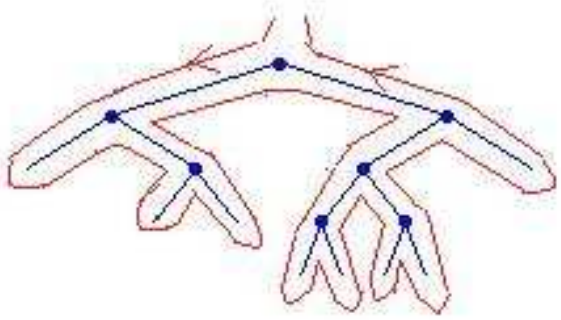
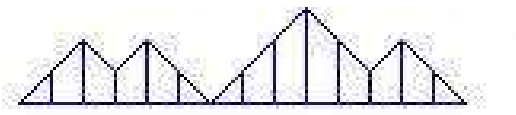
Chaînes de montagnes et arbres binaires

Comme nous l'avons vu, il existe autant de chaînes de montagnes de longueur 2*N* que d'arbres binaires avec *N* nœuds, et leur nombre est *c*(*N*), nombre de Catalan. Nous allons maintenant construire la bijection qui permet de passer de ces chaînes de montagnes aux arbres, et inversement.

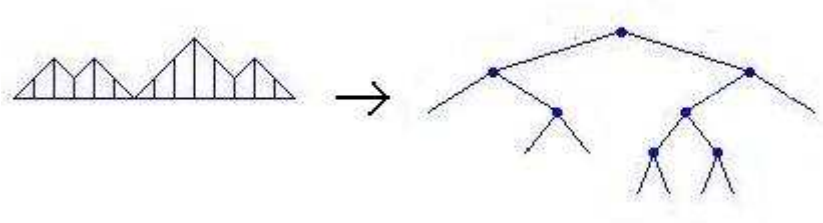
Pour faciliter les choses, un arbre binaire sera dessiné sous sa forme remplie : chaque nœud a deux branches pendantes, une à gauche et une à droite, auxquelles viennent s'accrocher ou pas un nœud. On peut alors en faire une lecture préfixée. En dessinant le chemin qui permet de parcourir l'arbre à partir de sa racine en y revenant finalement, ce chemin longe les branches de l'arbre en descente, ainsi que les nœuds, puis passe en dessous de chaque nœud, et longe à nouveau les branches en remontée. La lecture préfixée est celle qui longe les branches en descente, correspondant à l'ordre dans lequel les nœuds sont pour la première fois atteints. Par exemple pour *N* = 7 :



Remarquons que l'arbre possède *N* branches gauches et *N* branches droites. Il suffit alors, dans le parcours préfixé, de noter une branche gauche parcourue en descente avec un 0, et une branche droite parcourue en descente avec un 1, ce qui donne dans l'exemple choisi 00101100011011, qui est une chaîne de montagnes, ou le mot de parenthèses équivalent (() ()) ((()) ()) dessinée ainsi, avec 0 ou (représenté par un pas montant à 45°, et 1 ou) représenté par un pas descendant à 45° :



Inversement, partons d'une chaîne de montagnes et construisons l'arbre binaire correspondant. Chaque fois que l'on tombe sur un 0, on obtient un nœud de l'arbre et sa branche de gauche, et quant on tombe sur 1, on dessine une branche à droite à partir de là où on est, et en remontant éventuellement de proche en proche jusqu'aux nœuds où elle manque.



