```
   Help
#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2008+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
#else
/***********************************************************
*   CPS - A simple C PDE solver                           *
*                                                         *
*   Copyright (c) 2007,                                   *
*     Maya Briani       <m.briani@iac.rm.cnr.it>,         *

*     Francesco Ferreri <francesco.ferreri@gmail.com>,    *
*     Roberto Natalini  <r.natalini@iac.rm.cnr.it>,       *
*     Marco Papi        <m.papi@iac.rm.cnr.it>            *
*                                                         *

**********************************************************/
#include <stdlib.h>
#include "cps_function.h"
#include "cps_pde.h"
#include "cps_pde_term.h"
#include "cps_pde_integral_term.h"
#include "cps_utils.h"
#include "cps_assertions.h"

int pde_create(pde **p){

  STANDARD_CREATE(p,pde);
  return OK;
}

int pde_destroy(pde **p){

  int k;
  pde_term *tmp;

  /* destroy pde terms */
  for(k = 0; k < (*p)->terms_count; k++){
    tmp = (*p)->terms[k];
    pde_term_destroy(&tmp);
  }
  if(pde_has_integral_term(*p)){
    pde_integral_term_destroy(&((*p)->integral_term));
  }

  STANDARD_DESTROY(p);

  return OK;
}


int pde_add_term(pde *p, pde_term *t){
  REQUIRE("pde_not_null",(p != NULL));
  REQUIRE("pde_term_not_null",(t != NULL));
  REQUIRE("pde_term_count_available", p->terms_count < (
    PDE_MAX_TERMS - 1));

  p->terms[p->terms_count] = t;
  p->terms_count = p->terms_count + 1;
  return OK;
}

int pde_set_source_term(pde *p, const function *f){
  REQUIRE("pde_not_null",(p != NULL));
  REQUIRE("function_not_null", f != NULL);

  p->source_term = f;

  ENSURE("pde_has_source_term", pde_has_source_term(p));
  return OK;
}

int pde_set_integral_term(pde *p, pde_integral_term *t){
  REQUIRE("pde_not_null",(p != NULL));
  REQUIRE("term_not_null", t != NULL);

  p->integral_term = t;

  ENSURE("pde_has_integral_term", pde_has_integral_term(p)
    );
  return OK;
```

```
}

int pde_has_source_term(const pde *p){
  REQUIRE("pde_not_null",(p != NULL));

  return (p->source_term != NULL);
}

int pde_has_integral_term(const pde *p){
  REQUIRE("pde_not_null",(p != NULL));

  return (p->integral_term != NULL);
}

int pde_term_start(pde *pde){

  REQUIRE("pde_not_null", pde != NULL);
  pde->item = 0;
  return OK;
}

int pde_term_after(pde *pde){

  REQUIRE("pde_not_null", pde != NULL);
  return(pde->item == pde->terms_count);
}

int pde_term_forth(pde *pde){

  REQUIRE("pde_not_null", pde != NULL);
  REQUIRE("not_after", !pde_term_after(pde));

  pde->item++;
  return OK;
}

int pde_term_item(pde *pde, pde_term **term){

  REQUIRE("pde_not_null", pde != NULL);
  REQUIRE("not_after", !pde_term_after(pde));
  *term = pde->terms[pde->item];
  return OK;
}
/* end of pde.c */

#endif //PremiaCurrentVersion
```

References