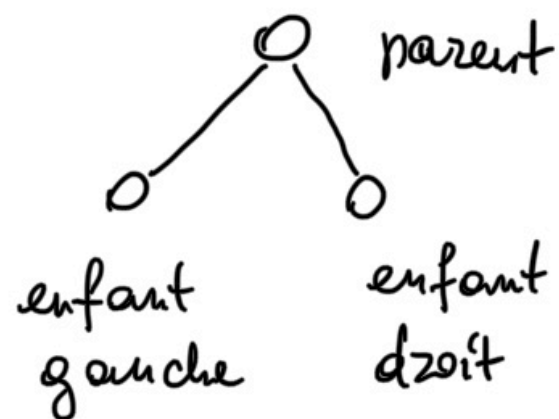
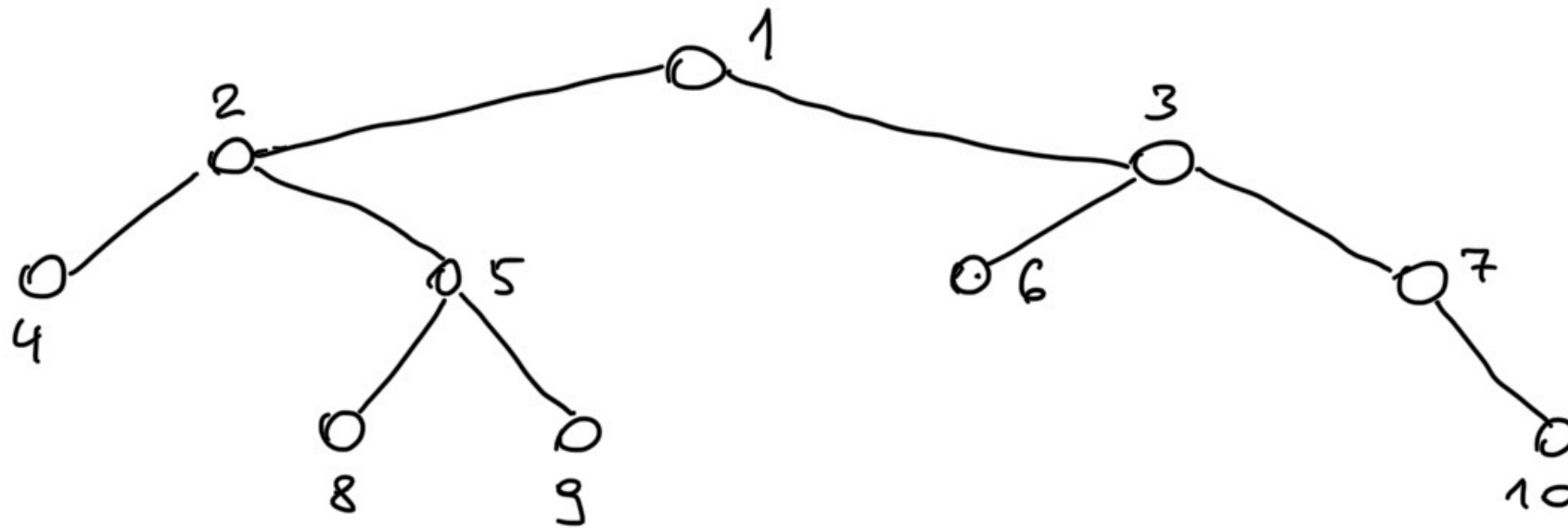


Langage C

Wieslaw Zielonka
zielonka@irif.fr

les arbres binaires

Arbre binaire



la racine - le sommet sans père

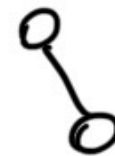
une feuille - sommet sans enfants
4, 8, 9, 6, 10

