```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define EPS  pow(10.0,-8.0)   /* epsilon の設定 */
#define KMAX 100              /* 最大反復回数 */
#define N    3                /* N 元方程式 */

/* 行列の領域確保 */
double **dmatrix(int nr1, int nr2, int nl1, int nl2);
/* 行列の領域解放 */
void free_dmatrix(double **a, int nr1, int nr2, int nl1, int nl2);
/* ベクトル領域の確保 */
double *dvector(int i, int j);
/* 領域の解放 */
void free_dvector(double *a, int i);
/* 部分ピボット選択付きガウス消去法 */
double *gauss( double **a, double *b );
/* 1 ノルムの計算 a[m...n] */
double vector_norm1( double *a, int m, int n );
/* 関数の定義 */
double f(double x, double y, double z);
double g(double x, double y, double z);
double h(double x, double y, double z);
double f_x(double x, double y, double z);
double f_y(double x, double y, double z);
double f_z(double x, double y, double z);
double g_x(double x, double y, double z);
double g_y(double x, double y, double z);
double g_z(double x, double y, double z);
double h_x(double x, double y, double z);
double h_y(double x, double y, double z);
double h_z(double x, double y, double z);
/* Newton法 */
void newton2( double x, double y, double z );

int main(void)
{
  double x, y, z;
  printf("初期値 x0, y0, z0 を入力してください---> x0 y0 z0\n");
  scanf("%lf %lf %lf", &x, &y, &z);

  newton2( x, y, z ); /* Newton法 */

  return 0;
}

/* Newton法 */
void newton2( double x, double y, double z)
{
  int i, k=0; double *xk, *d, **J;
  d = dvector(1,N);      /* d[1...N] */
  xk = dvector(1,N);     /* xk[1...N] */
  J = dmatrix(1, N, 1, N); /* 行列 J[1...N][1...N] */

  xk[1]=x; xk[2]=y; xk[3]=z;

  do{
    /* 右辺ベクトルの作成 */
    d[1]=-f(xk[1],xk[2],xk[3]); d[2]=-g(xk[1],xk[2],xk[3]);
    d[3]=-h(xk[1],xk[2],xk[3]);
    /* ヤコビ行列の作成 */
    J[1][1] = f_x(xk[1],xk[2],xk[3]); J[1][2] = f_y(xk[1],xk[2],xk[3]);
    J[1][3] = f_z(xk[1],xk[2],xk[3]); J[2][1] = g_x(xk[1],xk[2],xk[3]);
    J[2][2] = g_y(xk[1],xk[2],xk[3]); J[2][3] = g_z(xk[1],xk[2],xk[3]);
    J[3][1] = h_x(xk[1],xk[2],xk[3]); J[3][2] = h_y(xk[1],xk[2],xk[3]);
    J[3][3] = h_z(xk[1],xk[2],xk[3]);
    d = gauss( J, d );  /* 連立一次方程式を解く */
    for ( i=1; i <= N; i++) xk[i] += d[i];
    k ++;
  }while( vector_norm1(d,1,N) > EPS && k < KMAX);

  if ( k == KMAX )
  {
    printf("答えが見つかりませんでした\n");
  }
  else
  {
    printf("答えはx=%f, y=%f, z=%f です\n", xk[1], xk[2], xk[3]);
  }

  /* 領域の解放 */
  free_dmatrix( J, 1, N, 1, N ); free_dvector( d, 1 ); free_dvector( xk, 1 );
}

double f(double x, double y, double z)
{
  return( -1.0 - x + x*x - y + y*y + x*z );
}

double g(double x, double y, double z)
{
  return( -4.0 + 3.0*x*x + x*x*x - 2.0*z + 2.0*y*z - z*z + z*z*z );
}

double h(double x, double y, double z)
{
  return( -2.0*x - 3.0*y + 3.0*x*y - 4.0*z + 2.0*x*z + 4.0*y*z );
}

double f_x(double x, double y, double z)
{
  return( -1.0 + 2.0*x + z );
}

double f_y(double x, double y, double z)
{
  return( -1.0 + 2.0*y );
}

double f_z(double x, double y, double z)
