```
# Ce programme est une version MPD d'algorithmes avec backtracking et
# branch−and−bound pour le probleme du sac a dos 0−1.

  ##########################################################################

  const int NB_ITEMS = ...

  # Types auxiliaires divers (pour rendre plus claires les signatures).
  type PoidsItem = int;
  type Benefice  = int;
  type Item      = int;

  # Types pour les donnees du probleme.
  type Poids     = [NB_ITEMS]PoidsItem;
  type Benefices = [NB_ITEMS]Benefice;

  # Types pour la solution du probleme.
  # Pour chaque item, indique si l'item est selectionne ou non.
  type Selection = [NB_ITEMS]bool;

  ##########################################################################
  # Procedures auxiliaires.
  ##########################################################################

  # Benefice d'une serie d'items selectionnes (une selection).
  procedure benefice( Benefices benefs, Selection sel ) returns Benefice b
  {
    b = 0;
    for [i = 1 to NB_ITEMS st sel[i]] {
      b += benefs[i];
    }
  }

  # Poids total d'une selection.
  procedure poidsTotal( Poids poids, Selection sel ) returns PoidsItem p
  {
    p = 0;
    for [i = 1 to NB_ITEMS st sel[i]] {
      p += poids[i];
    }
  }

  # Mise a jour, si inferieure, de la selection si optimale (max).
  procedure mettreAJourMax( Selection sel, Benefice b,
                            ref Selection selMax, ref Benefice bMax )
  {
     if ( b > bMax ) {
       selMax = sel;
       bMax = b;
     }
  }
```

```
#############################################################################
# Exploration de type "Marche arriere" (backtracking) des selections.
#############################################################################

procedure remplirSacMARec( Benefices benefs,
                           Poids poids,
                           PoidsItem W,
                           Item it,
                           ref Selection sel,
                           ref Selection selMax,
                           ref Benefice benefMax )
{
  if (it > NB_ITEMS) {
    # La selection generee est acceptable: on verifie
    # si elle est mieux que la meilleure selection vue jusqu'a present.
    mettreAJourMax( sel, benefice(benefs, sel), selMax, benefMax );
  } else









  }
}

procedure remplirSacMarcheArriere( Benefices benefs, Poids poids, PoidsItem W )
          returns Selection selMax
{
  assert( ub(benefs) == ub(poids), "Nombre incorrect d'elements" );

  Selection sel = ([ub(benefs)] false);
  Benefice benefMax = 0;
  remplirSacMARec( benefs, poids, W, 1, sel, selMax, benefMax );
}
```

```
##############################################################################
# Exploration de type Branch−and−bound de l'espace des selections...
##############################################################################

procedure beneficeEstime( Selection sel,
                          Benefices benefs, Poids poids,
                          PoidsItem W,
                          Item it )
          returns Benefice b
{









}

procedure remplirSacBBRec( Benefices benefs,
                           Poids poids,
                           PoidsItem W,
                           Item it,
                           ref Selection sel,
                           ref Selection selMax,
                           ref Benefice benefMax )
{
  if (it > NB_ITEMS) {
    # La selection generee est acceptable: on verifie
    # si elle est mieux que la meilleure selection vue jusqu'a present.
    mettreAJourMax( sel, benefice(benefs, sel), selMax, benefMax );
  } else









  }
}

procedure remplirSacBB( Benefices benefs, Poids poids, PoidsItem W )
          returns Selection selMax
{
  assert( sontOrdonnesBeneficeParUniteDePoids(benefs, poids),
          "Pas ordonnes selon rapport benefice par unite de poids?!" );

  Selection sel = ([ub(benefs)] false);  # Aucun item encore selectionne.
  Benefice benefMax = 0;
  remplirSacBBRec( benefs, poids, W, 1, sel, selMax, benefMax );
}
```