```
#include <iostream.h>

int fact(int n){
  if (n == 0)
    return 1;
  else
    return n * fact(n-1);
}

int main(){
  int n;

  while (true){
    cout << "Calculer fact de : " << flush;
    cin >> n;
    cout << "Réponse : " << fact(n) << endl << endl;
  }
}
```

```
#include <iostream.h>

int fibo(int n){
  return n <= 1 ? n : fibo(n-2) + fibo(n-1);
}

int main(){
  int fin;
  cout << "Afficher la suite jusqu'à : " << flush;
  cin >> fin;

  for(int i = 0;i <= fin;i++) {
    cout << "fibo(" << i << ")=" << flush;
    cout << fibo(i) << endl;
  }
}
```

```
#include <iostream.h>

// Imprime les mouvements a effectuer pour
// deplacer n etages du piquet A au piquet B
// en passant par le piquet C
void hanoi(int n,int piquetA,int piquetB,int piquetC){
  if (n == 0)
    return;
  else {
    hanoi(n-1,piquetA,piquetC,piquetB);
    cout << piquetA << " -> " << piquetB << endl;
    hanoi(n-1,piquetC,piquetB,piquetA);
  }
}

void hanoi(int n){
  cout << "Solution :" << endl;
  hanoi(n,1,2,3);
}

int main(){
  int nombre;
  cout << "Nombre d'étages sur le premier piquet : " << flush;
  cin >> nombre;
  hanoi(nombre);
}
```

```
//###########################################################################

#include <iostream.h>

//###########################################################################

class Tour {
public:
  int hauteur;
  int* largeurs;
  Tour(int nombre_etages) : hauteur(0), largeurs(new int[nombre_etages]) {}
  void ajoute(int largeur) { largeurs[hauteur++] = largeur; };
  int retire() {return largeurs[--hauteur]; };
};

//###########################################################################

// Le nombre d'étages.
int nombre_etages;
// Les trois tours.
Tour* tours[3];

//###########################################################################

// Affiche un etage d'une tour.
void affiche(int etage, Tour* tour) {
  int largeur = (etage < tour->hauteur) ? tour->largeurs[etage] : 0;
  cout<<" ";
  for (int i = nombre_etages - largeur; i > 0; i--) cout<<" ";
  for (int i = largeur; i > 0; i--) cout<<"#";
  cout<<"|";
  for (int i = largeur; i > 0; i--) cout<<"#";
  for (int i = nombre_etages - largeur; i > 0; i--) cout<<" ";
  cout<<" ";
}

// Affiche les trois tours.
void affiche() {
  for (int etage = nombre_etages; etage >= 0; etage--) {
    affiche(etage, tours[0]);
    affiche(etage, tours[1]);
    affiche(etage, tours[2]);
    cout << endl;
  }
  cout << endl;
}

// Déplace un élément de la tour src à la tour dst.
void deplace1(Tour* src, Tour* dst) {
  dst->ajoute(src->retire());
  affiche();
}

// Déplace n éléments de la tour src à la tour dst en utilisant, si
// nécessaire, la tour tmp.
void deplaceN(int n, Tour* src, Tour* dst, Tour* tmp) {
  if (n == 0) return;
  deplaceN(n-1, src, tmp, dst);
  deplace1(src, dst);
  deplaceN(n-1, tmp, dst, src);
}
```



```
// Résoud le problème des tours de Hanoi avec n étages.
void hanoi(int n) {
  nombre_etages = n;
  tours[0] = new Tour(nombre_etages);
  tours[1] = new Tour(nombre_etages);
  tours[2] = new Tour(nombre_etages);
  for (int i = nombre_etages; i > 0; i--) tours[0]->ajoute(i);
  affiche();
  deplaceN(nombre_etages, tours[0], tours[2], tours[1]);
}

//###########################################################################

int main(){
  int n;
  cout << "Nombre d'étages sur le premier piquet : " << flush;
  cin >> n;
  hanoi(n);
}

//###########################################################################
```

```
// tri rapide

#include <stdlib.h>

class Elem{
public:
  int valeur;
  Elem* suivant;
  Elem(int valeur,Elem* suivant);
};

Elem::Elem(int valeur,Elem* suivant){
  this->valeur = valeur;
  this->suivant = suivant;
}

//--------------------------------------------
//           ALGORITHME DE TRI
//--------------------------------------------


void insererEnTete(Elem* e,Elem* &liste){
  ///-  A compléter.
  e->suivant = liste;
  liste = e;
  ///+
}

// Attache la liste "liste2" a la fin de la liste "liste1".
void concatener(Elem* &liste1,Elem* liste2){
  ///- A compléter (algorithme récursif).
  if (liste1 == NULL)
    liste1 = liste2;
  else
    concatener(liste1->suivant,liste2);
  ///+
}

// rajoute à la liste "petits" (resp. "grands") les éléments
// de la liste "liste" plus petits (resp. grands) que le "pivot".
void separer(int pivot,Elem* liste,Elem* &petits,Elem* &grands){
  ///- A compléter (algorithme récursif ou itératif).
  if (liste == NULL)
    return;
  else {
    Elem* apres = liste->suivant;
    if (liste->valeur < pivot)
      insererEnTete(liste,petits);
    else
      insererEnTete(liste,grands);
    separer(pivot,apres,petits,grands);
  }
  ///+
}

// Attache l'un derrière l'autre la liste "petits",
// l'élément "pivot" et la liste "grands", et retourne
// la liste "petits".
Elem* joindre(Elem* &petits,Elem* pivot,Elem* grands){
  pivot->suivant = grands;
  concatener(petits,pivot);
  return petits;
```



```
}

// Trie la liste "liste" sur place.
void trier(Elem* &liste){
  ///- A compléter (algorithme récursif).
  if (liste == NULL)
    return;
  else if (liste->suivant == NULL)
    return;
  else {
    Elem* petits = NULL;
    Elem* grands = NULL;
    Elem* pivot = liste;
    separer(pivot->valeur,liste->suivant,petits,grands);
    trier(petits);
    trier(grands);
    liste = joindre(petits,pivot,grands);
  }
  ///+
}

//--------------------------------------------
//         TEST DE L'ALGORITHME
//--------------------------------------------

#include <iostream.h>
#include <math.h>

void afficher(Elem* liste){
  Elem* courant = liste;
  while (courant != NULL){
    cout << courant->valeur << " ";
    courant = courant->suivant;
  }
  cout << endl;
}

// renvoie aléatoirement un nombre entre 0 et n.
int hasard(int n){
  return rand() % n;
}

int main(){
  Elem* liste = NULL;
  int longueur;

  cout << "Longueur de la liste à trier : " << flush;
  cin >> longueur;

  // Génération aléatoire de la liste de nombres.
  for(int i = 0;i < longueur;i++)
    insererEnTete(new Elem(hasard(longueur),NULL),liste);

  cout << "Liste générée : " << flush; afficher(liste);

  trier(liste);
  cout << "Liste triée  : " << flush; afficher(liste);
}
```