{
  return( x );
}

double g_x(double x, double y, double z)
{
  return( 6.0*x + 3.0*x*x );
}

double g_y(double x, double y, double z)
{
  return( 2.0*z );
}

double g_z(double x, double y, double z)
{
  return( -2.0 + 2.0*y - 2.0*z + 3.0*z*z );
}

double h_x(double x, double y, double z)
{
  return( -2.0 + 3.0*y + 2.0*z);
}

double h_y(double x, double y, double z)
{
  return( -3.0 + 3.0*x + 4.0*z );
}

double h_z(double x, double y, double z)
{
  return( -4.0 + 2.0*x + 4.0*y );
}

/* 部分ピボット選択付きガウス消去法 */
double *gauss( double **a, double *b )
{
  int i, j, k, ip;
  double alpha, tmp;
  double amax, eps=pow(2.0, -50.0); /* eps = 2^{-50}とする */

  for( k = 1; k <= N-1; k++)
  {
    /* ピボットの選択 */
    amax = fabs(a[k][k]); ip = k;
    for( i = k+1; i <= N; i++)
    {
      if ( fabs(a[i][k]) > amax )
      {
        amax = fabs(a[i][k]); ip = i;
      }
    }
    /* 正則性の判定 */
    if ( amax < eps ) printf("入力した行列は正則ではない!!\n");
    /* 行交換 */
    if ( ip != k)
    {
      for( j = k; j <= N; j++)
      {
        tmp = a[k][j]; a[k][j]=a[ip][j]; a[ip][j]=tmp;
      }
        tmp = b[k] ; b[k]=b[ip]; b[ip]=tmp;
    }
      /* 前進消去 */
    for( i = k+1; i <= N; i++)
    {
      alpha = - a[i][k]/a[k][k];
      for( j = k+1; j <= N; j++)
      {
        a[i][j] = a[i][j] + alpha * a[k][j];
      }
      b[i] = b[i] + alpha * b[k];
    }
  }

  /* 後退代入 */
  b[N] = b[N]/a[N][N];
  for( k = N-1; k >= 1; k--)
  {
    tmp = b[k];
    for( j = k+1; j <= N; j++)
    {
      tmp = tmp - a[k][j] * b[j];
    }
    b[k] = tmp/a[k][k];
  }

  return b;
}

double **dmatrix(int nr1, int nr2, int nl1, int nl2)
{
        int i, nrow, ncol;
        double **a;

        nrow = nr2 - nr1 + 1; /* 行の数 */
        ncol = nl2 - nl1 + 1; /* 列の数 */

        /* 行の確保 */
        if((a = malloc(nrow * sizeof(double *)) )== NULL){
                printf("メモリが確保できません (行列 a)\n");
                exit(1);
        }
        a = a - nr1; /* 行をずらす */

        /* 列の確保 */
        for(i = nr1; i <= nr2; i++) a[i] = malloc(ncol * sizeof(double));
        for(i = nr1; i <= nr2; i++) a[i] = a[i] - nl1;                    /* 列をずらす */

        return (a);
}

void free_dmatrix(double **a, int nr1, int nr2, int nl1, int nl2)
{
        int i;

        /* メモリの解放 */
        for(i = nr1; i <= nr2; i++) free((void *)(a[i] + nl1));
        free((void *)(a + nr1));
}

double *dvector(int i, int j)
{
        double *a;

        if((a = malloc( ((j - i + 1) * sizeof(double))) ) == NULL)
        {
                printf("メモリが確保できません (from dvector) \n");
                        exit(1);
        }

        return (a - i);
}

void free_dvector(double *a, int i)
{
        free( (void *)(a + i) ); /* (void *) 型へのキャストが必要 */
}

/* 1 ノルムの計算 a[m...n] */
double vector_norm1(double *a, int m, int n)
{
  int i;
  double norm = 0.0;
  for (i = m; i <= n; i++)
  {
    norm += fabs(a[i]);
  }
  return norm;
}
```