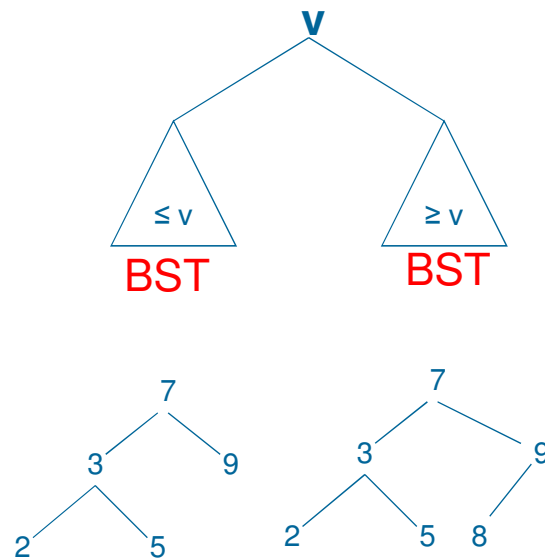


# Binary Search Tree

- 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);
}
```

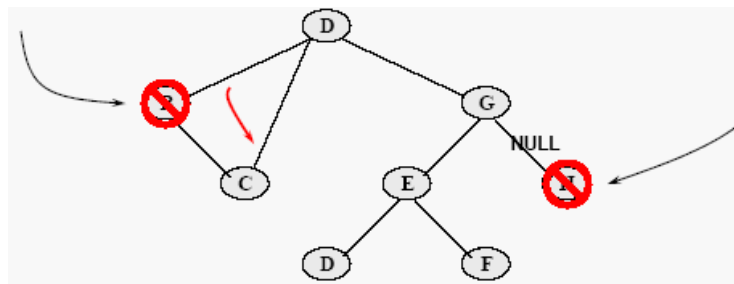


## Driver program

```
int main() {  
    int a[10]={4,9,5,100,26,-  
18,13,77,1,34};  
    int i;  
    TreeType root;  
    for(i=0;i<10;i++)  
        InsertNode(a[i], &root);  
    InOrderPrint(root);  
    return 0;  
}
```

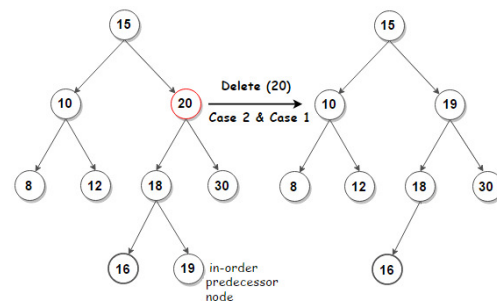
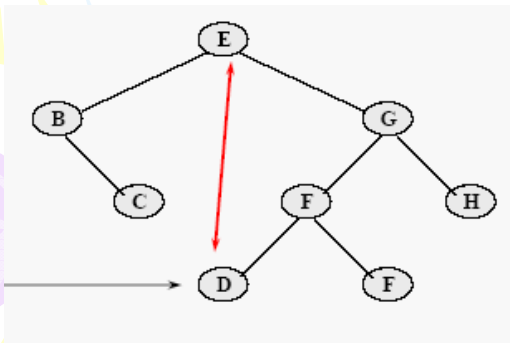
# Delete a node from a BST

- 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);
    }
}
```



## Lab 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.

# 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;
    }
```



## Homework: Finding Min Max in BST

- Write two FindMin - FindMax functions and add them to the binary search tree library.
- Argument: root pointer
- Return: pointer that point to the minumum node/maximum node.





# Homework

- Create a student list file of your class.
- Write a student management program that uses a binary search tree with nodes that are student names, knowing that the comparison between names based on the Vietnamese name matching principle is as follows:
  - Compare first name (in alphabetical order), then middle name, then last name
  - The program has the function of displaying the class list sorted in ascending order to the screen and searching students



# 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;
```