un sommet interne - sommet avec
au moins un enfant

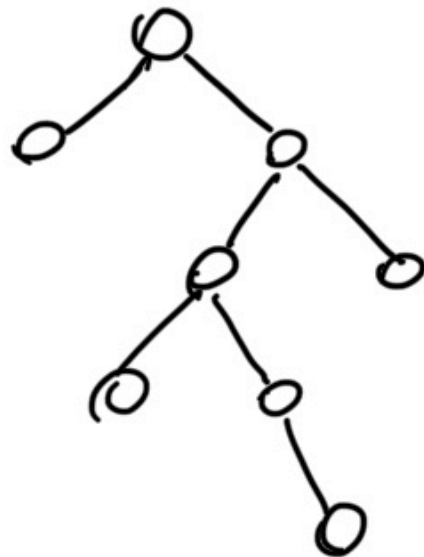
hauteur d'arbre

la hauteur de l'arbre = la longueur du plus long chemin de la racine vers une feuille

○ $h=0$
(arbre avec un sommet)



trois arbres avec $h=1$

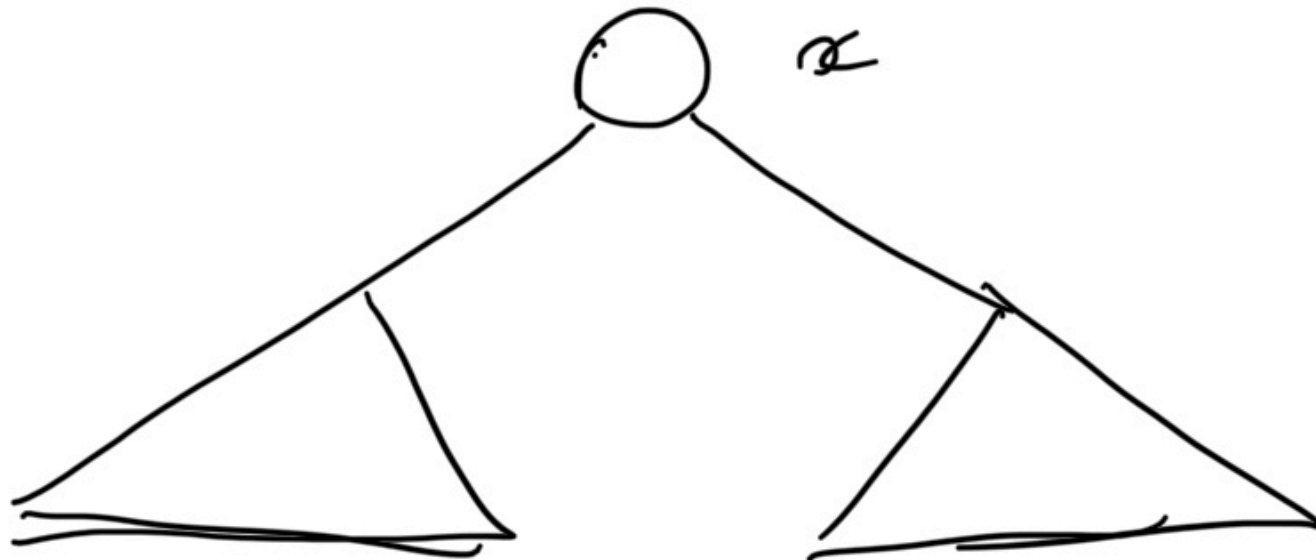


$h=4$

pour l'arbre vide
on définit $h=-1$

les sous-arbres

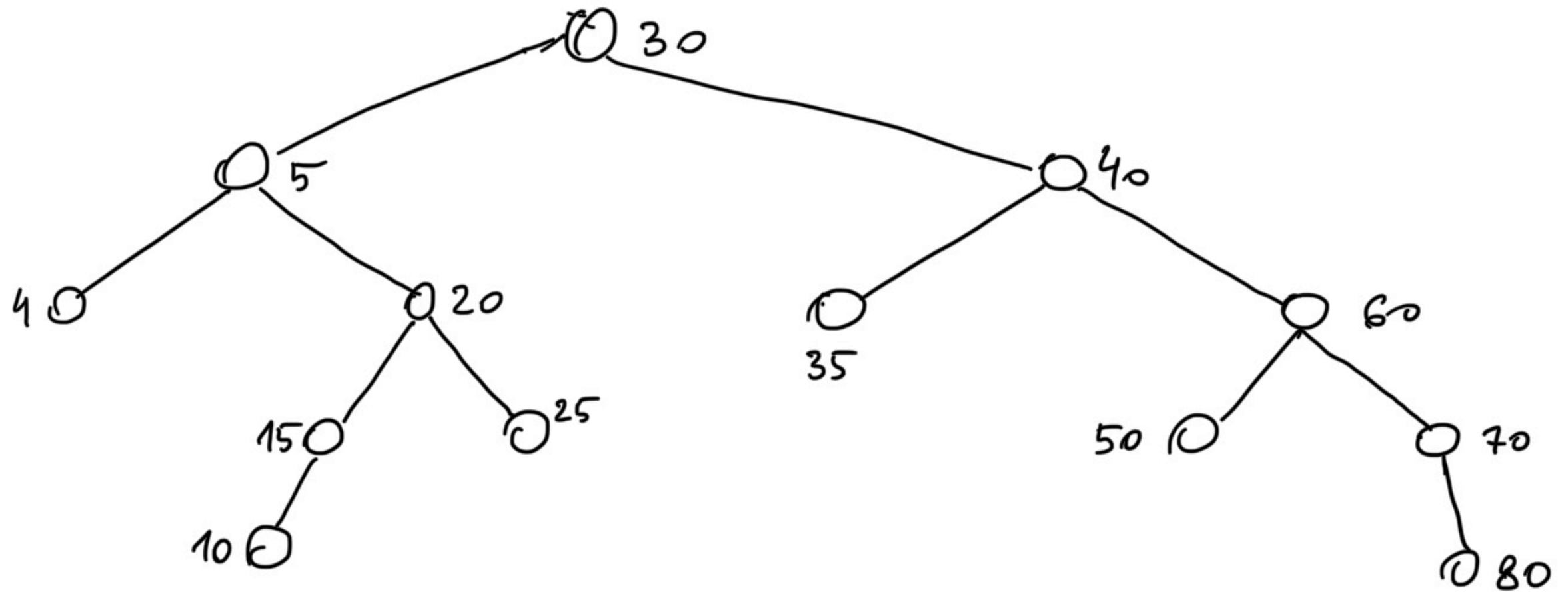
arbre vide - un arbre sans sommets



sous-arbre
gauche de x

sous-arbre
droit de x

