

The background features several large, flowing, abstract shapes in light green, light blue, and light purple. Interspersed among these are numerous small, yellow, triangular shapes pointing in various directions, creating a dynamic and celebratory feel.

Data Structure & Algorithm Basic Lab – week 8

The slide features a decorative background on the left side with a light green balloon at the top, a light blue balloon in the middle, and a light purple balloon at the bottom. Yellow streamers and triangular shapes are scattered around the balloons. The main content is centered on the right side of the slide.

Topics of this week

- How to use debugger tool(gdb)
- Tree data structure
 - Binary Tree
 - Binary Search Tree
- Recursive processing on Tree



gdb for debugging (1)

- **gdb**: the Gnu DeBugger
- <http://www.cs.caltech.edu/courses/cs11/material/c/mike/misc/gdb.html>
- Use when program core dumps
- or when want to walk through execution of program line-by-line

`gdb` for debugging (2)

- Before using `gdb`:
 - Must compile C code with additional flag:
`-g`
 - This puts all the source code into the binary executable
- Then can execute as: `gdb myprogram`
- Brings up an interpreted environment

`gdb` for debugging (3)

```
gdb> run
```

- Program runs...
- If all is well, program exits successfully, returning you to prompt
- If there is (*e.g.*) a core dump, `gdb` will tell you and abort the program



`gdb` – basic commands (1)

- Stack backtrace ("`where`")
 - Your program core dumps
 - Where was the last line in the program that was executed before the core dump?
 - That's what the `where` command tells you

gdb - basic commands (2)

gdb> where

last call

last call in your code

```
#0 0x4006cb26 in free () from /lib/libc.so.6
```

```
#1 0x4006ca0d in free () from /lib/libc.so.6
```

```
#2 0x8048951 in board_updater (array=0x8049bd0,  
ncells=2) at 1dCA2.c:148
```

```
#3 0x80486be in main (argc=3, argv=0xbffff7b4) at  
1dCA2.c:44
```

```
#4 0x40035a52 in __libc_start_main () from  
/lib/libc.so.6
```

stack backtrace

`gdb` – basic commands (3)

- Look for topmost location in stack backtrace that corresponds to your code
- Watch out for
 - freeing memory you didn't allocate
 - accessing arrays beyond their maximum elements
 - dereferencing pointers that don't point to part of a `malloc()` ed block

gdb – basic commands (4)

- **break, continue, next, step** commands
- **break** causes execution to stop on a given line

```
gdb> break foo.c: 100    (setting a  
breakpoint)
```

- **continue** resumes execution from that point
- **next** executes the next line, then stops
- **step** executes the next statement
 - goes into functions if necessary (**next** doesn't)

gdb – basic commands (5)

- **print** and **display** commands
- **print** prints the value of any program expression

```
gdb> print i
```

```
$1 = 100
```

- **display** prints a particular value every time execution stops

```
gdb> display i
```

`gdb` – printing arrays (1)

- `print` will print arrays as well

```
int arr[] = { 1, 2, 3 };
```

```
gdb> print arr
```

```
$1 = {1, 2, 3}
```

- N.B. the `$1` is just a name for the result

```
print $1
```

```
$2 = {1, 2, 3}
```

`gdb` - printing arrays (2)

- `print` has problems with dynamically-allocated arrays

```
int *arr;
```

```
arr = (int *)malloc(3 * sizeof(int));
```

```
arr[0] = 1; arr[1] = 2; arr[2] = 3;
```

```
gdb> print arr
```

```
$1 = (int *) 0x8094610
```

- Not very useful...

`gdb` – printing arrays (3)

- Can print this array by using `@` (`gdb` special syntax)

```
int *arr;
```

```
arr = (int *)malloc(3 * sizeof(int));
```

```
arr[0] = 1; arr[1] = 2; arr[2] = 3;
```

```
gdb> print *arr@3
```

```
$2 = {1, 2, 3}
```



`gdb` – abbreviations

- Common `gdb` commands have abbreviations
 - `p` (same as `print`)
 - `c` (same as `continue`)
 - `n` (same as `next`)
 - `s` (same as `step`)
- More convenient to use when interactively debugging

