C++ Lab 05 - C Structures and C++ Classes

2.680 Unmanned Marine Vehicle Autonomy, Sensing and Communications

Ten Short CPP Labs

IAP 2024

Michael Benjamin, mikerb@mit.edu
Department of Mechanical Engineering
MIT, Cambridge MA 02139

1 Lab Five Overview and Objectives

In this lab C++ *classes* are introduced, after first introducing and using their C predecessor, the structure. After building our first application using C++ classes, our attention is turned to generating random numbers in C/C++ while gaining a bit more experience with the command line environment. This lab will contain a bit more pointers to reading than prior labs.

- C-Style Structures
- Simple C++ Classes and Encapsulated Variables and Methods
- Complex C++ Classes (Class Containing Other Classes)
- Generating Random Numbers and Seeding a Random Number Generator

2 C-Style Structures

For certain kinds of data it is natural to group the data into a structure. For example a *person* may have a name, birth-date, and address, and a *publication* may have a title, author, publication date and so on. Structures have been part of C from its inception, and C++ is essentially an extension of the C structure into C++ classes where classes not only just hold data like structures, but also have methods associated with the data. So before diving into C++ classes, it's worth taking a look at simple C structures and building a couple simple programs with them. For a quick intro to C structures, see one or more of the following:

- `http://www.cplusplus.com/doc/tutorial/structures`
- `http://www.learncpp.com/cpp-tutorial/47-structs`
- `http://www.cprogramming.com/tutorial/c/lesson7.html`

2.1 Exercise 1: Defining and Using a Simple Vertex Structure in C

Write a program that defines a simple `Vertex` structure (a point on the x-y plane) having two integer components for the x and y values. Your program should generate 5 random vertices each having integer values in the range of [-100,100], and write them to the terminal. To generate a random number, use the `rand()` function described here:

- `http://www.cplusplus.com/reference/cstdlib/rand`
- `http://www.tutorialspoint.com/c_standard_library/c_function_rand.htm`

Call your file `rand_vertices.cpp` and build it to the executable `rand_vertices`. When your program runs, it should be invocable from the command line with:

```
$ ./rand_vertices
x=-93,y=-51
x=-27,y=-42
x=30,y=-28
x=3,y=69
x=9,y=57
```

The solution to this exercise is in Section 5.1.

2.2 Exercise 2: Generating Random Vertices and Saving to File

Write a program that augments your program from the first exercise to (a) write your randomly generated vertices to a file (b) specified from the command line with the command line argument `--filename=file.txt`, and (c) for a number of vertices also specified on the command line with the command line argument `--amt=N`. If either the filename or ammount is not specified, the program should exit immediately with a usage message.

Call your file `rand_vertices_file.cpp` and build it to the executable `rand_vertices_file`. When your program runs, it should be invocable from the command line with:

```
$ ./rand_vertices_file --filename=test.txt --amt=5
$ cat test.txt
x=-93,y=-51
x=-27,y=-42
x=30,y=-28
x=44,y=-22
x=23,y=9
```

The solution to this exercise is in Section 5.2.

3 An Introduction to C++ Classes

Classes in C++ can be thought of as extension or generalization of a C-style structure. Unlike a structure, *methods* (functions) may be associated with a class. Components of the class (member variables) may have their access configured to allow direct setting as with C-style structures in our first examples, or the access may be configured to be only allowed by class methods. Classes (like structures) may have components that are themselves classes or structures. Unlike structures however, classes may have subclasses which allows for properties and methods in superclasses to be shared among subclasses.

In short, the move to classes represents a shift toward *object-oriented* programming. My suggestion is to read the first discussion link below on object oriented programming and the following introductions to classes. Then we will return to do a simple exercise.

- http://www.learncpp.com/cpp-tutorial/81-welcome-to-object-oriented-programming
- http://www.learncpp.com/cpp-tutorial/82-classes-and-class-members
- http://www.learncpp.com/cpp-tutorial/83-public-vs-private-access-specifiers
- http://www.learncpp.com/cpp-tutorial/84-access-functions-and-encapsulation
- http://www.learncpp.com/cpp-tutorial/85-constructors
- http://www.cplusplus.com/doc/tutorial/classes/

3.1 Exercise 3: Defining and Using a Simple Vertex Class in C++

Write a program that replaces the `Vertex` C-style structure used in the first two exercises with a C++ Vertex class. In addition to having two integer member variables, x and y, the class should have two methods. A method for setting the x-y values to a random number in a given range, and a method for returning a string representation of the vertex. Your `main()` routine should then not be involved in the business of generating random numbers and string formatting.

