HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

C Programming Basic

Trees – part 1

Manipulation with general trees

- Objectives
  - Manipulate data structures representing general tree
  - Implement fundamental operations on trees: build the tree, count number of nodes, leaves, compute height, depth of a node in the tree, ..

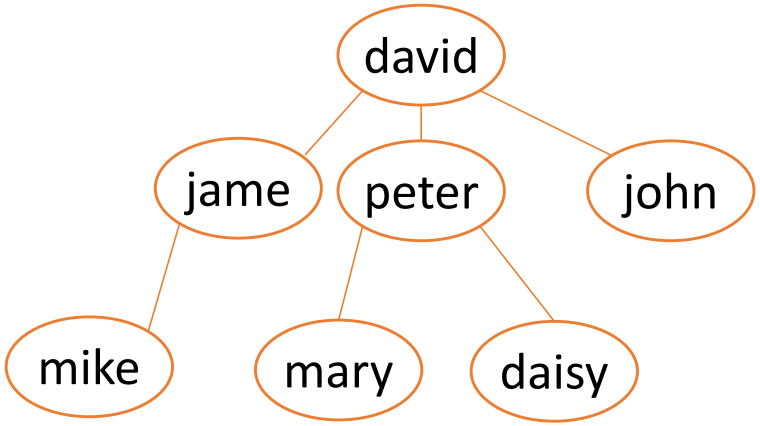
Manipulation with general trees

- Each node of a tree has the following structure

```
typedef struct Node{
    char name[256];
    struct Node* leftMostChild; // pointer to the left-most child
    struct Node* rightSibling;// pointer to the right sibling
}Node;
```

Manipulation with general trees

- The data of a tree is stored in an external text file with the format:
  - Each line contains a sequence of strings s_0, s_1, …, s_k terminated by a characer $ in which s_1, s_2, …, s_k are children of s_0 from left to right (s_1 is the left-most child) (***note***: in each line (except line 1), the string s_0 is a child of some node appearing in previous lines).
  - The file is terminated with $$




```
david jame peter john $
peter mary daisy $
jame mike $
$$
```

Manipulation with general trees

- Write a program running in an interactive mode for manipulating general trees representing members of a family with following instructions:
  - Load <filename>: load data from a text file and build the family tree
  - FindChildren <name>: print children of a given <name>
  - AddChild <name> <child>: add a new child to the children list of <name>
  - Print: print all members of the family
  - Height <name>: print the height of <name> in the tree
  - Count: print the number of members of the family
  - Store <filename>: store the family tree to a text file <filename>

Manipulation with general trees

- Define data structures and memory allocation method

```
#include <stdio.h>
typedef struct Node{
    char name[256];
    struct Node* leftMostChild;
    struct Node* rightSibling;
}Node;
Node* root;
```

```
Node* makeNode(char* name){
    Node* p =
          (Node*)malloc(sizeof(Node));
    strcpy(p->name,name);
    p->leftMostChild = NULL;
    p->rightSibling = NULL;
    return p;
}
```

Manipulation with general trees

- Find a node given its name in a tree

```
void processFind(){
    char name[256];
    scanf("%s",name);
    Node* p = find(root,name);
    if(p == NULL)
       printf("Not Found %s\n",name);
    else printf("Found %s\n",name);
}
```

```
Node* find(Node* r, char* name){
    if(r == NULL) return NULL;
    if(strcmp(r->name,name) == 0) return r;
    Node* p = r->leftMostChild;
    while(p != NULL){
        Node* q = find(p,name);
        if(q != NULL) return q;
        p = p->rightSibling;
    }
    return NULL;
}
```

Manipulation with general trees

- Adding a new node given its name to the end of the children list of another node

```
void processAddChild(){
    char name[256], child[256];
    scanf("%s%s",name,child);
    addChild(name,child);
}
```

```
Node* addLast(Node* p, char*name){
    if(p == NULL) return makeNode(name);
    p->rightSibling =
        addLast(p->rightSibling, name);
    return p;
}
void addChild(char*name, char* child){
    Node* r = find(root,name);
    if(r == NULL) return;
    r->leftMostChild =
        addLast(r->leftMostChild,child);
}
```

Manipulation with general trees

- Load data from an external file, create the tree

```
void load(char* filename){
  FILE* f = fopen(filename,"r"); root = NULL;
  while(1){
    char name[256];  fscanf(f,"%s",name);
    if(strcmp(name,"$$")==0) break;
      if(root == NULL)
        root = makeNode(name);// create the root
      while(1){
        char child[256];  fscanf(f,"%s",child);
        if(strcmp(child,"$")==0) break;
        addChild(name,child);
      }
    }
    fclose(f);
}
```

```
void processLoad(){
    char filename[256];
    scanf("%s",filename);
    load(filename);
}
```