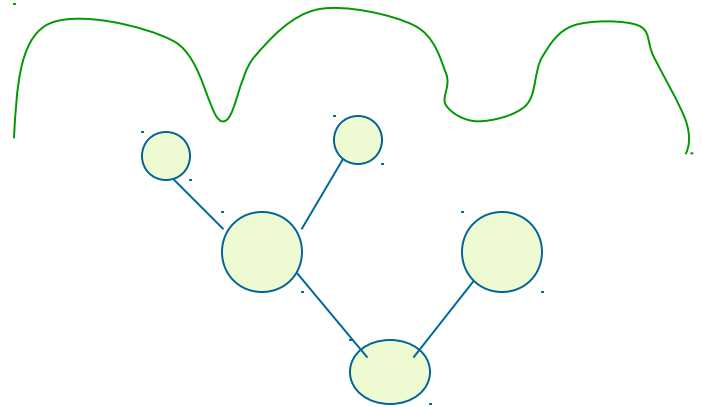
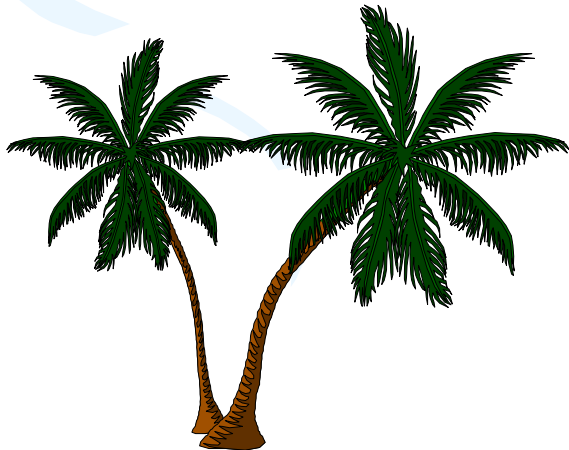
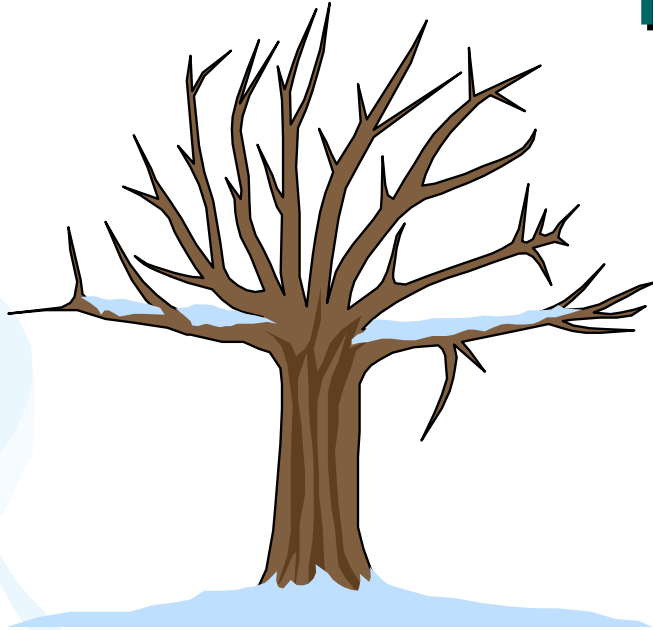
other instruction

- clear : delete break point of current file.
- delete [break position]: delete breakpoint at a specific file and position
- Conditional break

```
gdb> break foo.c: 100 if i==-1
```

- quit
- run: restart from beginning.