Passons à la programmation. Comme nous savons construire les chaînes de montagnes de longueur 2N, nous allons partir d'elles pour en déduire les arbres binaires correspondants. Chaque chaîne est enregistrée dans un tableau *m*[]. A partir de là la fonction *arbre*() est appelée. Elle commence par recopier le tableau *m*[] dans un tableau *mm*[], pour des raisons qui vont s'expliquer. Puis on parcourt le tableau *m*[]. Chaque fois qu'on tombe sur un 0, correspondant à un nœud de l'arbre, on le numérote dans l'ordre croissant à partir de 0 et on le place dans un tableau *c*[] dans la même position qu'il avait dans le tableau *m*[]. Dans l'exemple choisi, cela fait :

m[] : 00101100011011
c[] : 01- 2- - 345- -6- -

Puis on parcourt le tableau *mm*[], qui est au départ le même que *m*[], à la recherche des 1 (parenthèse fermante) de façon à lui donner le numéro du nœud correspondant (du 0 ou de la parenthèse ouvrante correspondante. Si le 1 est précédé de 0, il s'agit de deux parenthèses qui se correspondent, et dans *c*[] on place le numéro du nœud. Puis on élague le tableau *mm*[] en mettant -1 aux places des deux parenthèses concernées. Plus généralement, à partir de chaque 1, on circule à gauche tant qu'on rencontre des -1, et quand on tombe sur le premier 0, on donne le numéro du nœud correspondant dans le tableau *c*[] à la position où se trouve le 1 dans le tableau mm[], et l'on met des -1 aux positions des parenthèses ouvrante et fermante dans *mm*[]. Voici ce que cela donne progressivement :

mm[] : 0 0 **1** 0 1 1 0 0 0 1 1 0 1 1
c[] : 0 1 1 2 3 4 5 6

mm[] : 0-1-1 0 **1** 1 0 0 0 1 1 0 1 1
c[] : 0 1 1 2 2 3 4 5 6

mm[] : 0-1-1-1-1 **1** 0 0 0 1 1 0 1 1
c[] : 0 1 1 2 2 0 3 4 5 6

mm[] : -1-1-1-1-1-1 0 0 0 **1** 1 0 1 1
c[] : 0 1 1 2 2 0 3 4 5 5 6

mm[] : -1-1-1-1-1-1 0 0 -1 -1 **1** 0 1 1
c[] : 0 1 1 2 2 0 3 4 5 5 4 6

mm[] : -1-1-1-1-1-1 0 -1 -1 -1 -1 0 **1** 1
c[] : 0 1 1 2 2 0 3 4 5 5 4 6 6

mm[] : -1-1-1-1-1-1 0 -1 -1 -1 -1 -1 -1 **1**
c[] : 0 1 1 2 2 0 3 4 5 5 4 6 6 3

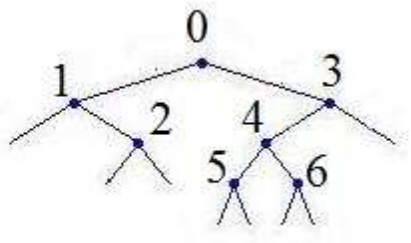
mm[] : -1-1-1-1-1-1 -1 -1 -1 -1 -1 -1 -1 -1

Finalement, à partir du tableau *m*[] et du tableau *mm*[] qui à partir de sa copie de *m*[] se modifie peu à peu, on a le tableau *c*[] qui donne chaque nœud numéroté avec sa branche gauche et sa branche droite :

m[] : **(** **(**) **(**)) **(** **(** **(**)) **(**))
c[] : 0 1 1 2 2 0 3 4 5 5 4 6 6 3

Le nœud 0 étant la racine, on parcourt à nouveau le tableau *m*[] à partir de la case d'indice 1, à la recherche de chaque 0 (parenthèse ouvrante correspondant à un nœud). Si celle-ci, en position *i*, est précédée d'une parenthèse ouvrante - un nœud dont le numéro se trouve dans *c*[] en position *i* - 1 cela signifie que le fils gauche de *c*[*i* - 1] est *c*[*i*], et sinon cela signifie que le fils droit de *c*[*i* – 1] est *c*[*i*].

