```c
/*
 * Author:  Pate Williams (c) 1997
 *
 * "Algorithm 2.6.3 (LLL Algorithm). Given
 * a basis b[1], b[2],..., b[n] of a lattice
 * (L, q), this algorithm transforms the vectors
 * b[i] so that when the algorithm terminates the
 * b[i] form an LLL-reduced basis." -Henri Cohen-
 * See "A Course in Computational Algebraic
 * Number Theory" by Henri Cohen pages 87-88.
 * Solution of the subset problem using LLL
 * reduction. See "Handbook of Applied
 * Cryptography" by Alfred J. Menezes, Paul
 * C. van Oorschot, and Scott A. Vanstone
 * pages 120 - 121.
 */

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void system_error(const char error_message[])
{
    fprintf(stderr, "%s", error_message);
    exit(1);
}

double** allocate_real_matrix(int lr, int ur, int lc, int uc)
{
    /* Allocates a real matrix of range [lr..ur][lc..uc]. */
    double** p = (double**)calloc((ur - lr + 1), sizeof(double*));
    int i;

    if (!p) system_error("Failure in allocate_real_matrix().");
    p -= lr;
    for (i = lr; i <= ur; i++) {
        p[i] = (double*)calloc((uc - lc + 1), sizeof(double));
        if (!p[i]) system_error("Failure in allocate_real_matrix().");
        p[i] -= lc;
    }
    return p;
}

double* allocate_real_vector(int l, int u)
{
    /* Allocates a real vector of range [l..u]. */
    double* p;

    p = (double*)calloc((u - l + 1), sizeof(double));
    if (!p) system_error("Failure in allocate_real_vector().");
    return p - l;
```

```
 53  }
 54
 55  void free_real_matrix(double** m, int lr, int ur, int lc)
 56  {
 57      /* Frees a real matrix of range [lr..ur][lc..uc]. */
 58      int i;
 59
 60      for (i = ur; i >= lr; i--)
 61          free((char*)(m[i] + lc));
 62      free((char*)(m + lr));
 63  }
 64
 65  void free_real_vector(double* v, int l)
 66  {
 67      /* Frees a real vector of range [l..u]. */
 68      free((char*)(v + l));
 69  }
 70
 71  double Scalar(long n, double* u, double* v)
 72  {
 73      /* Calculate the scalar product of two vectors [1..n] */
 74      double sum = 0.0;
 75      int i;
 76
 77      for (i = 1; i <= n; i++) sum += u[i] * v[i];
 78      return sum;
 79  }
 80
 81  void RED(int k, int l, int n,
 82      double** b, double** h,
 83      double** mu)
 84  {
 85      int i, q = (int)(0.5 + mu[k][l]);
 86
 87      if (fabs(mu[k][l]) > 0.5) {
 88          for (i = 1; i <= n; i++) {
 89              b[k][i] -= q * b[l][i];
 90              h[i][k] -= q * h[i][l];
 91          }
 92          mu[k][l] -= q;
 93          for (i = 1; i <= l - 1; i++) mu[k][i] -= q * mu[l][i];
 94      }
 95  }
 96
 97  void SWAP(int k, int k1, int kmax, int n,
 98      double m, double* B, double** b,
 99      double** bs, double** h, double** mu)
100  {
101      double C, t;
102      double* c = allocate_real_vector(1, n);
103      int i, j;
104
```

```
105     for (j = 1; j <= n; j++) {
106         t = b[k][j], b[k][j] = b[k1][j], b[k1][j] = t;
107         t = h[j][k], h[j][k] = h[j][k1], h[j][k1] = t;
108     }
109     if (k > 2)
110         for (j = 1; j <= k - 2; j++)
111             t = mu[k][j], mu[k][j] = mu[k1][j], mu[k1][j] = t;
112     C = B[k] + m * m * B[k1];
113     mu[k][k1] = m * B[k1] / C;
114     for (i = 1; i <= n; i++) c[i] = bs[k1][i];
115     for (i = 1; i <= n; i++)
116         bs[k1][i] = bs[k][i] + m * c[i];
117     for (i = 1; i <= n; i++)
118         bs[k][i] = -m * bs[k][i] + B[k] * c[i] / C;
119     B[k] *= B[k1] / C;
120     B[k1] = C;
121     for (i = k + 1; i <= kmax; i++) {
122         t = mu[i][k];
123         mu[i][k] = mu[i][k1] - m * t;
124         mu[i][k1] = t + mu[k][k1] * mu[i][k];
125     }
126     free_real_vector(c, 1);
127 }
128
129 void LLL(int n, double** b, double** h)
130 {
131     /* Lattice reduction algorithm. */
132     double m;
133     double* B = allocate_real_vector(1, n);
134     double** bs = allocate_real_matrix(1, n, 1, n);
135     double** mu = allocate_real_matrix(1, n, 1, n);
136     int i, j, k, k1, kmax = 1, l;
137
138     for (i = 1; i <= n; i++) bs[1][i] = b[1][i];
139     B[1] = Scalar(n, bs[1], bs[1]);