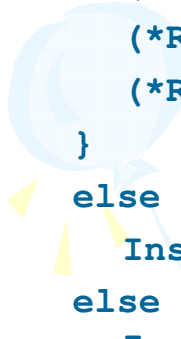
## Search function

```
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

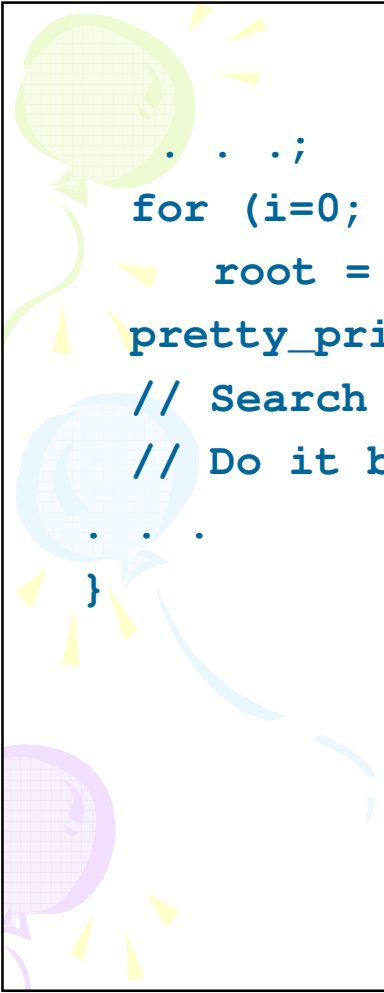
```
void InsertNode(phoneaddress x, TreeType *Root ){
    if (*Root == NULL){
        *Root=(NodeType*)malloc(sizeof(NodeType));
        (*Root)->Key = x;
        (*Root)->left = NULL;
        (*Root)->right = NULL;
    }
    else if (strcmp((*Root)->Key).email, x.email) > 0)
        InsertNode(x, (*Root)->left);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



# Homework: Dictionary in ICT

- Write a English-Vietnamese dictionary in ICT program with basic functionalities.
- Hint:
  - An item in the dictionary includes: english word – description in Vietnamese (you can add some example sentences or list of synonyms but they are optional).
  - BST should be used.
- Main features: searching, adding word item, delete word and save data to file.





## Homework: Evaluate the searching performance of BST

- Write a program that generates 1 million different random integers and loads them into the tree. Take care not to add a number if it is already in the tree.
- The program allows:
  - Calculate the tree height.
  - Regenerate the dataset (Note: need to free memory for old trees).
  - Perform a search and display the number of comparisons to get the result.



## Note

- When submitting assignments - attach a library that has been instructed in class and developed from previous practice sessions.
- Separate the data structure library and the main program.



# Homework

- Rewrite PhoneDB management program using BST. The searching key in a node is the phone model.
- Required Features: Import data from text file, Import data from dat file, Add phone model, Delete, Ipda, Searching, Displaying and Save data to File Phone.dat.

A decorative graphic on the left side of the slide, featuring overlapping circles in light green, light blue, and light purple, with yellow triangular rays emanating from them.

## Representing arithmetic expression using binary tree

- Write a program that builds a binary tree for representing a postfix arithmetic expression.
- Traverse the tree in middle order to check if the tree is correctly built.
- Evaluate the expression.

# Evaluation Expression using Binary Tree

```
int Eval ( TreeNode* ptr )
{
    switch ( ptr->info.whichType )
    {
        case OPERAND : return ptr->info.operand ;
        case OPERATOR :
            switch ( ptr->info.operation )
            {
                case '+' : return ( Eval ( ptr->left ) + Eval ( ptr->right ) ) ;
                case '-' : return ( Eval ( ptr->left ) - Eval ( ptr->right ) ) ;
                case '*' : return ( Eval ( ptr->left ) * Eval ( ptr->right ) ) ;
                case '/' : return ( Eval ( ptr->left ) / Eval ( ptr->right ) ) ;
            }
        }
    }
}
```

A decorative graphic on the left side of the slide, consisting of several overlapping circles in light green, light blue, and light purple, with small yellow triangles scattered around them.