D'où le résultat :
fils gauche de 0 = 1
fils droit de 1 = 2
fils droit de 0 = 3
fils gauche de 3 = 4
fils gauche de 4 = 5
fils droit de 4 = 6



Dans le programme, le fils gauche d'un nœud i est noté *f*[0][*i*] et son fils droite est *f*[1][*i*]. Ces deux tableaux sont mis au départ à -1, de façon qu'il reste -1 là où aucun nœud n'est accroché.

noeud	fils gauche	fils droit
0	1	3
1	-1	2
2	-1	-1
3	4	-1
4	5	6
5	-1	-1
6	-1	-1

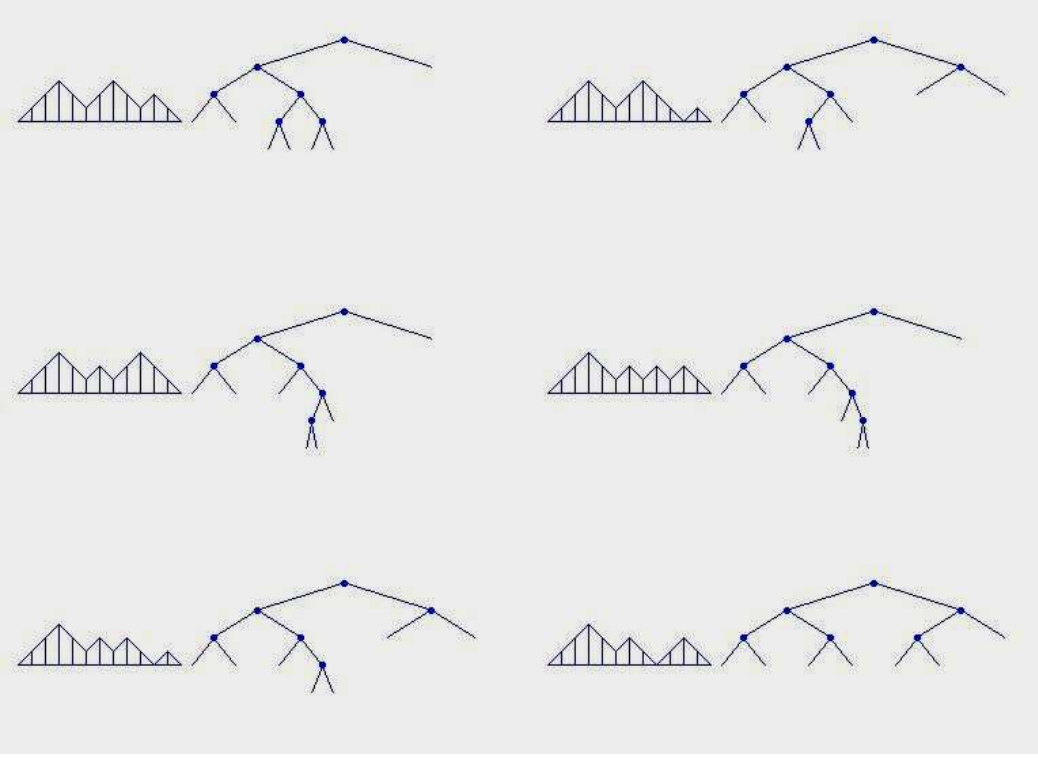
Il reste à dessiner l'arbre. On a besoin de connaître les coordonnées de chaque nœud de l'arbre. Pour cela on détermine d'abord l'étage où se trouve le nœud, la racine étant à l'étage 0. Cela permet de connaître l'ordonnée de chaque nœud. Pour les abscisses, on ajoute ou on retranche, selon qu'on est à gauche ou à droite, une puissance décroissante de 2 par rapport au nœud prédécesseur.