140     for (i = 1; i <= n; i++) {
141         for (j = 1; j <= n; j++)
142             h[i][j] = 0.0;
143         h[i][i] = 1.0;
144     }
145     for (k = 2; k <= n; k++) {
146         if (k <= kmax) goto L3;
147         kmax = k;
148         for (i = 1; i <= n; i++) bs[k][i] = b[k][i];
149         for (j = 1; j <= k - 1; j++) {
150             mu[k][j] = Scalar(n, b[k], bs[j]) / B[j];
151             for (i = 1; i <= n; i++)
152                 bs[k][i] -= mu[k][j] * bs[j][i];
153         }
154         B[k] = Scalar(n, bs[k], bs[k]);
155         if (B[k] == 0.0)
156             system_error("Failure in LLL.");
```

```
157     L3:
158         k1 = k - 1;
159         m = mu[k][k1];
160         RED(k, k1, n, b, h, mu);
161         if (B[k] < (0.75 - m * m) * B[k1]) {
162             SWAP(k, k1, kmax, n, m, B, b, bs, h, mu);
163             k = 2 > k1 ? 2 : k1;
164             goto L3;
165         }
166         for (l = k - 2; l >= 1; l--)
167             RED(k, l, n, b, h, mu);
168     }
169     free_real_matrix(bs, 1, n, 1);
170     free_real_matrix(mu, 1, n, 1);
171     free_real_vector(B, 1);
172 }
173
174 int SubsetSum(int n, double s, double* a, double* x)
175 {
176     int n1 = n + 1;
177     double** b = allocate_real_matrix(1, n1, 1, n1);
178     double** h = allocate_real_matrix(1, n1, 1, n1);
179     double sum;
180     int i, j, m = (int)ceil(sqrt(n) / 2.0);
181
182     for (i = 1; i <= n1; i++) {
183         if (i <= n) {
184             b[i][i] = 1.0;
185             b[i][n1] = m * a[i];
186         }
187         else {
188             for (j = 1; j <= n; j++) b[i][j] = 0.5;
189             b[i][n1] = m * s;
190         }
191     }
192     printf("the matrix to be reduced is:\n\n");
193     for (i = 1; i <= n1; i++) {
194         for (j = 1; j <= n1; j++)
195             printf("%6.2f ", b[i][j]);
196         printf("\n");
197     }
198     printf("\n");
199     LLL(n1, b, h);
200     printf("the reduced matrix is:\n\n");
201     for (i = 1; i <= n1; i++) {
202         for (j = 1; j <= n1; j++)
203             printf("%6.2f ", b[i][j]);
204         printf("\n");
205     }
206     printf("\n");
207     for (i = 1; i <= n1; i++) {
208         for (j = 1; j <= n; j++)
```

```
209             x[j] = b[i][j] + 0.5;
210         sum = 0.0;
211         for (j = 1; j <= n; j++)
212             sum += a[j] * x[j];
213         if (sum == s) {
214             free_real_matrix(b, 1, n1, 1);
215             free_real_matrix(h, 1, n1, 1);
216             return 1;
217         }
218         for (j = 1; j <= n; j++)
219             x[j] = -b[i][j] + 0.5;
220         sum = 0.0;
221         for (j = 1; j <= n; j++)
222             sum += a[j] * x[j];
223         if (sum == s) {
224             free_real_matrix(b, 1, n1, 1);
225             free_real_matrix(h, 1, n1, 1);
226             return 1;
227         }
228     }
229     free_real_matrix(b, 1, n1, 1);
230     free_real_matrix(h, 1, n1, 1);
231     return 0;
232 }
233
234 int main(void)
235 {
236     while (1) {
237         int answer = 0, i = 0, n = 8;
238         double* a = allocate_real_vector(1, n);
239         double* x = allocate_real_vector(1, n);
240         double s;
241
242         srand((unsigned int)time(NULL));
243         printf("\n");
244         for (i = 1; i <= n; i++)
245             a[i] = pow(2, i);
246         s =
247             a[1 + rand() % n] +
248             a[1 + rand() % n] +
249             a[1 + rand() % n];
250         if (SubsetSum(n, s, a, x)) {
251             printf("sum: %3.0lf\n\n", s);
252             printf("x[i]\ta[i]\n\n");
253             for (i = 1; i <= n; i++)
254                 printf("%1.0lf\t%3.0lf\n", x[i], a[i]);
255         }
256         else printf("subset sum has no solution\n");
257         free_real_vector(a, 1);
258         free_real_vector(x, 1);
259         printf_s("\nAnother problem (0 = no, 1 = yes) = ");
260         scanf_s("%d", &answer);
```

```
261             if (answer == 0)
262                 break;
263         }
264         return 0;
265     }
```