```
//  file: pointer_test.cpp
//
//  Program to demonstrate the use of void pointers in C/C++.
//
//  Programmer:  Dick Furnstahl  furnstahl.1@osu.edu
//
//  Revision history:
//      01/14/04  original version, converted pointer_test.c
//      01/16/05  added dummy return for functions [later returned_result]
//      01/23/11  added printout of returned result
//      01/25/14  switched to just use typedef struct
//
//  Notes:
//   * Based on the use of void pointers in gsl library, using the
//      discussion in "Practical C++" and online sources.
//   * As a convention (advocated in "Practical C++"), we'll append
//      "_ptr" to all pointers.
//   * Each of the sample functions has arguments like those in the
//      GSL integration routines.  We illustrate in each two
//      ways to access the passed values (by setting them
//      equal to local variables or pointers).
//   * If my_struct is a structure containing doubles a,b,c,
//      then we can set a = b + 1. using:
//         my_struct.a = my_struct.b + 1
//     If my_struct_ptr is a pointer to a structure containing
//      doubles a,b,c, then we can set a = b + 1. using:
//         my_struct_ptr->a = my_struct_ptr->b + 1.
//   * C/C++ uses for void for two different purposes:
//       * if a function is declared as void, it means that
//          the function doesn't return anything
//       * if a pointer is declared as void, it means that the
//          pointer can point to ANY type of thing (int, double,
//          structure, etc.).  This is the usage in GSL functions.
//   * When we want to refer to the thing pointed to by the
//      void pointer, we have to "cast" it; that is, we have
//      to specify the type.  E.g., if we declare params_ptr
//      as a void pointer:
//         void * params_ptr
//      we can dereference a double using
//         (double *) params_ptr
//      (See the examples below!)
//
//    * The output of the program should be:
//
//  Program to illustrate the use of void pointers in C/C++
//
//  passed_int and *passed_ptr should have the same value: 5
//   so . . . passed_int = 5 and *passed_ptr = 5
//
//  passed_double and *passed_ptr should have the same value: 13.3
//   so . . . passed_double = 13.3 and *passed_ptr = 13.3
//
//  Expected: a = 1.1, b = 2.2, c = 3.3, num = 4
//  Passed:   a = 1.1, b = 2.2, c = 3.3, num = 4
//  Expected: a = -1.4, b = 20.1, c = 0.9, num = 2
//  Passed:   a = -1.4, b = 20.1, c = 0.9, num = 2
//
//
///*********************************************************************

// include files
#include <iostream>             // note that .h is omitted
#include <iomanip>              // note that .h is omitted
using namespace std;            // we need this when .h is omitted

// function prototypes
double f_int (double x, void *params_ptr);
double f_double (double x, void *params_ptr);
double f_struct (double x, void *params_ptr);
```



```
double f_osu_parameters (double x, void *params_ptr);

typedef struct                  // define a type to hold parameters
{
  double a;
  double b;
  double c;
  int num;
}
osu_parameters;                 // now we can define a structure of this type
                                //   using the keyword "osu_parameters"

//************************* main program **************************
int
main (void)
{
  void *params_ptr;             // the void pointer to be passed to functions

  int my_int = 5;               // initialize some variables to pass
  double x = 2.1;
  double my_double = 13.3;

  osu_parameters my_coefficients;

  // set up values in the osu_parameters structure
  my_coefficients.a = -1.4;
  my_coefficients.b = 20.1;
  my_coefficients.c = 0.9;
  my_coefficients.num = 2;

  cout << endl << "Program to illustrate the use of void pointers in C/C++"
       << endl << endl;

  params_ptr = &my_int;         // first point to an integer
  cout << "passed_int and *passed_ptr should have the same value: "
    << my_int << endl;
  double returned_result = f_int (x, params_ptr);
  cout << "The returned result is " << returned_result
       << ". Is it correct?" << endl << endl;

  params_ptr = &my_double;      // then point to a double
  cout << "passed_double and *passed_ptr should have the same value: "
    << my_double << endl;
  returned_result = f_double (x, params_ptr);
  cout << "The returned result is " << returned_result
       << ". Is it correct?" << endl << endl;

  params_ptr = &my_coefficients;        // now point to an osu_parameter structu
re
  cout << "Expected: a = " << my_coefficients.a
    << ", b = " << my_coefficients.b
    << ", c = " << my_coefficients.c
    << ", num = " << my_coefficients.num << endl;
  returned_result = f_osu_parameters (x, params_ptr);
  cout << "The returned result is " << returned_result
       << ". Is it correct?" << endl << endl;

  return (0);                   // successful completion
}

//*************************************************************



//************************ f_int ***************************
double
f_int (double x, void *params_ptr)
{
  int *passed_ptr;
```

```
  int passed_int = *(int *) params_ptr;
  passed_ptr = (int *) params_ptr;

  cout << " so . . . passed_int = " << passed_int
    << " and *passed_ptr = " << *passed_ptr << endl;

  return (x * (double) passed_int);     // sample return value
}

//*********************************************************



//************************ f_double ************************
double
f_double (double x, void *params_ptr)
{
  double *passed_ptr;

  double passed_double = *(double *) params_ptr;
  passed_ptr = (double *) params_ptr;

  cout << " so . . . passed_double = " << passed_double
    << " and *passed_ptr = " << *passed_ptr << endl;

  return (x * passed_double);   // sample return value
}

//*********************************************************



//******************** f_osu_parameters *********************
double
f_osu_parameters (double x, void *params_ptr)
{
  osu_parameters *passed_ptr;
  passed_ptr = (osu_parameters *) params_ptr;

  double passed_double_1 = ((osu_parameters *) params_ptr)->a;
  double passed_double_2 = passed_ptr->b;
  double passed_double_3 = ((osu_parameters *) params_ptr)->c;
  int passed_int = ((osu_parameters *) params_ptr)->num;

  cout << "Passed:  a = " << passed_double_1
    << ", b = " << passed_double_2
    << ", c = " << passed_double_3 << ", num = " << passed_int << endl;

  return (x * passed_double_1); // sample return value
}

//*********************************************************
```