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

/* 関数の定義 */
double func1(double x);
double func2(double x);
/* ベクトル領域の確保 */
double *dvector(int i, int j);
/* ベクトル領域の解放 */
void free_dvector(double *a, int i);
/* ロンバーグ法 */
double romberg( double a, double b, int N, double eps, double (*f)(double) );

int main(void)
{
  int N = 6;
  double eps = pow(10.0, -10.0);

  printf("2.0/(x*x) を [1,2] で積分します．最大反復回数は%dです\n", N);
  printf("結果は%20.15fです\n",romberg(1.0, 2.0, N, eps, func1) );

  printf("4.0/(1+x*x) を [0,1] で積分します．最大反復回数は%dです\n", N);
  printf("結果は%20.15fです\n",romberg(0.0, 1.0, N, eps, func2) );

  return 0;
}

/* ロンバーグ法 */
double romberg( double a, double b, int N, double eps, double (*f)(double) )
{
  double S, h, *t, f0, f1;
  int j, k, m, n;

  t = dvector( 0, N );
  h = b - a;
  f0 = (*f)(a); f1 = (*f)(b);
  t[0] = h*( f0 + f1 )/2.0;

  /* ロンバーグ法 */
  for( n = 1; n <= N; n++)
  {
    h = h / 2.0;  S = 0.0;
    for( j = 1; j <= (int)(pow(2.0,n)-1.0); j++ ) S += (*f)( a + j*h );
    t[n] = h*( f0 + 2.0*S + f1 )/2.0;
    if( fabs(t[n]-t[n-1]) < eps ) return t[n];
    k = n;
    for( m = 1; m <= n; m++)
    {
      k = k-1;
      t[k] = ( pow(4.0,m)*t[k+1]-t[k] )/( pow(4.0,m)-1.0 );
      if( k >= 1 &&  fabs(t[k]-t[k-1]) < eps  ) return t[k];
    }
  }

  /* ベクトル領域の解放 */
  free_dvector(t,0);

  return t[N];  /* 収束しなければ t[N] を積分値とする */
}

/* 関数の定義 */
double func1(double x)
{
  return( 2.0/(x*x) );
}

double func2(double x)
{
  return( 4.0 / (1.0+x*x) );
}


double *dvector(int i, int j)  /* a[i]～a[j] の領域を確保 */
{
        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 */
}
```