Your program should consist of three files: `rand_vertices_class.cpp` will contain your `main()` function and the `VertexSimple` class will be in files `VertexSimple.h` and `VertexSimple.cpp`. The executable should be called `rand_vertices_class`. When your program runs, it should be invocable from the command line with:

```
$ ./rand_vertices_class --filename=test.txt --amt=5
$ cat test.txt
x=-93,y=-51
x=-27,y=-42
x=30,y=-28
x=44,y=-22
x=23,y=9
```

The solution to this exercise is in Section 5.3.

3.2 Exercise 4: The Object-Oriented/C++ Has-A Relationship

C++ classes, like C-style structures, allow for components to be other classes. This is often referred to as the *has-a* relationship. In this exercise we use the `Vertex` class to be used as a component in another class holding a set of line segments, comprised of the vertices.

Write a program defining a new class, `SegList`, which primarily holds a list of line segments implied from a list of vertices in the x-y plane. Your `SegList` class should have a vector of `Vertex` objects, and be defined with a method for accepting new vertices and a method for producing a string representation the entire set of vertices. This method for producing a string should utilize the method defined on the `Vertex` class for producing a string representation of the single vertex.

Your program should consist of five files:

- `rand_seglist.cpp` will contain your `main()` function.
- `Vertex.h` will define your Vertex class
- `Vertex.cpp` will hold your method implementations for the Vertex class
- `SegList.h` will define your SegList class
- `SegList.cpp` will hold your method implementations for the SegList class

When your program runs, it should be invocable from the command line with:

```
$ ./rand_seglist --filename=test.txt --amt=5
$ cat test.txt
x=-93,y=-51,x=-27,y=-42,x=30,y=-28,x=44,y=-22,x=23,y=9
```

The solution to this exercise is in Section 5.4.

4 Seeding the Random Number Generator

The random numbers generated so far have only been pseudo-random. If you run your previous program, `rand_seglist`, a few times in a row on the same computer, you may see something like the below - the same random numbers on each run:

```
$ ./rand_seglist --filename=test.txt --amt=5 ; cat test.txt
x=-93,y=-51,x=-27,y=-42,x=30,y=-28,x=44,y=-22,x=23,y=9
$ ./rand_seglist --filename=test.txt --amt=5 ; cat test.txt
x=-93,y=-51,x=-27,y=-42,x=30,y=-28,x=44,y=-22,x=23,y=9
$ ./rand_seglist --filename=test.txt --amt=5 ; cat test.txt
x=-93,y=-51,x=-27,y=-42,x=30,y=-28,x=44,y=-22,x=23,y=9
```

A couple command-line points from above: The `cat` command just takes one or more filenames and dumps the contents of the file to the terminal. The semicolon preceding the `cat` command on each line just separates multiple command line invocations to occur one after the other. Otherwise you would have to type the separately on two lines like:

```
$ ./rand_seglist --filename=test.txt --amt=5
$ cat test.txt
x=-93,y=-51,x=-27,y=-42,x=30,y=-28,x=44,y=-22,x=23,y=9
```

Returning to the issue of randomness, you will get the same random number each time the program is invoked unless you *seed* the generator. Requiring this extra step may seem counter-intuitive, but if you think about it, sometimes getting the same sequence of random numbers is very useful. When debugging a program that is based on the random sequence it is often helpful to reliably re-create the fault so that your debugging efforts are focused.

We're interested in getting a sequence of pseudo-random numbers that looks more random, at least different each time the program is invoked. We say "pseudo" random because even in the best case efforts described here, the sequence is not truly random in the purest definitions of random, but it will be pretty darn close. So, to get a more random number, we need to *seed* the generator by invoking the following function:

```
srand(unsigned int);
```

4.1 Using UTC Time to Seed the Random Number Generator

The `srand()` function takes an integer argument, or 1 by default if left unspecified. So the trick is to provide it an argument that is itself random, or at least different each time it is called. The latter can be achieved by using the UTC time in seconds, i.e., the number of seconds since January 1, 1970. There is a C library, the *C Time Library* that provides a function to get UTC time (the `time()` function). More information can be found at the below two links, but basic usage is shown below on this page.