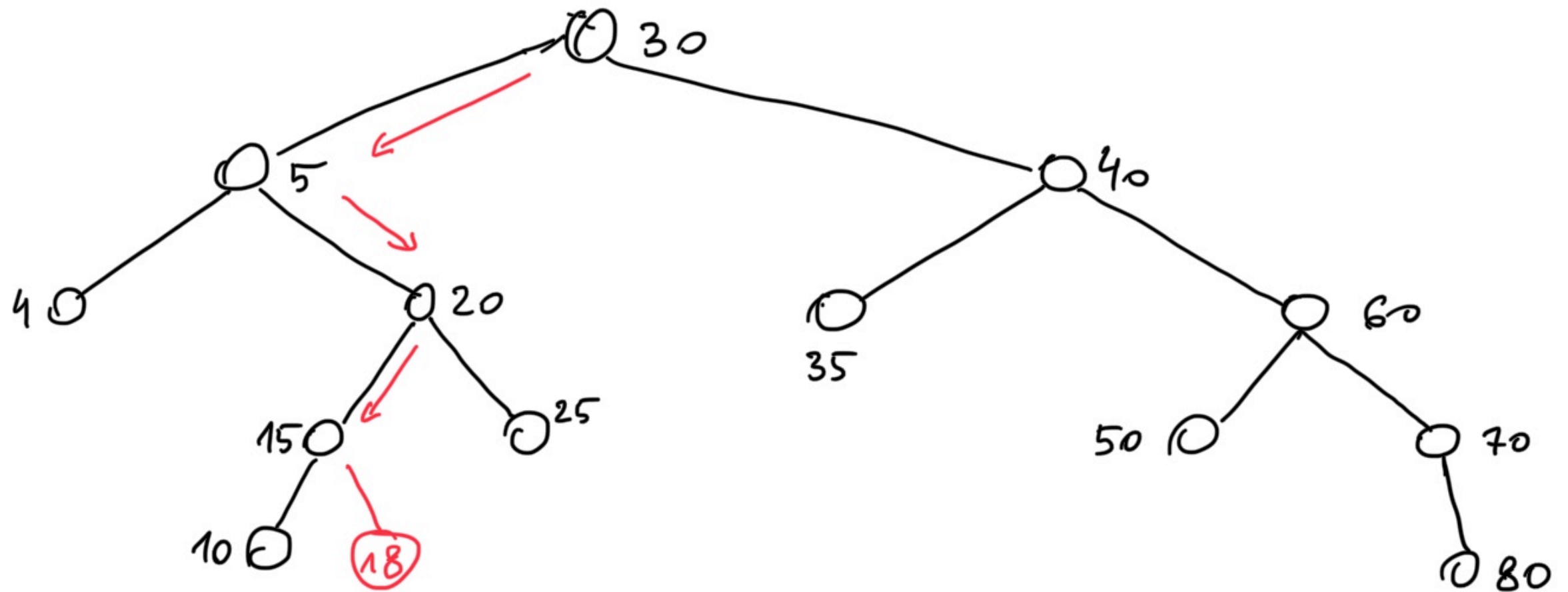
arbre binaire de recherche



pour chaque sommet s :

valeurs (sous-arbre gauche de s) $<$ valeur(s) $<$
valeurs (sous-arbre droit de s)

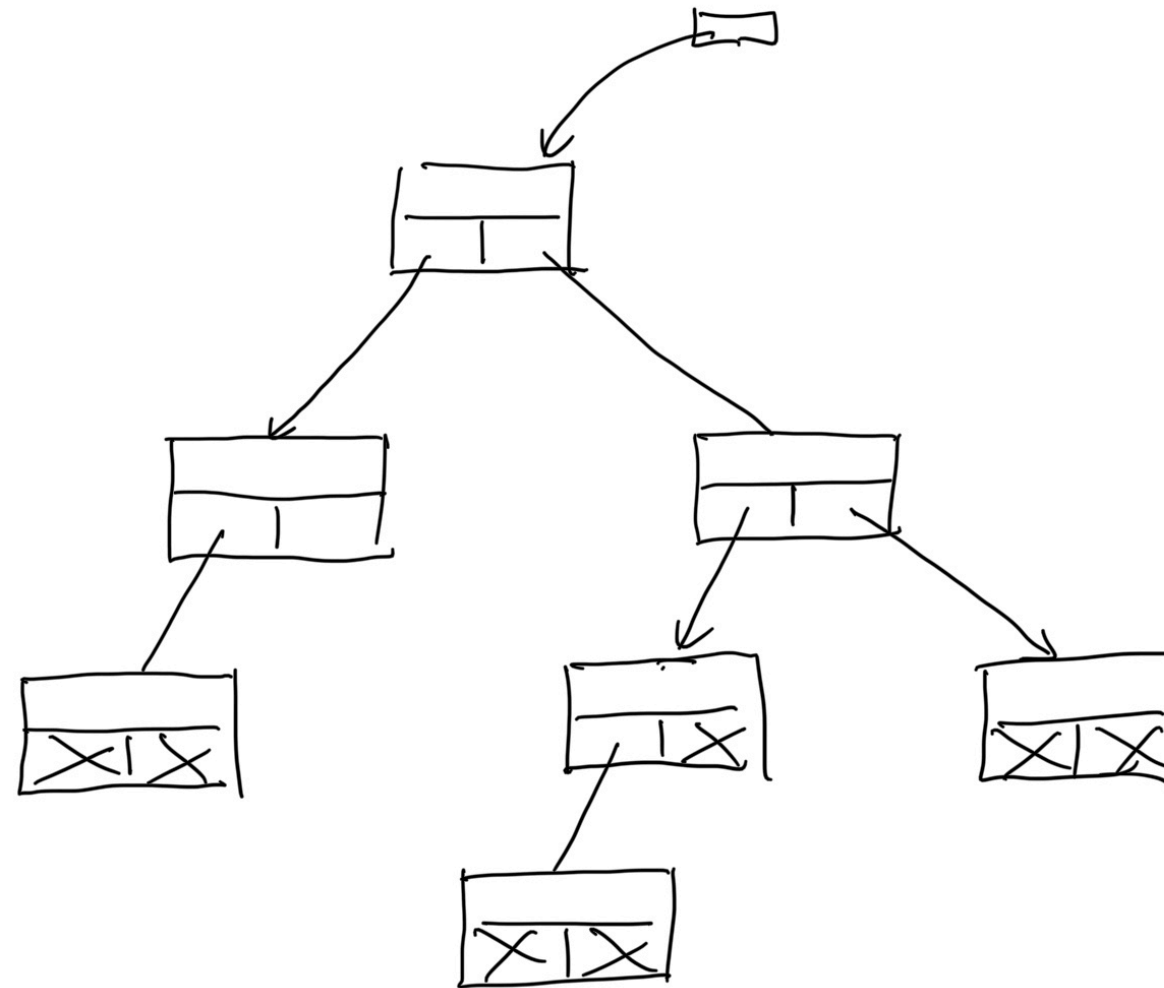
insérer dans un arbre binaire de recherche




