Right;
} NodeType;
typedef Node* TreeType;
```



# Search function

```
TreeType Search(char* email, TreeType Root) {  
    if (Root == NULL) return NULL; // not found  
    else if (strcmp((Root->Key).email, email) == 0)  
        return Root;  
    else if (strcmp((Root->Key).email, email) < 0)  
        //continue searching in the right sub tree  
        return Search(email, Root->right);  
    else {  
        // continue searching in the left sub tree  
        return Search(email, Root->left);  
    }  
}
```

# Insert a node

```
void InsertNode(phoneaddress x, TreeType *Root ) {  
    if (*Root == NULL) {  
        *Root = (NodeType*) malloc (sizeof (NodeType)) ;  
        (*Root) -> Key = x ;  
        (*Root) -> left = NULL ;  
        (*Root) -> right = NULL ;  
    }  
    else if (strcmp ((*Root) -> Key).email, x.email) > 0)  
        InsertNode(x, (*Root) -> left) ;  
    else if (strcmp ((*Root) -> Key).email, x.email) < 0)  
        InsertNode(x, (*Root) -> right) ;  
}
```

# Solution

```
int main(void)
{
    FILE *fp;
    phoneaddress phonearr[MAX];
    treetype root;
    int i,n, irc; // return code
    int reval = SUCCESS;
    int n=10;
    //read from this file to array again
    if ((fp = fopen("phonebook.dat","rb")) == NULL) {
        printf("Can not open %s.\n", "phonebook.dat");
        reval = FAIL;
    }
    irc = fread(phonearr, sizeof(phoneaddress), n, fp);
    fclose(fp);
}
```

# Solution

```
. . . ;  
for (i=0; i<n; i++)  
    root = InsertNode(phonearr[i],root) ;  
pretty_print(root,0) ;  
// Search for an email  
// Do it by your self  
. . .  
}
```