- For information on the C Time Library:
  http://www.cplusplus.com/reference/ctime
- For information on the `time()` function in the C Time Library:
  http://www.cplusplus.com/reference/ctime/time

So the following will seed a different sequence of random numbers each time it is called (assuming it's not called twice within the same second):

```
#include <time.h>
...
srand(time(NULL));
```

4.2 Using The Process ID to Seed the Random Number Generator

To get a bit more randomness, it's not uncommon to include another factor in the seed generation. One way to do this is to get the *process id* of the program. In Linux-like systems (of which I include Mac OS X in this category), each time a program is launched, the operating system gives it a *process id*, sometime PID for short. You can see this in action if you type `top` in the Terminal window. You will see many of the processes currently running on your computer with certain statistics (see the man page for `top` if you're curious. The left-most column in the `top` output is the PID. For a side-note, you can force-quit or kill any runaway or hung application with the command if you know its PID:

```
$ kill -9 PID
```

Each time the operating system launches a process, it uses a new process ID. So if you launch your `rand_vertices` program from the first exercise multiple times, each time it gets a new PID even though it's the same program running each time.
The PID can be obtained within the C++ program using the following function call and library:

```
#include <unistd.h>
...
unsigned long int getpid();
```

We now have what we need to know to try the final two short exercises of this lab.

4.3 Exercise 5: Experimenting with Top, Kill, Sleep and PIDs

Modify your program from the first exercise to include the following library and the following single line before returning. This will cause your program to just as the function implies, *sleep*, for 30 seconds, consuming virtually no CPU load, before resuming its business. We add it here so our otherwise super fast program is actually alive for a while longer so we can see it as an active process on your machine:

```
#include <unistd.h>
...
sleep(30);
return(0);
```

Call your file `rand_vertices_sleep.cpp` and your executable `rand_vertices_sleep` After building the program, open another terminal window so you can run `top` in one terminal window and run your program in the other:

```
$ top
```

```
$ ./random_vertices_sleep
```

Once your program is launched, look for its name in the `COMMAND` column in the output of `top`. Note its process ID in the left-most column and then in a third terminal window invoke:

```
$ kill -9 67032 (or whatever the PID is in your case)
```

You have now touched a few of the many-more-to-come nice tools of the command-line, the *nix (OSX and/or GNU/Linux) operating system and its relationship to C/C++.

The solution to this exercise is in Section 5.5.

4.4 Exercise 6: Seeding the Random Number Generator with Time and PID

Write a program that extends the `rand_vertices` program of the first lab to seed the random number generator with a seed in the range of zero to 1 million, derived in part from the UTC time and the program's PID. Your program should print the random vertices as before, but then also print the following three pieces of information to the terminal:

```
time_seed: 1420645005
pid_seed:  69033
rand_seed: 630165
```

Call your file `rand_vertices_seed.cpp` and build it to the executable `rand_vertices_seed`. When your program runs, it should be invocable from the command line with:

```
$ ./rand_vertices_seed
time_seed: 1420648632
pid_seed:  69383
rand_seed: 34056
x=92,y=-69
x=-16,y=55
x=59,y=-100
x=70,y=75
x=90,y=-53
$ ./rand_vertices_seed
time_seed: 1420648638
pid_seed:  69384
rand_seed: 98992
x=44,y=-79
x=10,y=-58
x=-73,y=-1
x=69,y=-18
x=48,y=59
```

Verify that different random output is now being generate on each invocation.

The solution to this exercise is in Section 5.6.

5 Solutions to Exercises

5.1 Solution to Exercise 1

```
/*--------------------------------------------------------------------------*/
/*  FILE:  rand_vertices.cpp (Fifth C++ Lab Exercise 1)                    */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/rand_vertices.cpp            */
/*  BUILD: g++ -o rand_vertices rand_vertices.cpp                           */
/*  RUN:   ./rand_vertices                                                  */
/*--------------------------------------------------------------------------*/

#include <iostream>      // For use of the cout function
#include <cstdlib>       // For use of the rand function

using namespace std;

int main()
{
  struct Vertex {
    int x;
    int y;
  };

  for(int i=0; i<5; i++) {
    Vertex vertex;
    vertex.x = (rand() % 201) + -100;
    vertex.y = (rand() % 201) + -100;
    cout << "x=" << vertex.x << ",y=" << vertex.y << endl;
  }

 return(0);
}
```

