Appendix B

RANDOM NUMBERS

1. Introduction

2. Strategy for generating random numbers

3. Class and method development

Introduction to Random Numbers

- Variety is the spice of life. Computers, on the other hand, tend to be entirely predictable and hence rather dull. Random numbers provide a way to inject unpredictability into computer programs and therefore, sometimes, to make them better imitate external events.

- Random numbers have important applications to computer games, graphics displays, and simulations. Random numbers can add a great deal of interest, and, when a program is run repeatedly, it may show a range of behavior not unlike that of the natural system it is imitating.

- The header files `<cstdlib>` and `<stdlib.h>` provide random number generation routines in C++ systems. These routines can be used in place of the ones developed here. But system random-number generators are sometimes not very good, so it is worthwhile to see how better ones can be constructed.

- We shall design a **class** Random whose methods generate and return random numbers of various kinds.

Random Number Generation

- We start with one number and apply a series of arithmetic operations that will produce another number with no obvious connection to the first.
- The numbers we produce are not truly random at all, as each one depends in a definite way on its predecessor, and we should more properly speak of ***pseudorandom*** numbers.
- The number we use (and simultaneously change) is called the ***seed***.
- If the seed begins with the same value each time the program is run, then the whole sequence of pseudorandom numbers will be exactly the same, and we can reproduce any experimental results that we obtain.
- We include an option to initialize the seed according to the exact time of day measured in seconds, which should lead to different behavior every time the client program is run.
- We create a constructor for the **class** Random that uses a parameter **bool** pseudo. When pseudo has the value **true**, we shall generate random numbers starting from a predefined seed, whereas when pseudo is **false** we shall generate unreproducible random numbers.
- We declare the seed as a **private** data member of the **class** Random, so it will be accessible to all the methods and auxiliary functions in the class, but it cannot be accessed at all from outside the class.

Program Development

Class declaration:

```
class Random {
public:
   Random(bool pseudo = true);
//    Declare random-number generation methods here.
private:
   int reseed( );                  //    Re-randomize the seed.
   int seed,
       multiplier, add_on;         //    constants for use in arithmetic operations
};
```

Random-number calculation:

```
int Random :: reseed( )
/* Post:  The seed is replaced by a psuedorandom successor.  */
{
   seed = seed * multiplier + add_on;
   return seed;
}
```

Initialization:

```
Random :: Random(bool pseudo)
/* Post:  The values of seed, add_on, and multiplier are initialized.  The seed
          is initialized randomly only if pseudo ==  false.  */
{
   if (pseudo) seed = 1;
   else seed = time(NULL) % max_int;
   multiplier = 2743;
   add_on = 5923;
}
```

Random-Number Methods

Real values:

