```
#ifndef BLUENOISE_H
#define BLUENOISE_H

#include <cmath>
#include <cstdlib>

// Provides type Vec<N,T>, a C++ wrapper around T[N] to
// encapsulate N-dimensional vectors.
#include "vec.h"

// Produce a sample uniformly chosen from the annulus between
// (radius,2*radius] around center x. Uses random() for underlying PRNG.
template<unsigned int N, class T>
void sample_annulus(T radius, const Vec<N,T> &centre, Vec<N,T> &x)
{
   Vec<N,T> r;
   for(;;){ // simple rejection sampling approach
      for(unsigned int i=0; i<N; ++i){
         r[i]=4*(random()/(T)2147483647-(T)0.5);
      }
      T r2=mag2(r); // magnitude squared of r
      if(r2>1 && r2<=4)
         break;
   }
   x=centre+radius*r;
}

// Translate a multidimensional array index into a one dimensional array index.
// (rounds towards zero if T is float or double)
template<unsigned int N, class T>
unsigned long int n_dimensional_array_index(const Vec<N,unsigned int> &dimensions,
                                            const Vec<N,T> &x)
{
   unsigned long int k=0;
   for(unsigned int i=N; i>0; --i){
      k*=dimensions[i-1];
      if(x[i-1]>=0){
         unsigned int j=(int)x[i-1];
         if(j>=dimensions[i-1]) j=dimensions[i-1]-1;
         k+=j;
      }
   }
   return k;
}
```

```
// Generate blue noise samples at least radius apart in N dimensions
// within box bounded by xmin and xmax. T should either be float or double.
template<unsigned int N, class T>
void blue_noise_sample(T radius, Vec<N,T> xmin, Vec<N,T> xmax,
                       std::vector<Vec<N,T> > &sample, // contains samples on return
                       int max_sample_attempts=30)
{
   // initialize acceleration grid (step 0)
   T grid_dx=T(0.99999)*radius/std::sqrt((T)N);
   Vec<N,unsigned int> dimensions;
   unsigned long int total_array_size=1;
   for(unsigned int i=0; i<N; ++i){
      dimensions[i]=(unsigned int)std::ceil((xmax[i]-xmin[i])/grid_dx);
      total_array_size*=dimensions[i];
   }
   std::vector<int> accel(total_array_size, -1); // -1 indicates no sample there;
                                                 // otherwise index of sample point

   // first sample (step 1)
   Vec<N,T> x;
   for(unsigned int i=0; i<N; ++i){
      x[i]=(xmax[i]-xmin[i])*(random()/(T)2147483647) + xmin[i];
   }
   sample.clear();
   sample.push_back(x);
   std::vector<unsigned int> active_list;
   active_list.push_back(0);
   unsigned int k=n_dimensional_array_index(dimensions, (x-xmin)/grid_dx);
   accel[k]=0;

   // generate the remaining samples (step 2)
   while(!active_list.empty()){
      unsigned int r=(unsigned int)((random()/(T)2147483648u)*active_list.size());
      int p=active_list[r];
      bool found_sample=false;
      Vec<N,unsigned int> j, jmin, jmax;
      for(int attempt=0; attempt<max_sample_attempts; ++attempt){
         sample_annulus(radius, sample[p], x);
         // check this sample is within bounds
         for(unsigned int i=0; i<N; ++i){
            if(x[i]<xmin[i] || x[i]>xmax[i])
               goto reject_sample;
         }
         // find range in acceleration grid that may contain interfering samples
         for(unsigned int i=0; i<N; ++i){
            int thismin=(int)((x[i]-radius-xmin[i])/grid_dx);
            if(thismin<0) thismin=0;
            else if(thismin>=(int)dimensions[i]) thismin=dimensions[i]-1;
            jmin[i]=(unsigned int)thismin;
            int thismax=(int)((x[i]+radius-xmin[i])/grid_dx);
            if(thismax<0) thismax=0;
            else if(thismax>=(int)dimensions[i]) thismax=dimensions[i]-1;
            jmax[i]=(unsigned int)thismax;
         }
         // loop over the selected grid cells (a little obfuscated since this is,
         // in effect, a nested N-dimensional loop)
         for(j=jmin;;){
            // check if there's a sample at j that's too close to x
            k=n_dimensional_array_index(dimensions, j);
            if(accel[k]>=0 && accel[k]!=p){ // if there is a sample different from p
               if(dist2(x, sample[accel[k]]) < radius*radius)
                  goto reject_sample; // proposed sample is too close to accel[k]
            }
            // move on to next j (N-dimensional nested version of ++j)
            for(unsigned int i=0; i<N; ++i){
               ++j[i];
               if(j[i]<=jmax[i]){
                  break;
               }else{
                  if(i==N-1) goto done_j_loop;
                  else j[i]=jmin[i]; // and try incrementing the next dimension along
               }
            }
         }
         done_j_loop:
         // if we made it here, we're good!
         found_sample=true;
         break;
         // if we goto here, x is too close to an existing sample
         reject_sample:
         ; // nothing to do except go to the next iteration in this loop
      }
      if(found_sample){
         unsigned int q=sample.size(); // the index of the new sample
         sample.push_back(x);
         active_list.push_back(q);
         k=n_dimensional_array_index(dimensions, (x-xmin)/grid_dx);
         accel[k]=(int)q;
      }else{
         // since couldn't find a sample on p's disk, remove p from the active list
         active_list[r]=active_list.back(); // overwrite with the last entry here
         active_list.pop_back(); // and delete the last entry
      }
   }
}

#endif
```