Manipulation with general trees

- Print the information of a tree given the pointer to its root

```
void processPrint(){
    printTree(root);
}
```

```
void printTree(Node* r){
    if(r == NULL) return;
    printf("%s: ",r->name);
    Node* p = r->leftMostChild;
    while(p != NULL){
        printf("%s ",p->name);
        p = p->rightSibling;
    }
    printf("\n");
    p = r->leftMostChild;
    while(p != NULL){
        printTree(p);
        p = p->rightSibling;
    }
}
```

Manipulation with general trees

- Store the information of a tree given the pointer to its root to an external file

```
void processStore(){
    char filename[256];
    scanf("%s",filename);
    FILE* f = fopen(filename,"w");
    printTreeF(root,f);
    fprintf(f,"$$");
    fclose(f);
}
```

```
void printTreeF(Node* r, FILE* f){
    if(r == NULL) return;
    fprintf(f,"%s ",r->name);
    Node* p = r->leftMostChild;
    while(p != NULL){
        fprintf(f,"%s ",p->name);
        p = p->rightSibling;
    }
    fprintf(f," $\n");
    p = r->leftMostChild;
    while(p != NULL){
        printTreeF(p,f);
        p = p->rightSibling;
    }
}
```

Manipulation with general trees

```
void processFindChildren(){
    char name[256];    scanf("%s",name);
    Node* p = find(root,name);
    if(p == NULL) printf("Not Found %s\n",name);
    else{
        printf("Found %s with children: ",name);
        Node* q = p->leftMostChild;
        while(q != NULL){
            printf("%s ",q->name);     q = q->rightSibling;
        }
    }
    printf("\n");
}
```

Manipulation with general trees

- Compute the height of a given node

```
int height(Node* p){
    if(p == NULL) return 0;
    int maxH = 0;
    Node* q = p->leftMostChild;
    while(q != NULL){
        int h = height(q);
        maxH = maxH < h ? h : maxH;
        q = q->rightSibling;
    }
    return maxH + 1;
}
```

```
void processHeight(){
  char name[256];
  scanf("%s",name);
  Node* p = find(root,name);
  if(p == NULL) printf("Not Found %s\n",name);
  else{
    printf("Found %s having
                height = %d\n",name,height(p));
  }
}
```

Manipulation with general trees

- Count the number of nodes of a tree given its root r

```
int count(Node* r){
    if(r == NULL) return 0;
    int cnt = 1;
    Node* q = r->leftMostChild;
    while(q != NULL){
        cnt += count(q);
        q = q->rightSibling;
    }
    return cnt;
}
void processCount(){
    printf("Number of members is %d\n",count(root));
}
```

Manipulation with general trees

- Deallocate memory

```
void freeTree(Node* r){
    if(r == NULL) return;
    Node* p = r->leftMostChild;
    while(p != NULL){
        Node* sp = p->rightSibling;
        freeTree(p);
        p = sp;
    }
    printf("free node %s\n",r->name); free(r);
    r = NULL;
}
```

Manipulation with general trees

```
void main(){
    while(1){
        char cmd[256];
        printf("Enter command: ");  scanf("%s",cmd);
        if(strcmp(cmd,"Quit") == 0) break;
        else if(strcmp(cmd,"Load")==0) processLoad();
        else if(strcmp(cmd,"Print")==0) processPrint();
        else if(strcmp(cmd,"Find")==0) processFind();
        else if(strcmp(cmd,"FindChildren")==0) processFindChildren();
        else if(strcmp(cmd,"Height")==0) processHeight();
        else if(strcmp(cmd,"Count")==0) processCount();
        else if(strcmp(cmd,"AddChild")==0) processAddChild();
        else if(strcmp(cmd,"Store")==0) processStore();
    }
    freeTree(root);
}
```

Manipulation with general trees

- **Exercise** Each node of a tree has an id which is an integer. Perform a sequence of operations for building a tree and traverse the tree
  - MakeRoot u: create a root node with id = u of the tree
  - Insert u v: create a node with id = u, insert this node at the end of the children list of the node having id = v
  - PreOrder: print the sequence of nodes of the tree by pre-order traversal
  - InOrder: print the sequence of nodes of the tree by in-order traversal
  - PostOrder: print the sequence of nodes of the tree by post-order traversal
- **Input**: sequence of lines, each line is a command describing the operations described above
- Final line is * (mark the end of the data input).
- **Result**: Write on each line the sequence of nodes (separated by a SPACE character) visited by PreOrder, InOrder, PostOrder operations met in the input

Manipulation with general trees

- **Example**

Input	Output
MakeRoot 10 Insert 11 10 Insert 1 10 Insert 3 10 InOrder Insert 5 11 Insert 4 11 Insert 8 3 PreOrder Insert 2 3 Insert 7 3 Insert 6 4 Insert 9 4 InOrder PostOrder *	11 10 1 3 10 11 5 4 1 3 8 5 11 6 4 9 10 1 8 3 2 7 5 6 9 4 11 1 8 2 7 3 10

ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

soict.hust.edu.vn/ fb.com/groups/soict