On en déduit le programme complet :

```
#include <SDL/SDL.h>
#include <math.h>
#include <stdlib.h>
#define N 6    /* on peut aller jusqu'à N=7 sans changer les échelles */
#define pas 10
void arbre(void);
void chainemontagne(void);
void pause(void);
void putpixel(int xe, int ye, Uint32 couleur);
Uint32 getpixel(int xe, int ye) ;
```

```
void ligne(int x0,int y0, int x1,int y1, Uint32 cc);
void cercle( int xo, int yo, int RR, Uint32 couleur);
void disque( int xo, int yo, int RR, Uint32 couleur);

SDL_Surface * ecran;    Uint32 rouge1,rouge2,blanc,bleu, jaune,noir, vert1,vert2;
int m[14]; int xorig =20, yorig =100;

int main(int argc, char ** argv)
{ int i,j,k,pospivot,compteur0;

  SDL_Init(SDL_INIT_VIDEO);
  ecran=SDL_SetVideoMode(800,600,32, SDL_HWSURFACE|SDL_DOUBLEBUF);
  blanc=SDL_MapRGB(ecran->format,255,255,255);
  bleu=SDL_MapRGB(ecran->format,0,0,255);
  SDL_FillRect(ecran,0,blanc);

  for(i=0;i<2*N;i++)
  if (i<N) m[i]=0; else m[i]=1;

  for(;;)
    { chainemontagne();
      arbre();
      i=2*N-1; compteur0=0;
      while( i>=2 && !(m[i]==1 && m[i-1]==1 && m[i-2]==0))
        { if (m[i]==0) compteur0++;
          i--;
        }
      pospivot=i-2; if (pospivot==-1) break;
      m[pospivot]=1; m[pospivot+1]=0;

      k=0;
      for(j=pospivot+2; j<2*N; j++)
        { if (k<compteur0) m[j]=0; else m[j]=1; k++;
        }
    }
  pause();return 0;
}

void chainemontagne(void)
{ int x,y,oldx,oldy,i;
   x=xorig; y=yorig; ligne(x,y,x+2*N*pas,y,bleu);
   for(i=0;i<2*N;i++)
     { oldx=x;oldy=y; x+=pas;
       if(m[i]==0) y-=pas; else y+=pas;
       ligne(oldx,oldy,x,y,bleu);
       ligne( x,y,x,yorig,bleu);
     }
   SDL_Flip(ecran);
}

void arbre(void)
{ int i,k,j;int etage[N],x[N],y[N],fg,fd,pasyy=20;
   int mm[2*N], c[2*N], f[2][N];
  for(i=0;i<2*N;i++) mm[i]=m[i];
  k=0;    for(i=0;i<2*N;i++) if (m[i]==0)  c[i]=k++;
```

```
 for(i=1;i<2*N;i++)  if (mm[i]==1)
  { j=i-1;   while(mm[j]==-1) j--;
    c[i]=c[j];mm[i]= -1;mm[j]= -1;
  }
 for(i=0;i<2;i++) for(j=0;j<N;j++) f[i][j]= -1;
 for(i=1;i<2*N;i++) if (m[i]==0)
 if (m[i-1]==0)  f[0][c[i-1]]=c[i];   else f[1][c[i-1]]=c[i];
      /* dessin */
 etage[0]=0; x[0]=xorig+2*N*pas+120; y[0]=yorig-60;
 for(i=0;i<N;i++)   /* placement des noeuds */
  { if(f[0][i]!= -1)
                  { fg=f[0][i]; etage[fg]=etage[i]+1;
                    x[fg]=x[i]-128/pow(2,etage[fg]); y[fg]=y[i]+pasyy;
                  }
    if (f[1][i]!=-1)
       { fd=f[1][i]; etage[fd]=etage[i]+1;
                    x[fd]=x[i]+128/pow(2,etage[fd]);  y[fd]=y[i]+pasyy;
                  }
   }
 for(i=0;i<N;i++)    /* dessin des branches gauche et droite, ainsi que des noeuds */
   { ligne(x[i],y[i],x[i]-128/pow(2, etage[i]+1),y[i]+pasyy,bleu);
     ligne(x[i],y[i],x[i]+128/pow(2, etage[i]+1),y[i]+pasyy,bleu);
    disque(x[i],y[i],2,bleu);
   }
  SDL_Flip(ecran);
 xorig+=390;
 if (xorig>750)  {xorig=20; yorig+=200;}
 if (yorig>600) {pause();SDL_FillRect(ecran,0,blanc); xorig=20; yorig=100;}
}
```



Pour $N = 6$, avec 132 chaînes de montagnes, une des 22 pages où sont dessinés les chaînes et arbres correspondants, chaque page contenant 6 chaînes.