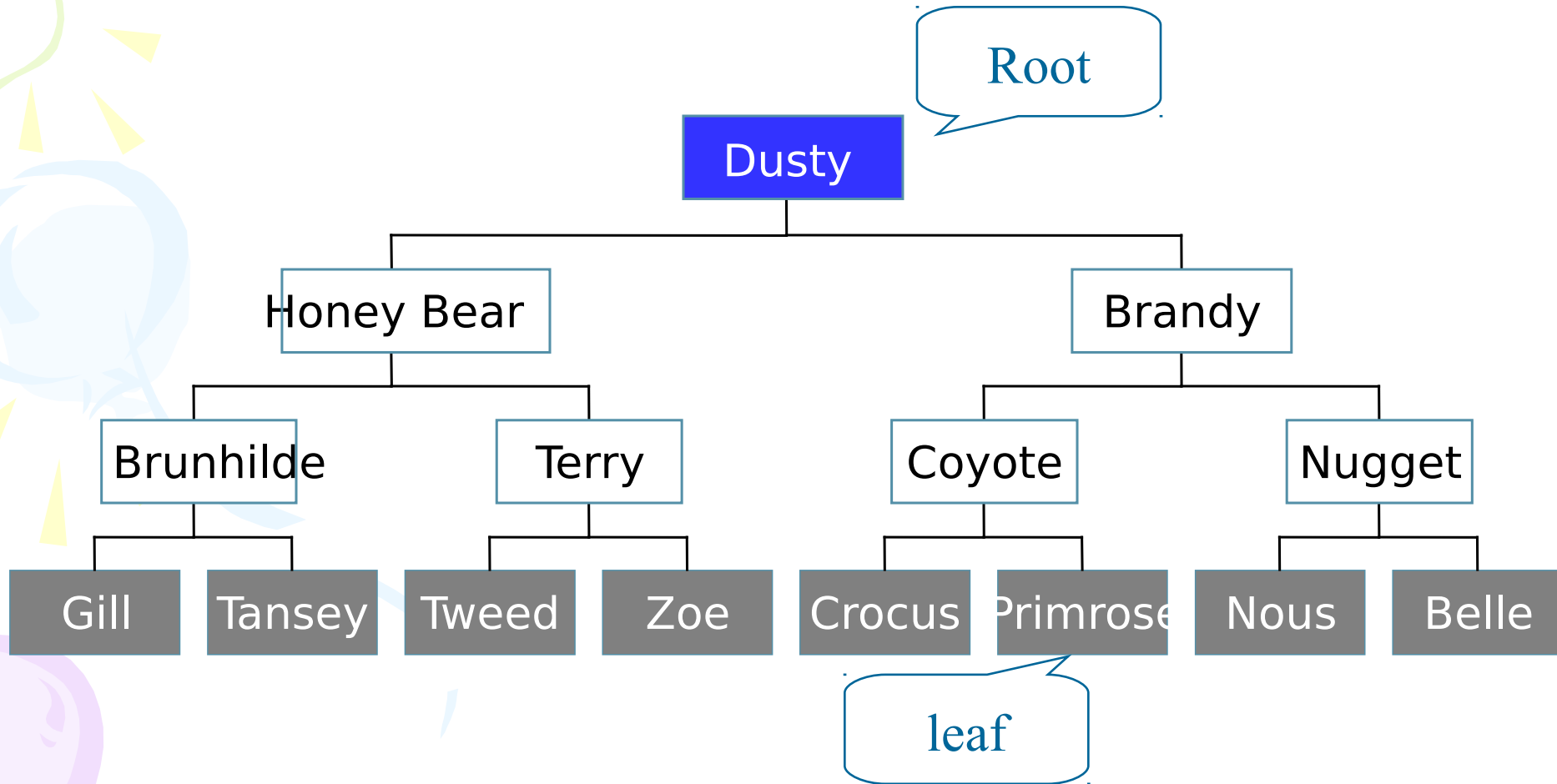
Tree



Trees, Binary Trees, and Binary Search Trees

- Linked lists are **linear structures** and it is difficult to use them to organize a **hierarchical** representation of objects.
- Although stacks and queues reflect some hierarchy, they are limited to only **one dimension**.
- To overcome this limitation, we create a new data type called a **tree** that consists of **nodes** and **arcs**. Unlike natural trees, these trees are **depicted upside down** with the **root** at the top and the **leaves** at the bottom.