## Constructing an Expression Tree

- There is a simple  $O(N)$  stack-based algorithm to convert a postfix expression into an expression tree.
- Recall we also have an algorithm to convert an infix expression into postfix, so we can also convert an infix expression into an expression tree without difficulty (in  $O(N)$  time).

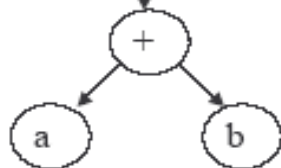


# Expression Tree Algorithm

- Read the postfix expression one symbol at time:
  - If the symbol is an operand, create a one-node tree and push a pointer to it onto the stack.
  - If the symbol is an operator, pop two tree pointers T1 and T2 from the stack, and form a new tree whose root is the operator, and whose children are T1 and T2.
  - Push the new tree pointer on the stack.

# Example

a b + :

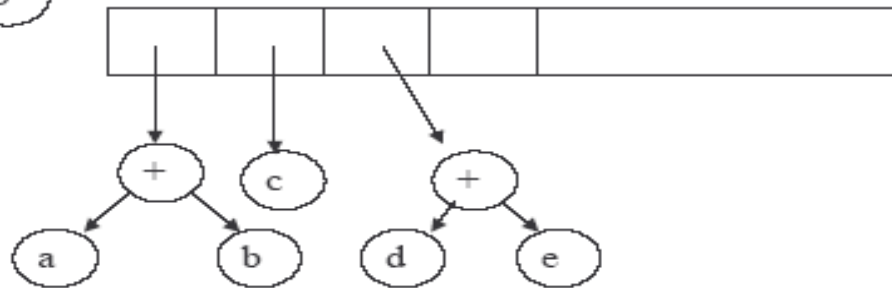
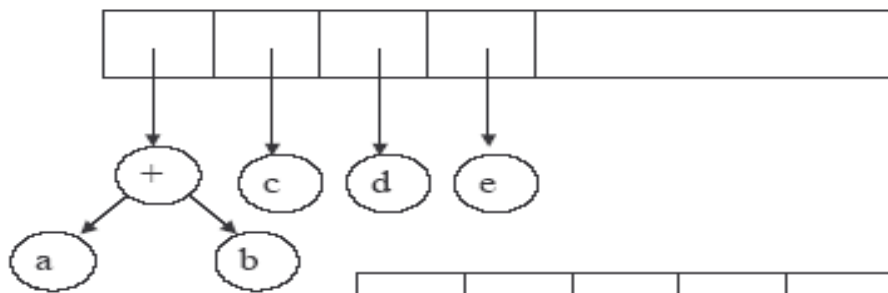


Note: These stacks are depicted horizontally.



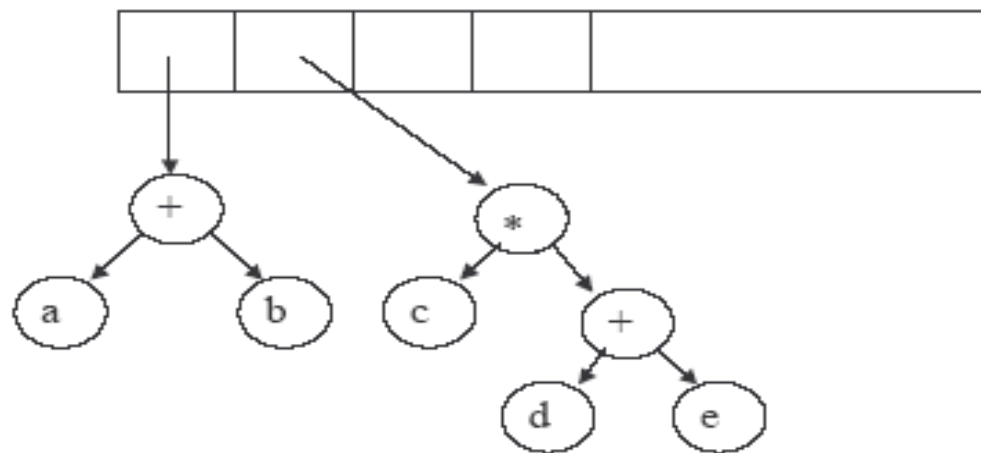
# Example

a b + c d e + :