5.2 Solution to Exercise 2

```
/*----------------------------------------------------------------------*/
/*  FILE:  rand_vertices_file.cpp (Fifth C++ Lab Exercise 2)            */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/rand_vertices_file.cpp   */
/*  BUILD: g++ -std=c++0x -o rand_vertices_file rand_vertices_file.cpp  */
/*  RUN:   rand_vertices_file --filename=vertices.txt --amt=5           */
/*----------------------------------------------------------------------*/

#include <vector>
#include <iostream>      // For use of the cout function
#include <cstdlib>       // For use of the atoi function
#include <cstdio>        // For use of the fopen, fclose, fprintf functions

using namespace std;

int main(int argc, char **argv)
{
  // Part 1: Handle and check command line arguments
  string filename;
  int    amount = 0;
  for(int i=1; i<argc; i++) {
    string argi = argv[i];
    if(argi.find("--filename=") == 0)
      filename = argi.substr(11);
    else if(argi.find("--amt=") == 0) {
      string str_amt = argi.substr(6);
      amount = atoi(str_amt.c_str());
    }
  }

  if((filename=="") || (amount<=0)) {
    cout << "Usage: rand_vertices --filename=FILE --amt=AMT" << endl;
    return(1);
  }

  // Part 2: Define the C-style structure and create N instances into a vector
  struct Vertex {
    int x;
    int y;
  };

  vector<Vertex> vertices;

  for(int i=0; i<amount; i++) {
    Vertex vertex;
    vertex.x = (rand() % 201) + -100;
    vertex.y = (rand() % 201) + -100;
    vertices.push_back(vertex);
  }

  // Part 3: Open the output file and write the vector of vertices
  FILE *f = fopen(filename.c_str(), "w");
  if(!f) {
    cout << "Unable to open file: " << filename << endl;
    return(1);
  }

  for(int i=0; i<vertices.size(); i++) {
    fprintf(f, "x=%d,", vertices[i].x);
    fprintf(f, "y=%d\n", vertices[i].y);
  }

  fclose(f);
  return(0);
}
```

5.3 Solution to Exercise 3

```
/*----------------------------------------------------------------------*/
/*  FILE:  rand_vertices_class.cpp (Fifth C++ Lab Exercise 3)           */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/rand_vertices_class.cpp  */
/*  BUILD: g++ -o rand_vertices_class   \                               */
/*         VertexSimple.cpp rand_vertices_class.cpp                     */
/*  RUN:   rand_vertices_class --filename=vertices.txt 100              */
/*----------------------------------------------------------------------*/

#include <vector>
#include <iostream>      // For use of the cout function
#include <cstdlib>       // For use of the atoi function
#include <cstdio>        // For use of the fopen, fclose, fprintf functions
#include "VertexSimple.h"

using namespace std;

int main(int argc, char **argv)
{
  // Part 1: Handle and check command line arguments
  string filename;
  int    amount = 0;
  for(int i=1; i<argc; i++) {
    string argi = argv[i];
    if(argi.find("--filename=") == 0)
      filename = argi.substr(11);
    else if(argi.find("--amt=") == 0) {
      string str_amt = argi.substr(6);
      amount = atoi(str_amt.c_str());
    }
  }

  if((filename=="") || (amount<=0)) {
    cout << "Usage: rand_vertices_class --filename=FILE --amt=AMT" << endl;
    return(1);
  }

  vector<VertexSimple> vertices;
  for(int i=0; i<amount; i++) {
    VertexSimple vertex;
    vertex.setRandom(-100, 100);
    vertices.push_back(vertex);
  }

  // Part 3: Open the output file and write the vector of vertices
  FILE *f = fopen(filename.c_str(), "w");
  if(!f) {
    cout << "Unable to open file: " << filename << endl;
    return(1);
  }

  for(int i=0; i<vertices.size(); i++) {
    string vertex_spec = vertices[i].getSpec();
    fprintf(f, "%s\n", vertex_spec.c_str());
  }

  fclose(f);
  return(0);
}
```

```
/*----------------------------------------------------------------------*/
/*  FILE:  VertexSimple.h                                               */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/VertexSimple.h           */
/*----------------------------------------------------------------------*/

#ifndef VERTEX_SIMPLE_HEADER
#define VERTEX_SIMPLE_HEADER

#include <string>

class VertexSimple {
 public:
  VertexSimple() {m_x=0; m_y=0;};
  ~VertexSimple() {};

  void setRandom(int min, int max);

  std::string getSpec() const;

 protected:
  int m_x;
  int m_y;
};
#endif
```

```
/*----------------------------------------------------------------------*/
/*  FILE:  VertexSimple.cpp                                             */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/VertexSimple.cpp         */
/*----------------------------------------------------------------------*/

#include <sstream>
#include "VertexSimple.h"

using namespace std;

//----------------------------------------------------------------------
// Procedure: getSpec()

string VertexSimple::getSpec() const
{
  stringstream ss;
  ss << "x=" << m_x << ",y=" << m_y;
  return(ss.str());
}

//----------------------------------------------------------------------
// Procedure: setRandom()

void VertexSimple::setRandom(int min, int max)
{
  if(min >= max)
    return;

  int range = max - min;
  m_x = (rand() % range) + min;
  m_y = (rand() % range) + min;
}
```