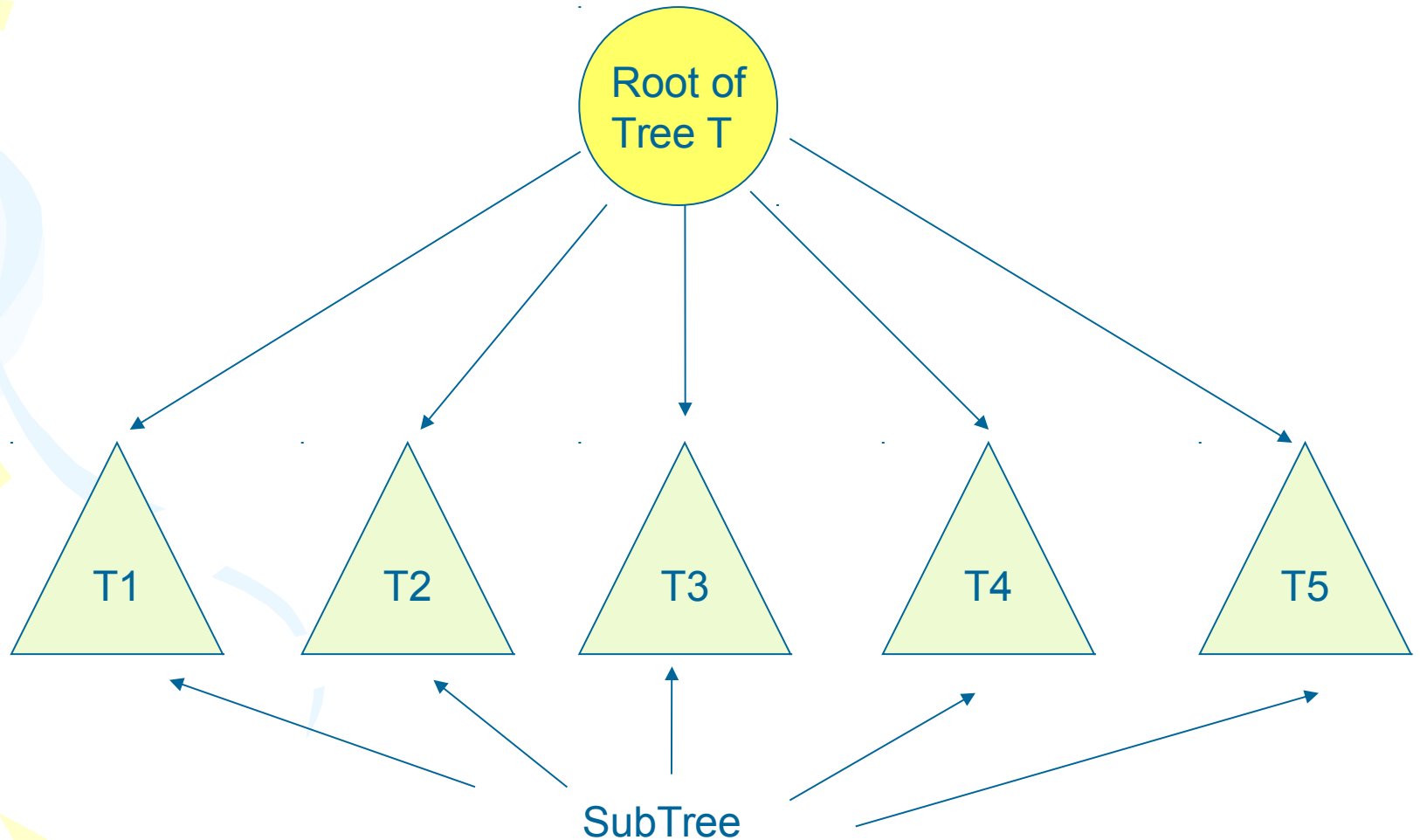
Family Tree



Definition of tree

- A tree is a finite set of one or more nodes such that:
- There is a specially designated node called the root.
- The remaining nodes are partitioned into $n \geq 0$ disjoint sets T_1, \dots, T_n , where each of these sets is a tree.
- We call T_1, \dots, T_n the subtrees of the root.

Recursive definition



A decorative background on the left side of the slide features a light green balloon at the top, a light blue balloon in the middle, and a light purple balloon at the bottom. Yellow streamers and triangular flags are scattered around the balloons.