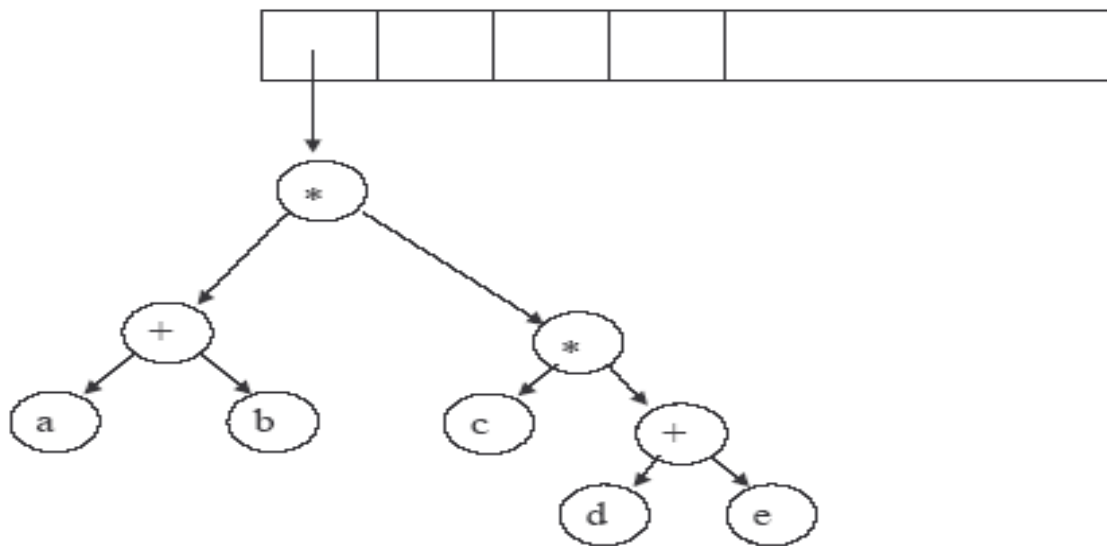
# Example

$ab + cde + * :$



# Example

**a b + c d e + \* \* :**





# Customer Service Application

- **BestBuy Supermarket manages the shopping information of customers from which to build an appropriate business strategy. Each customer is given a card with a code. When customers pay money at the supermarket, the payment amount, the most valuable commodity group will be stored in the database. There are three categories of goods: Food (TP), Electronics (DT), and Apparel (MM).**
- **The BST tree should be used with each node containing the fields for customer number, customer name, Total purchase money amount, statistics of the number of transaction customers have purchased by commodity groups. For example:**
- **CLI01234AA9    Trần Bình    12.000.230    TP = 1    DT = 6    MM=3**
- **The program reads data from text files. each line in the file is a customer transaction at the supermarket having the following structure:**
  - **CustomerID    Name    FoodAmount    ElectronicAmount    ApparelAmount**
- **Navigate Tree and display information.**



## Input.txt

- CLI0123SDH Nguyễn Trần Đạt – 230000 450000 51000
- CLI0032IPA Hoàng Thu Nga – 0 200000 31000
- CLI0124SDH Lê Phú – 0 0 121000
- CLI0032IPA Hoàng Thu Nga – 23000 110000 0
- CLI0003IPA Trần Hiếu – 630000 150000 1051000
- CLI0124SDH Lê Phú – 0 0 700000
- CLI0123SDH Nguyễn Trần Đạt – 50000 150000 33000

# Stack Exercise: Function call simulation.

- In C program, when a function is invoked, a *activation frame* with the following structure is created and push to system stack.
- Activation Frame
  - Function name
  - Return address to the caller
  - parameters of the callee
  - Locals to the callee.
- When a function finish, the corresponding AF is pop off the system stack and its value is returned back to the address of caller.
- Write a program with the menu that help you debug the functions call in a source code with the following functionality
  - 1. Invoked Function
  - 2. End Function.
  - 3. Print Top of Stack
  - Notice that you can not terminate a function that is not on top of stack. It means that this function call to another function.