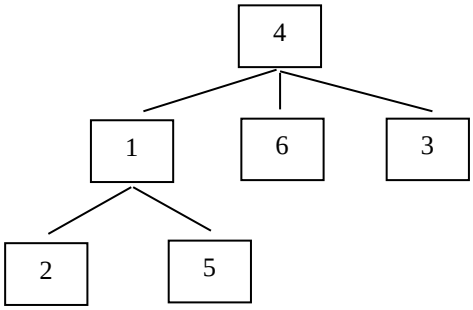
Travaux Dirigés
Séance nr. 6

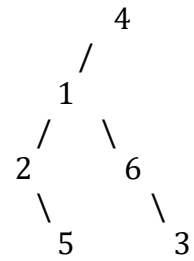
Implantation d'une forêt générale par liste chaînée de fils

On souhaite représenter des arbres quelconques : chaque noeud peut avoir *n* fils, *n* n'étant pas borné. Pour cela on utilise la binarisation d'un arbre tel que vu en cours : concrètement, en chaque nœud on stocke un pointeur vers le premier fils et un pointeur vers le frère (soit deux pointeurs seulement, comme pour un arbre binaire !).

1. Faire une représentation « binarisée » de l'arbre suivant :



Corrigé :



Dans cet arbre binaire, on prend comme fils gauche, le premier fils et comme fils droit, le frère.

2. On se propose d'utiliser la structure de données suivante:

```
typedef struct s_noeud {
        T       val;
        struct s_noeud    *fils, *frere;
} noeud, *arbre;
```

Proposer une implantation des opérations de base :

```
arbre       arbrevide() ;
arbre       nouvnoeud(T x, arbre frere, arbre fils); // création
arbre       fils(arbre a); // récupérer le premier fils
```

```
arbre        frere(arbre a) ; // récupérer le frère
bool         est_vide(arbre a)
bool         est_feuille(arbre a) ;
bool         est_fils(T x, arbre a) ; // teste si x est un des fils
                                      // de a (fils, pas descendant)
unsigned int nb_fils(arbre a);  // nombre de fils de a,
                                //dans l'exemple « 4 » a trois fils
```

Corrigé

```
arbre arbrevide() { return NULL; }
arbre nouvnoeud(T x, arbre frere, arbre fils)
{
        arbre anouv;

        anouv = (arbre)malloc(sizeof(noeud));
        anouv->fils = fils;
        anouv->frere = frere;
        anouv->val = x;
        return anouv;
}
arbre fils(arbre a) { return a->fils; }
arbre frere(arbre a) { return a->frere;  }
bool est_vide(arbre a) { return a == NULL;  }
bool est_feuille(arbre a) { return a->fils == NULL;  }
bool est_fils(T x, arbre a)
{
        if (a == NULL) return false;
        arbre f = a->fils;
        while (f != NULL)
        {
                if (f->val == x) return true;
                f = f->frere;
        }
        return false;
}
unsigned int nb_fils(arbre a)
{
        if (a == NULL) return 0;
        arbre f = a->fils;
        unsigned int n = 0;
        while (f != NULL)
        {
                n++;
                f = f->frere;
        }
        return n;
}
```

En utilisant ces opérations faire également un « main » qui construit l'arbre de la question 1.

Corrigé

```
int main()
{
        arbre a = nouvnoeud(4,
                        arbrevide(),
                        nouvnoeud(1,
```

```
nouvnoeud(6,
        nouvnoeud(3,
                arbrevide(),
                arbrevide()),
        arbrevide()),
nouvnoeud(2,
        nouvnoeud(5,
                arbrevide(),
                arbrevide()),
        arbrevide())));
```

3. Ajouter maintenant des opérations pour :

a. calculer la hauteur de l'arbre (attention ce n'est pas la hauteur de l'arbre binaire). La hauteur de l'arbre de l'exemple est 3.
b. tester l'existence d'une étiquette. Par exemple 5 existe dans l'arbre précédent : on renvoie alors le pointeur vers le nœud qui contient 5 (un pointeur nul si le nœud n'existe pas).
c. écrire un algorithme permettant d'afficher l'arbre de la façon suivante :

```
|4
   |1
     |2
     |5                ...
   |6
   |3
```

A chaque niveau de profondeur on décale l'affichage vers la droite. Pour chaque nœud on affiche ses fils les uns en dessous des autres. On fait également précéder d'un caractère pipe chaque affichage.

Corrigé

```
int hauteur(arbre a)
{
        if (a == NULL) return -1;
        int h = hauteur(a->fils)+1;
        arbre f = a->frere;
        while (f != NULL)
        {
                int ha = hauteur(f);
                if (ha > h) h = ha;
                f = f->frere;
        }
        return h;
}

arbre existenoeud(T x, arbre a)
{
        if (a == NULL) return NULL;
        if (a->val == x) return a;
        arbre r = existenoeud(x, a->fils);
        if (r != NULL) return r;
        r = existenoeud(x, a->frere);
        return r;
}
void affiche(arbre a, unsigned int n)
{
```

```
        if (a == NULL) return;
        for (unsigned int i = 0; i < n; i++) printf("  ");
        printf("%d\n", a->val);
        affiche(a->fils, n + 1);
        affiche(a->frere, n);
}
```