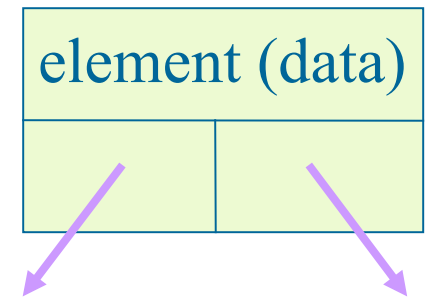
Binary Tree

- A binary tree is a tree in which no node can have more than two children.
- Each node has 0, 1, or 2 children

Linked Representation

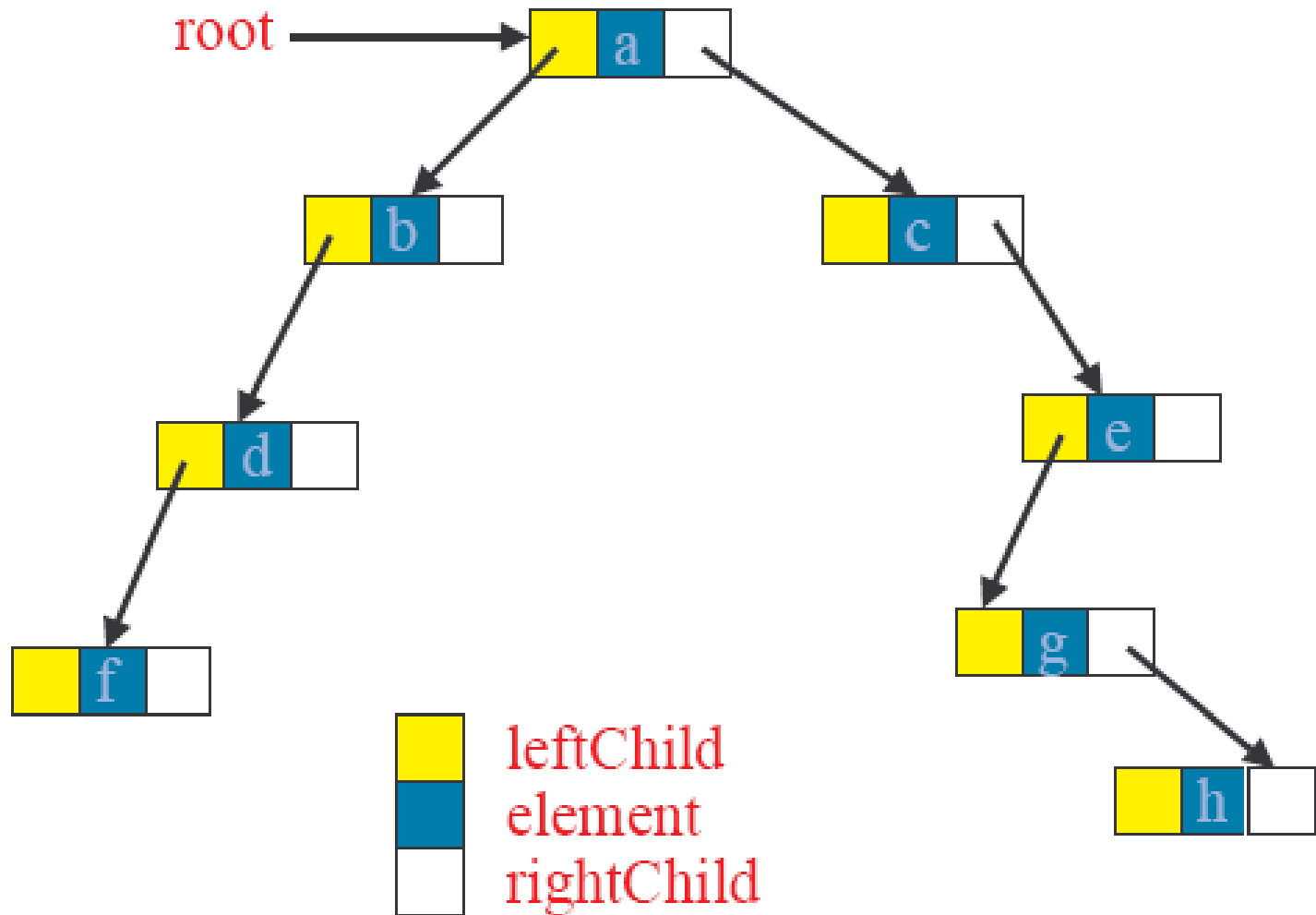
- Each tree node is represented as an object whose data type is
- The space required by an n node binary tree is $n * (\text{space required by one node})$

```
typedef ... elmType;
//whatever type of element
typedef struct nodeType {
    elmType element;
    struct nodeType *left, *right;
};
typedef struct nodeType *treeType;
```