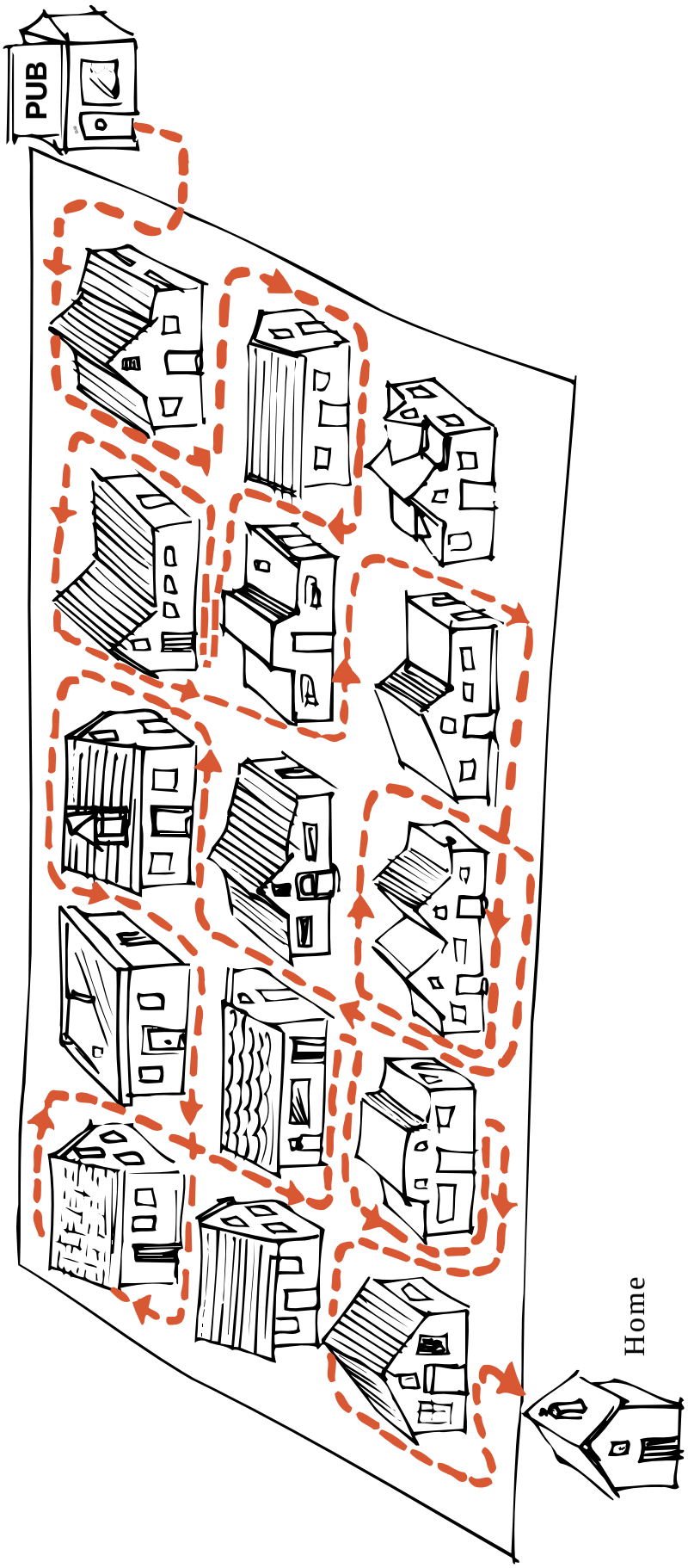
- The result is a real number uniformly distributed between 0 and 1. By ***uniformly distributed*** we mean that, if we take two intervals of the same length within the range of the method, then it is equally likely that the result will be in one interval as in the other.
- 0 may appear as a result but 1 may not. In mathematical terms, the range is the *half-open* interval [0, 1).

```
double Random :: random_real( )
/* Post:  A random real number between 0 and 1 is returned.  */
{
   double max = max_int + 1.0;
   double temp = reseed( );
   if (temp < 0) temp = temp + max;
   return temp/max;
}
```

Integer values:

- We generate integers in a range between two integers low and high, inclusive, such that every integer in that range is equally likely.

```
int Random :: random_integer(int low, int high)
/* Post:  A random integer between low and high (inclusive) is returned.  */
{
   if (low > high) return random_integer(high, low);
   else return ((int) ((high − low + 1) * random_real( ))) + low;
}
```



A random walk
Transp. 6, Sect. B.3, Program Development

Data Structures and Program Design In C++

Poisson Random Integers

- A more sophisticated form of random numbers is often needed for simulation projects, called a *Poisson distribution* of random integers.
- Start with a positive real number called the ***expected value*** v of the random numbers.
- A sequence of nonnegative integers is said to satisfy a ***Poisson distribution*** with expected value v if, over long subsequences, the mean (average) value of the integers in the sequence approaches v.
- The derivation of the following method and the proof that it works correctly require techniques from advanced mathematical statistics that are far outside the scope of this book.

```
int Random :: poisson(double mean)
/* Post: A random integer, reflecting a Poisson distribution with parameter mean,
         is returned. */
{
   double limit = exp(−mean);
   double product = random_real( );
   int count = 0;
   while (product > limit) {
      count++;
      product *= random_real( );
   }
   return count;
}
```