5.4 Solution to Exercise 4

```
/*----------------------------------------------------------------------*/
/*  FILE:  rand_seglist.cpp (Fifth C++ Lab Exercise 4)                  */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/rand_seglist.cpp         */
/*  BUILD: g++ -o rand_seglist Vertex.cpp SegList.cpp rand_seglist.cpp  */
/*  RUN:   rand_seglist --filename=vertices.txt --amt=100               */
/*----------------------------------------------------------------------*/

#include <iostream>      // For use of the cout function
#include <cstdlib>       // For use of the atoi function
#include <cstdio>        // For use of the fopen, fclose, fprintf functions
#include "SegList.h"

using namespace std;

int main(int argc, char **argv)
{
  // Part 1: Handle and check command line arguments
  string filename;
  int    amount = 0;
  for(int i=1; i<argc; i++) {
    string argi = argv[i];
    if(argi.find("--filename=") == 0)
      filename = argi.substr(11);
    else if(argi.find("--amt=") == 0) {
      string str_amt = argi.substr(6);
      amount = atoi(str_amt.c_str());
    }
  }

  if((filename=="") || (amount<=0)) {
    cout << "Usage: rand_seglist --filename=FILE --amt=AMT" << endl;
    return(1);
  }

  // Part 2: Build the SegList with randomly generated vertices
  SegList seglist;
  for(int i=0; i<amount; i++) {
    Vertex vertex;
    vertex.setRandom(-100, 100);
    seglist.addVertex(vertex);
  }

  // Part 3: Open the output file and write the vector of vertices
  FILE *f = fopen(filename.c_str(), "w");
  if(!f) {
    cout << "Unable to open file: " << filename << endl;
    return(1);
  }

  string seglist_spec = seglist.getSpec();
  fprintf(f, "%s\n", seglist_spec.c_str());

  fclose(f);
  return(0);
}
```

```
/*----------------------------------------------------------------------*/
/*  FILE:  Vertex.h                                                     */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/Vertex.h                 */
/*----------------------------------------------------------------------*/

#ifndef VERTEX_HEADER
#define VERTEX_HEADER

#include <string>

class Vertex {
 public:
  Vertex() {m_x=0; m_y=0;};
  Vertex(int x, int y) {m_x=x, m_y=y;};
  ~Vertex() {};

  void setXY(int x, int y) {m_x=x; m_y=y;};
  void setRandom(int min, int max);
  void setRandom(int xmin, int xmax, int ymin, int ymax);

  int  getX() const {return(m_x);};
  int  getY() const {return(m_y);};

  std::string getSpec() const;

 protected:
  int m_x;
  int m_y;
};
#endif
```

```
/*----------------------------------------------------------------------*/
/*  FILE:  Vertex.cpp                                                   */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/Vertex.cpp               */
/*----------------------------------------------------------------------*/

#include <sstream>
#include "Vertex.h"

using namespace std;

//----------------------------------------------------------------------
// Procedure: getSpec()

string Vertex::getSpec() const
{
  stringstream ss;
  ss << "x=" << m_x << ",y=" << m_y;
  return(ss.str());
}

//----------------------------------------------------------------------
// Procedure: setRandom()

void Vertex::setRandom(int min, int max)
{
  setRandom(min, max, min, max);
}

//----------------------------------------------------------------------
// Procedure: setRandom()

void Vertex::setRandom(int xmin, int xmax, int ymin, int ymax)
{
  if((xmin >= xmax) || (ymin >= ymax))
    return;

  int xrange = xmax - xmin;
  m_x = (rand() % xrange) + xmin;

  int yrange = ymax - ymin;
  m_y = (rand() % yrange) + ymin;
}
```

```
/*----------------------------------------------------------------------*/
/*  FILE:  SegList.h                                                    */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/SegList.h                */
/*----------------------------------------------------------------------*/

#ifndef SEGLIST_HEADER
#define SEGLIST_HEADER

#include <string>
#include <vector>
#include "Vertex.h"

class SegList {
 public:
  SegList() {};
  ~SegList() {};

  void addVertex(double x, double y);
  void addVertex(Vertex vertex) {m_vertices.push_back(vertex);};

  unsigned int size() {return(m_vertices.size());};
  double totalEdgeLength() const;
  std::string getSpec() const;

 protected:

  std::vector<Vertex> m_vertices;
  std::string         m_label;
};

#endif
```

```
/*--------------------------------------------------------------------*/
/*  FILE:  SegList.cpp                                                */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/SegList.cpp            */
/*--------------------------------------------------------------------*/

#include <cmath>           // For using the hypot function
#include "SegList.h"

using namespace std;

//--------------------------------------------------------------------
// Procedure: addVertex()

void SegList::addVertex(double x, double y)
{
  Vertex vertex(x,y);
  m_vertices.push_back(vertex);
}

//--------------------------------------------------------------------
// Procedure: totalEdgeLength

double SegList::totalEdgeLength() const
{
  double total = 0;
  for(unsigned int i=0; i<m_vertices.size(); i++) {
    if((i+1) < m_vertices.size()) {
      double xdelta = m_vertices[i].getX() - m_vertices[i+1].getX();
      double ydelta = m_vertices[i].getY() - m_vertices[i+1].getY();
      double dist = hypot(xdelta, ydelta);
      total += dist;
    }
  }

  return(total);
}

//--------------------------------------------------------------------
// Procedure: getSpec

string SegList::getSpec() const
{
  string spec;
  for(unsigned int i=0; i<m_vertices.size(); i++) {
    if(i != 0)
      spec += ",";
    spec += m_vertices[i].getSpec();
  }

  return(spec);
}
```