left child right
child

A linked binary tree



A decorative background on the left side of the slide featuring a green balloon at the top, a light blue balloon in the middle, and a purple balloon at the bottom. Yellow streamers and triangular confetti are scattered around the balloons.

Binary Tree ADT

- `makeNullTree(treeType *t)`
- `creatNewNode()`
- `isEmpty()`

Tree initialization and verification

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

typedef struct nodeType *treeType;

void makeNullTree(treeType *T) {
    (*T)=NULL;
}

int emptyTree(treeType T) {
    return T==NULL;
}
```

Access left and right child

```
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;
}
```

create a new node

```
nodeType *createNode(elmType NewData)
{
    nodeType *N;
    N=(nodeType*)malloc(sizeof(nodeType));
    if (N != NULL)
    {
        N->left = NULL;
        N->right = NULL;
        N->element = NewData;
    }
    return N;
}
```

check if a node is a leaf

```
int isLeaf(treeType n) {  
    if (n != NULL)  
        return (leftChild(n) == NULL) &&  
            (rightChild(n) == NULL);  
    else return -1;  
}
```

Recursive processing: Number of nodes

- As tree is a recursive data structure, recursive algorithms are useful when they are applied on tree.

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

Create a tree from two subtrees

```
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;  
}
```

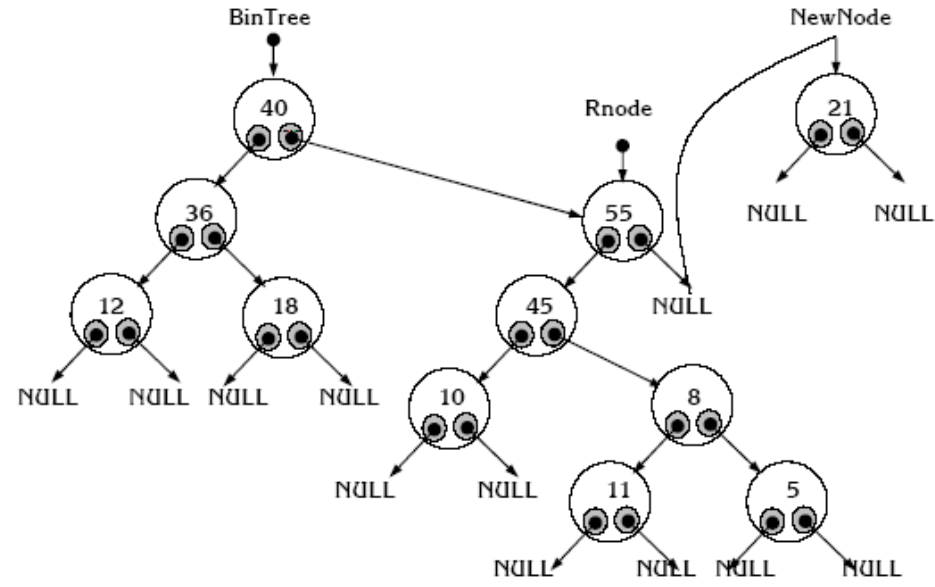
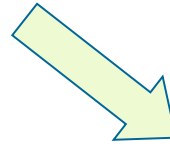
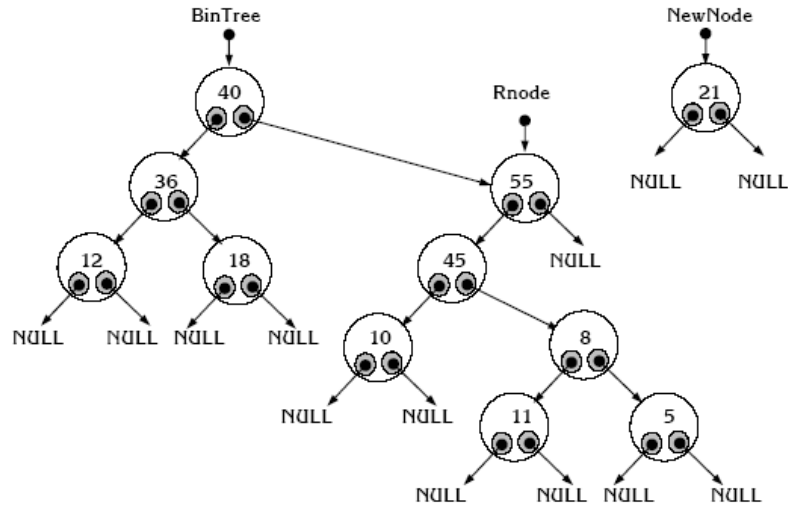
Adding a new node to the left most position

```
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);
}
```

Adding a new node to the right most position

```
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 *Rnode = *Tree;
    while (Rnode->right != NULL)
