Compléter les fonctions récursives dans le programme suivant.

```
#include<stdio.h>
int pcgd(int a, int b) { return b ? pgcd(...,...) : ... ; }
// pgcd(70,49)=pgcd(49,70%49)=pgcd(49,21)=pgcd(21,49%21)=pgcd(21,7)=pgcd(7,0)=7
int fact(int n) { return ... ? n*fact(...) : ... ; }
/* fact(4)=4*fact(3)
     fact(3)=3*fact(2)
       fact(2)=2*fact(1)
         fact(1)=1
       fact(2)=2*1=2
     fact(3)=3*2=6
   fact(4)=4*6=24  */
int F(int n) { return n<=1 ? ... : F(...)+F(...) ; }
// F(5)=F(4)+F(3)  F(1)=1  F(0)=0
int F2(int m, int Fnm, int Fnm1) // retourne F(n) à partir de Fnm=F(n-m) et Fnm1=F(n-m-1)
{ return n ? F2(n-1, ...+..., ...) : ... ; }
int F3(int n) { return F2(n, ... , ... ); }
// F(1)=F(0)+F(-1) donne 1=0+F(-1)
// F3(6)=F2(6,0,1)=F2(5,1,0)=F2(4,1,1)=F2(3,2,1)=F2(2,3,2)=F2(1,5,3)=F2(0,8,5)=8
int C1(int n, int p) { return fact(n)/fact(p)/fact(n-p); }
int C2(int n, int p) { return ... ? 1 : C2(n-1,p)+C2(...,...); }
int C3(int n, int p) { return ... ? 1 : C3(n,p-1)*.../...; }
int C4(int n, int p) { return ... ? C3(n,p) : C3(n,n-p) ; }
void P1(int n) { if(n) P1(...), printf(...); }  // P1(6) écrit 1 2 3 4 5 6
void P2(int n) { if(n) ...    , ...        ; }  // P2(6) écrit 6 5 4 3 2 1
void P3(int n) { if(n) ... , ... , ...     ; }  // P3(6) écrit 6 5 4 3 2 1 1 2 3 4 5 6
void D1(int n) { if(n) D1(...), printf(...); } /* D1(3) écrit
 *
  *
   *    */
void D2(int n) { if(n) ...    , ...        ; } /* D2(3) écrit
   *
  *
 *      */
void D3(int n) { if(n) ... , ... , ...     ; } /* D3(3) écrit
   *
  *
 *
 *
  *
   *
On pourra transformer Pi en Di en remplaçant printf("%d ",n) par printf("%*s*\n",n,"") */
#define cas(n,nom) case n:printf(#nom" "); break
void libelle(unsigned n)  /* libelle(321456789) écrit trois cent vingt un million quatre
cent cinquante six mille sept cent octante neuf.  On oublie donc les s à la fin de vingt,
cent, million et milliard. On oublie aussi les "et". */
{ if(n) switch(n)
  { cas(1,un); cas(2,deux); ...  cas(16,seize);
    cas(20,vingt); cas(30,trente); ...cas(70,septante); cas(80,octante); cas(90,nonante);
    default:
    if(n>=1000000000) libelle(n/1000000000), printf("milliard "), libelle(n%1000000000);
```

```
    else if(n>=1000000) ...
    else if(n>=2000) ...
    else if(n>=1000) ...
    else if(n>=200) ...
    else if(n>=100) ...
    else if(n>=20) ...
    else libelle(10), libelle(...);
  }
}
void abracadabra(char c) { if(c>='a') abracadabra(c-1),putchar(c),abracadabra(c-1); }
void hanoi(int n,int a, int b)
{ if(n) hanoi(n-1,...,6-a-b),
        printf("La rondelle %d va de la tige %d à la tige %d\n",n,a,b),
        hanoi(n-1,...,...);
}
void testpgcd(int a, int b) { printf("pgcd(%d,%d)=%d\n",a,b,pgcd(a,b)); }
void testfact(int n) ...
void testbinomial(int n, int p) { printf("...", n<20?C1(n,p):0, C2(n,p), ..., C4(n,p)); }
void testfibo(n) { int a=n>45?0:F(n), b=F3(n);   printf("F%d=%d=%d  %d\n",n,a,b,a-b); }
void fusion(int *u, int m, int *v, int p, int *w)
{ if(...) *w=*u, fusion(u+1,m-1,v,p,w+1); else
  if(...) *w=*v, fusion(...            );
}
void copie(int *t, int n, int *u) { if(n) *u=*t, copie(...); }
void trifusion(int *t, int n)
{ int m=n/2, p=n-m,  u[m], *v=t+m;
  // u est une copie de la première moitié du tableau t.
  // v est la deuxième moitié du tableau t.
  if(m) copie(t,m,u), trifusion(u,m), trifusion(v,p), fusion(u,...,v,...,t);
}
void aff(int t[],n) { if(n) printf(...,*t), aff(t+1,...); else printf("\n"); }
void testtribulle(int n)
{ int t[n], i, z=1;
  for(i=0;i<n;i++) t[i]=z=z*641%(n*n);
  aff(t,n);
  tribulle(t,n);
  aff(t,n);
}
int main()
{ testpgcd(70,49); testpgcd(1001,343);
  testfact(6);     testfact(10);
  testfibo(6);     testfibo(20);      testfibo(40);      testfibo(50);
  testbinomial(5,2); testbinomial(15,6); testbinomial(30,10); testbinomial(30,20);
  P1(6), P2(7), P3(4);
  D1(6), D2(7), D3(4);
  for(int n=1;n>=0;n+=n/7+1) printf("%d ",n), libelle(n), printf("\n");
  abracadabra('d');
  hanoi(4,1,2);
  testtribulle(6), testtribulle(100), testtribulle(1000);
  return 0;
}
```