5.5 Solution to Exercise 5

```
/*----------------------------------------------------------------------*/
/*  FILE:  rand_vertices_sleep.cpp (Fifth C++ Lab Exercise 5)           */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/rand_vertices_sleep.cpp  */
/*  BUILD: g++ -o rand_vertices_sleep rand_vertices_sleep.cpp           */
/*  RUN:   ./rand_vertices_sleep                                        */
/*----------------------------------------------------------------------*/

#include <iostream>      // For use of the cout function
#include <cstdlib>       // For use of the rand function
#include <unistd.h>      // For use of the sleep function

using namespace std;

int main()
{
  struct Vertex {
    int x;
    int y;
  };

  for(int i=0; i<5; i++) {
    Vertex vertex;
    vertex.x = (rand() % 201) + -100;
    vertex.y = (rand() % 201) + -100;
    cout << "x=" << vertex.x << ",y=" << vertex.y << endl;
  }

  sleep(30);
  return(0);
}
```

5.6 Solution to Exercise 6

```
/*----------------------------------------------------------------------*/
/*  FILE:  rand_vertices_seed.cpp (Fifth C++ Lab Exercise 6)            */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/rand_vertices_seed.cpp   */
/*  BUILD: g++ -o rand_vertices_seed rand_vertices_seed.cpp             */
/*  RUN:   ./rand_vertices_seed                                         */
/*----------------------------------------------------------------------*/

#include <iostream>      // For use of the cout function
#include <cstdlib>       // For use of the rand function
#include <ctime>         // For use of the time function
#include <unistd.h>      // For use of the getpid function

using namespace std;

int main()
{
  struct Vertex {
    int x;
    int y;
  };

  unsigned long int tseed = time(NULL);
  unsigned long int pseed = getpid() + 1;

  unsigned int rseed = (tseed*pseed) % 50000;

  cout << "time_seed: " << tseed << endl;
  cout << "pid_seed:  " << pseed << endl;
  cout << "rand_seed: " << rseed << endl;

  srand(rseed);

  for(int i=0; i<5; i++) {
    Vertex vertex;
    vertex.x = (rand() % 201) + -100;
    vertex.y = (rand() % 201) + -100;
    cout << "x=" << vertex.x << ",y=" << vertex.y << endl;
  }

  return(0);
}
```