      Rnode = Rnode->right;
    Rnode->right = NewNode;
  }
  return (NewNode);
}
```


Illustration



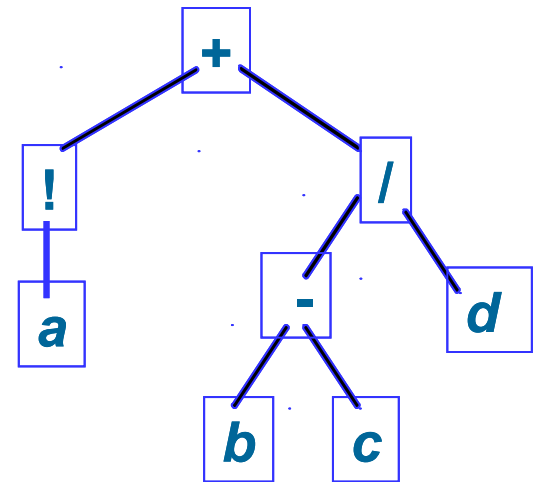
The slide features a decorative background on the left side with a light green balloon at the top, a light blue balloon in the middle, and a light purple balloon at the bottom. Yellow streamers and triangular shapes are scattered around the balloons. The main content is centered on the right side of the slide.

Exercise (BTVN)

- Develop the following helper functions for a tree:
 - return the height of a binary tree.
 - return the number of leafs
 - return the number of internal nodes
 - count the number of right children.

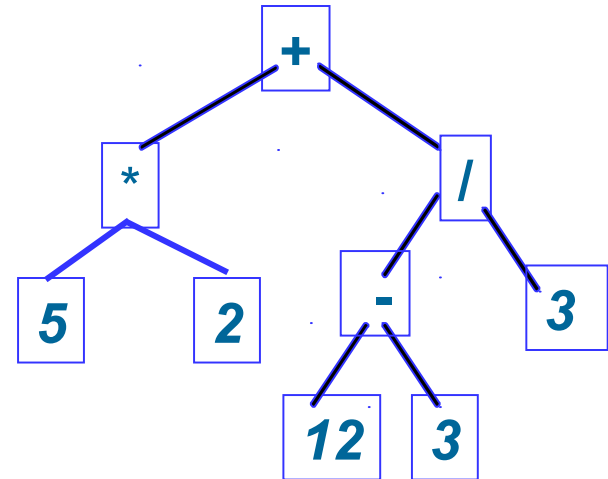
Exercise

- A binary tree can represent an arithmetic expression:
The leaves are operands and the other nodes are operators.
- The left and right subtrees of an operator node represent **subexpressions** that must be evaluated **before** applying the operator at the root of the subtree.
- For example
 $a + (b - c)/d$
- Write a program create a tree representing this expression



Exercise at Home

- Write an menu program that take a valid arithmetic expression as input and:
 - Store and represent it in a tree
 - Evaluate the expression.

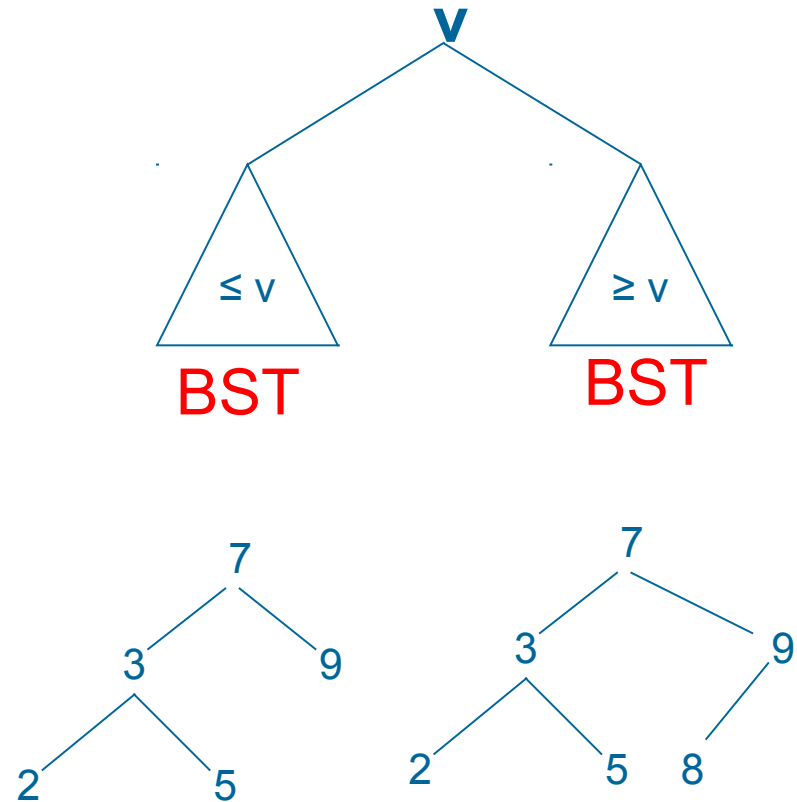


BTVN 4EF

- Tạo file văn bản USopen.txt có nội dung sau:
 - Dòng 1 chứa 16 đối thủ
- Xây dựng cây mô tả kết quả thi đấu cho tới trận chung kết. Ban đầu 16 đối thủ là các nút lá.
- Lựa chọn người chơi thắng trong một trận đấu là ngẫu nhiên.
- In kết quả ra file treegame.txt

Binary Search Tree

- Every element has a unique key.
- The keys in a nonempty left subtree (right subtree) are smaller (larger) than the key in the root of subtree.
- The left and right subtrees are also binary search trees.



Binary Search Tree Implementation