inserez 18

arbre binaires en C

Les arbres binaires en C



 - pointeur NULL



Définir un arbre en C

```
#define LEN 32

struct noeud{

    char info[LEN];    /* information stockée dans le noeud */

    struct noeud *gauche; /* l'adresse de l'enfant gauche */

    struct noeud *droit;  /* l'adresse de l'enfant droit */

};

typedef struct noeud noeud;

typedef noeud *arbre;
```



```
arbre arbre_ajouter_feuille(arbre a, const char *s)
```

```
arbre arbre_ajouter_feuille( arbre a,  
                             const char *s){  
  
    arbre new = malloc( sizeof( noeud ) );  
    assert( new );  
    new->gauche = new->droit = NULL;  
  
    /* copier le string pointé par s dans info*/  
    strncpy( new->info, s, LEN-1);  
    new->info[LEN-1] = '\\0';  
    if( a == NULL )  
        return new;
```

```
arbre arbreajouter_feuille(arbre a, const char *s)
```

```
    arbre pere = NULL;
    arbre courant = a;
    int direction;

    while( courant != NULL ){

        direction = random() & 1 ; /* 0 ou 1 aléatoirement en prenant le dernier bit
                                     * de la valeur retournée par random() */

        pere = courant;
        if( direction == 0 )
            courant = courant -> gauche ;
        else
            courant = courant -> droit;
    }/* fin while */

    if( direction == 0 )
        pere->gauche = new;
    else
        pere->droit = new;
    return a;
}
```

```
int arbre_nb_feuilles( arbre a )
```

```
int arbre_nb_feuilles( arbre a ){  
    if( a == NULL )  
        return 0;  
  
    if( a->gauche == NULL && a->droit == NULL )  
        return 1;  
  
    return arbre_nb_feuilles( a->gauche ) +  
           arbre_nb_feuilles( a->droit );  
}
```

```
void arbre_afficher( arbre a )
```

```
void arbre_afficher( arbre a ){  
    if( a == NULL )  
        return;  
    printf("%s ", a->info);  
    arbre_afficher( a->gauche );  
    arbre_afficher( a->droit );  
}
```

tester

```
int main( void ){  
    /* initialiser le générateur de nombre aléatoires */  
    srand( time(NULL) );  
  
    char *mots[]={ "chien", "chat", "cheval",  
"vache", "singe", "pinguin","requin", "tortue",  
"bison", "giraffe", "papillion", "sangsue",  
"renard"};  
  
    size_t nmots = sizeof( mots )/ sizeof( mots[0] );  
  
    arbre a = NULL;  
    for( size_t i = 0 ; i < nmots; i++)  
        a = arbre_ajouter_feuille( a, mots[i] );  
}
```

tester

```
int i = arbre_nb_feuilles( a );  
printf("nb feuilles %d\n", i);
```

```
arbre_afficher( a );  
printf("\n");
```

```
}
```