```
  Help
/*
 * Writen by David Pommier <david.pommier@gmail.com>
 * INRIA 2009
 */

#include "gd_list.h"


/**
 * allocates a contains.c * @param ind key
 * @param val value
 * @return a pointeur to PremiaSparsePoint
 */
PremiaSparsePoint * premia_sparse_point_create(const PnlVec
    tInt * ind,int val)
{
  PremiaSparsePoint *C;
  if((C=malloc(sizeof(PremiaSparsePoint)))==NULL) return
    NULL;
  C->index=pnl_vect_int_copy(ind);
  C->value=val;
  return C;
}

/**
 * allocates a contains.c * @param ind key
 * @param val value
 * @return a pointeur to PremiaSparsePoint
 */
PremiaSparsePoint * premia_sparse_point_clone(PnlVectInt *
    ind,int val)
{
  PremiaSparsePoint *C;
  if((C=malloc(sizeof(PremiaSparsePoint)))==NULL) return
    NULL;
  C->index=malloc(sizeof(PnlVectInt));
  C->index->owner=0;
  C->index->size=ind->size;
  C->index->array=&(ind->array[0]);
  //C->index=ind;
  C->value=val;
  return C;
}

/**
 * allocates a contains - copy constructor.
 * @param C2 contains pointer
 * @return a pointeur to PremiaSparsePoint
 */
PremiaSparsePoint * premia_sparse_point_copy(const Premia
    SparsePoint *C2)
{
  PremiaSparsePoint *C;
  if((C=malloc(sizeof(PremiaSparsePoint)))==NULL) return
    NULL;
  C->index=pnl_vect_int_copy(C2->index);
  C->value=C2->value;
  return C;
}

/**
 * free a contains
 * @param C address of a contains
 */
void premia_sparse_point_free(PremiaSparsePoint **C)
{
  if (*C != NULL)
    {
      pnl_vect_int_free(&((*C)->index));
      free(*C);
      *C=NULL;
    }
}

/**
 * Prints a contains to a file
 *
 * @param fic a file descriptor.
 * @param C a Contains pointer.
 */
void premia_sparse_point_fprint(FILE *fic,PremiaSparsePoint
     *C)
{
  pnl_vect_int_print(C->index);
  fprintf(fic," Index  %d {n",C->value);
}


/**
 * Add - do nothing in this case
 *
 * @param C a PremiaSparsePoint pointer, C.Value Value.
 * @param C2 a Contains pointer.
 */
void  premia_sparse_point_add(PremiaSparsePoint *C,const
    PremiaSparsePoint *C2)
{}

/**
 * Less compute relation C1<C2
 *
 * @param C1 a PremiaSparsePoint pointer.
 * @param C2 a Contains pointer.
 * @return a int C1<C2
 */
int  premia_sparse_point_less(const PremiaSparsePoint *C1,
    const PremiaSparsePoint *C2)
{return pnl_vect_int_less(C1->index,C2->index);}

/**
 * Equal compute relation C1==C2
 *
 * @param C1 a PremiaSparsePoint pointer.
 * @param C2 a Contains pointer.
 * @return a int C1==C2
 */
int  premia_sparse_point_equal(const PremiaSparsePoint *C1,
    const PremiaSparsePoint *C2)
{return pnl_vect_int_equal(C1->index,C2->index);}
```

References