```
#include <stdio.h>
#include <stdlib.h>
typedef . . . KeyType; // specify a type
for the data
typedef struct Node{
    KeyType key;
    struct Node* left, right;
} NodeType;
typedef Node* TreeType;
```

Search on 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);
    }
}
```


Insert a node from a BST

- In a binary, there are not two nodes with the same key.

```
void InsertNode(KeyType x, TreeType *Root ) {
    if (*Root == NULL) {
        /* Create a new node for key x */
        *Root = (NodeType*) malloc (sizeof (NodeType));
        (*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);
}
```

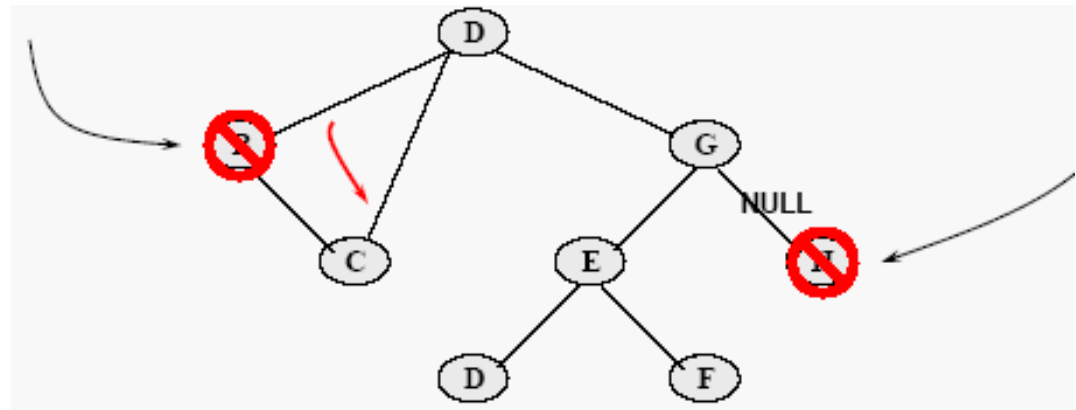
Insert a node from a BST

- Version with the return type

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

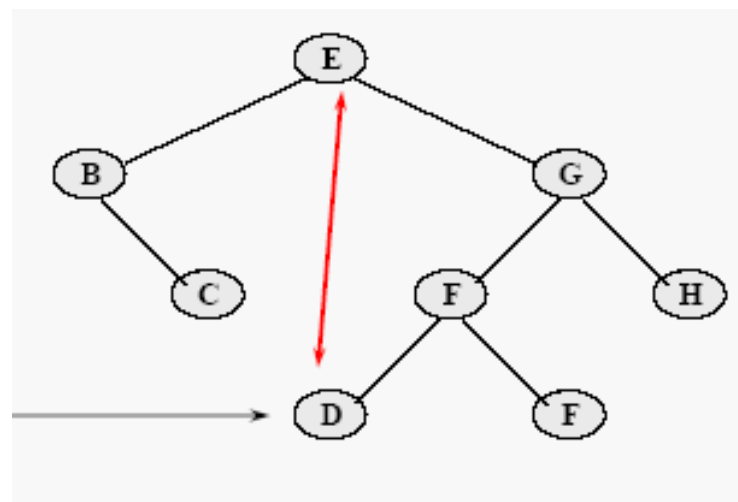
Delete a node from a BST

- Removing a leaf node is trivial, just set the relevant child pointer in the parent node to NULL.
- Removing an internal node which has only one subtree is also trivial, just set the relevant child pointer in the parent node to target the root of the subtree.



Delete a node from a BST

- Removing an internal node which has two subtrees is more complex
 - Find the left-most node of the right subtree, and then swap data values between it and the targeted node.
 - Delete the swapped value from the right subtree.



Find the left-most node of right sub tree

- This function find the leftmost node then delete it.

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

Delete a node from a 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);
}
```

Pretty print a BST

```
void prettyprint(TreeType tree, char *prefix) {
    char *prefixend=prefix+strlen(prefix);
    if (tree!=NULL) {
        printf("%04d", tree->key);
        if (tree->left!=NULL) if (tree->right==NULL) {
            printf("\304"); strcat(prefix, "    ");
        }
        else {
            printf("\302"); strcat(prefix, "\263    ");
        }
        prettyprint(tree->left, prefix);
        *prefixend='\0';
        if (tree->right!=NULL) if (tree->left!=NULL) {
            printf("\n%s", prefix); printf("\300");
        } else printf("\304");
        strcat(prefix, "    ");
        prettyprint(tree->right, prefix);
    }
}
```



Exercise

- Write a function to delete all node of a tree. This function must be called before terminating program.

Solution

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

Exercise

- Create a binary search tree with 10 nodes. Each node contains a random integer.
- Ask user to input a number and search for it.
- Print the content of the trees.

Min Max trong BST

- Viết hai hàm FindMin – FindMax cho thư viện cây nhị phân tìm kiếm.
- Tham số: con trỏ root
- Return: con trỏ tới nút nhỏ nhất hay lớn nhất.

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 – BTVN1

- 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 store 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  
    . . .  
}
```

Bài tập về nhà 2

- Viết chương trình từ điển thuật ngữ tin học Anh Việt. Tiếng Việt không dấu.
- Gợi ý: Một mục từ gồm: từ tiếng Anh – giải nghĩa tiếng Việt.
- Dùng cấu trúc BST để lưu trữ.
- Chương trình phải cho phép: tìm kiếm, thêm từ, xóa từ và ghi dữ liệu vào file.
- Mở rộng: Mỗi từ có 1 danh sách các từ đồng nghĩa.

Bài tập về nhà 3

- Kiểm tra tốc độ tìm kiếm của một cây nhị phân như sau.
- Sinh ra 1 triệu số nguyên ngẫu nhiên (chú ý đảm bảo tính ngẫu nhiên) và cho vào cây. Chú ý nếu một số đã nằm trong cây thì không cần bổ sung.
- Sau đó in ra chiều cao của cây.
- Chương trình cho phép:
 - Tạo lại bộ số liệu (Chú ý giải phóng bộ nhớ dành cho cây cũ).
 - Thực hiện tìm kiếm và in ra số phép so sánh.

Ghi chú

- Nộp bài tập – cần nộp kèm thư viện đã làm trên lớp / phát triển từ các buổi thực hành trước.
- Tách riêng thư viện cấu trúc dữ liệu và chương trình chính.s

BTVN ICT54C

- Tiếp tục phát triển Chương trình CSDL Điện thoại NokiaDB.
- Viết lại chương trình sử dụng cây nhị phân tìm kiếm. Khóa của 1 nút là Model của điện thoại.
- Chương trình có các chức năng: Import, Thêm, Xóa, Cập Nhật, Tìm Kiếm, Hiển thị danh sách.