This file contains a set of rules used by the "make" command.  
This makefile $(MAKEFILE) tells "make" how the executable $(COMMAND) 
should be generated from the source files $(SRCS) and the header files 
$(HDRS) via the object files $(OBJ); type the command:  
"make -f make_program" where make_program should be replaced by the name of the makefile.  
To remove the OBJ files; type the command:  
"make -f make_program clean"  
To create a zip archive with name $(COMMAND).zip containing this 
makefile and the SRCS and HDRS files, type the command:  
"make -f make_program.zip"  
The name of this makefile  
MAKEFILE= make_qags_test  
The command you type to run the program (executable name)  
COMMAND= qags_test  
The name of this makefile  
MAKEFILE= make_qags_test  
The command you type to run the program (executable name)  
COMMAND= qags_test  
Here are the C++ (or whatever) source files to be compiled, with \'s as 
continuation lines.  If you get a "missing separator" error pointing 
to a line here, make sure that each \ has NO spaces following it.  
SRCS= \  
qags_test.cpp  

# Header files (if any) here  

# Commands and options for compiling  

# instructions to compile and link -- allow for different dependencies  

$(COMMAND): $(OBJ) $(HDRS) $(MAKEFILE)  
$(CC) -o $(COMMAND) $(OBJ) $(LDFLAGS) $(LIBS)  
qags_test.o : qags_test.cpp $(MAKEFILE)  
$(CC) -c $(CFLAGS) $(WARNFLAGS) -c qags_test.cpp -o qags_test.o  

# Additional tasks  

clean:  
		rm -f $(OBJ)  
zip:  
		zip -r $(COMMAND).zip $(MAKEFILE) $(SRCS) $(HDRS)  

# End of makefile  

// file: qags_test.cpp  
// C++ Program to test the qags automatic integrator from  
// the gsl numerical library.  
// Programmer: Dick Furnstahl furnstahl.1@osu.edu  
// Revision history:  
// 12/26/03 original C++ version, modified from C version  
// Notes:  
// * Example taken from the GNU Scientific Library Reference Manual  
// Edition 1.1, for GSL Version 1.1 9 January 2002  
// URL: gsl/ref/gsl-ref_23.html#SEC364  
// * Compile and link with:  
//   g++ -Werror -ansi -Wall -o qags_test qags_test.cpp -lgsl -lgslcblas  
// * gsl routines have built-in  
//   extern "C" {  
//    <header stuff>  
// }  
// so they can be called from C++ programs without modification  

// The following details are taken from the GSL documentation  
// Each algorithm computes an approximation to a definite integral of 
// the form,  
// I = \int_a^b w(x) f(x) dx  
// where w(x) is a weight function (for general integrands w(x)=1). The 
// user provides absolute and relative error bounds (epsabs, epsrel) 
// which specify the following accuracy requirement,  
// |RESULT - I| <= max(epsabs, epsrel |I|)  
// where |RESULT| is the numerical approximation obtained by the 
// algorithm. The algorithms attempt to estimate the absolute error  
// ABSERR = |RESULT - I| in such a way that the following inequality 
// holds,  
// |RESULT - I| <= ABSERR <= max(epsabs, epsrel |I|)  
// The routines will fail to converge if the error bounds are too 
// stringent, but always return the best approximation obtained up to 
// that stage.  

// QAGS adaptive integration with singularities  

// Function: int gsl_integration_qags (const gsl_function * f,  
// double a, double b,  
// double epsabs, double epsrel,  
// size_t limit,  
// gsl_integration_workspace * workspace,  
// double *result,  
// double *abserr)  
//  
// This function applies the Gauss-Kronrod 21-point integration rule  
// adaptively until an estimate of the integral of f over (a,b) is  
// achieved within the desired absolute and relative error limits,  
// epsabs and epsrel. The results are extrapolated using the  
// epsilon-algorithm, which accelerates the convergence of the integral  
// in the presence of discontinuities and integrable singularities. The  
// function returns the final approximation from the extrapolation,  
// result, and an estimate of the absolute error, abserr. The
subintervals and their results are stored in the memory provided by
workspace. The maximum number of subintervals is given by limit,
which may not exceed the allocated size of the workspace.

The integrator QAGS will handle a large class of definite integrals.
For example, consider the following integral, which has a
algebraic-logarithmic singularity at the origin,
\[ \int_0^1 x^{-1/2} \log(x) \, dx = -4 \]
The program below computes this integral to a relative accuracy bound
of 1e-8.

The results below show that the desired accuracy is achieved after 8
subdivisions.

result          = -3.999999999999973799
exact result    = -4.000000000000000000
estimated error =  0.000000000000499600
actual error    =  0.000000000000026201
intervals =  8

In fact, the extrapolation procedure used by QAGS produces an
accuracy of many more digits. The error estimate returned
by the extrapolation procedure is larger than the actual error,
giving a margin of safety of one order of magnitude.

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cmath>
using namespace std;

struct My_function
{
    double function (double x, void *params);
    double* params;
};
My_function My_function;

my_integrand (double x, void *params)
{
    // Mathematica form:  Log[alpha*x]/Sqrt[x]
    // The next line recovers alpha from the passed params pointer
    double alpha = *(double*) params;
    return log (alpha * x) / sqrt (x);
}
```