```
// Pile.h
#ifndef _PILE_H
#define _PILE_H

template <class T>
class Pile {
public:
  Pile(); // constructeur
  Pile(const Pile<T> & p); // constr. de copie
  ~Pile(); // destructeur

  // les primitives classiques :
  T valeurSommet() const;
  void empiler(const T& elem);
  void depiler();
  bool pileVide() const;

private:
  static const int TAILLE_BLOC; //unité d'allocation
  int m_taille_pile;
  T * m_corps; // le tableau sera alloué dynamiquement
  int m_sommet;
  void agrandir(); // sera utilisé par empiler s'il le faut
};

#include "Pile.cxx"

#endif
```



```
// Pile.cxx : EXTRAITS
// seulement ce qui change par rapport à la version allocation statique

template <class T>
const int Pile<T>::TAILLE_BLOC = 10; //par exemple

template <class T>
Pile<T>::Pile()
{
  m_taille_pile = TAILLE_BLOC; // taille par défaut : 1 bloc
  m_corps = new T[m_taille_pile];
  m_sommet = -1;
}

template <class T>
Pile<T>::Pile<T>(const Pile<T> &p)
{
  m_corps = new T[p.m_taille_pile]; // chaque pile a sa taille
  m_sommet = p.m_sommet;
  for(int i = 0; i <= m_sommet; i++)
    m_corps[i] = p.m_corps[i];
}

template <class T>
Pile<T>::~Pile<T>()
{
  delete[] m_corps;
}

template <class T>
void
Pile<T>::empiler(const T& elem)
{
  if(m_sommet == m_taille_pile-1) // pile pleine
    agrandir();
//empilement proprement dit.
  m_sommet++;
  m_corps[m_sommet] = elem;
}

template <class T>
void
Pile<T>::agrandir()
{
  m_taille_pile += TAILLE_BLOC ;
//allocation nouvelle zone
  T* tmp = new T[m_taille_pile];
//recopie de la pile dans cette zone
  for(int i=0; i<= m_sommet; i++)
    tmp[i] = m_corps[i];
//libération ancienne zone
  delete[] m_corps;
// nouvelle zone devient le corps de la pile
  m_corps = tmp;
}
```