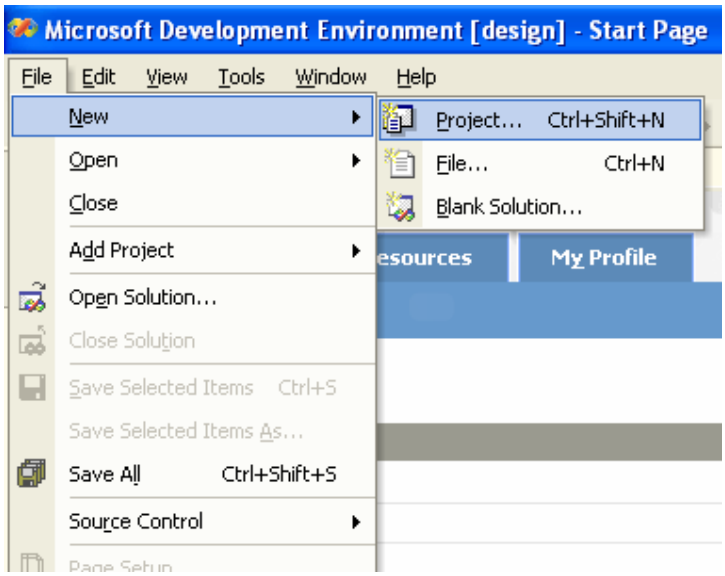
Universidade Federal de Santa Maria
Prof. Cesar Tadeu Pozzer
11/12/2007

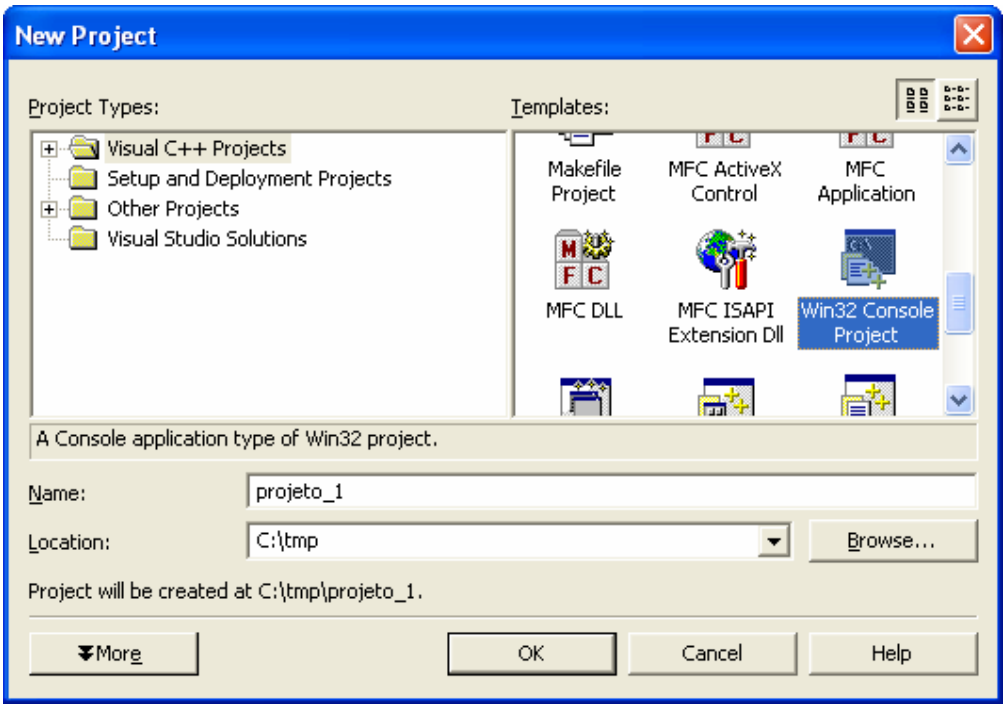
Índice:

1. Utilização do Compilador Microsoft Visual Studio .NET 2003

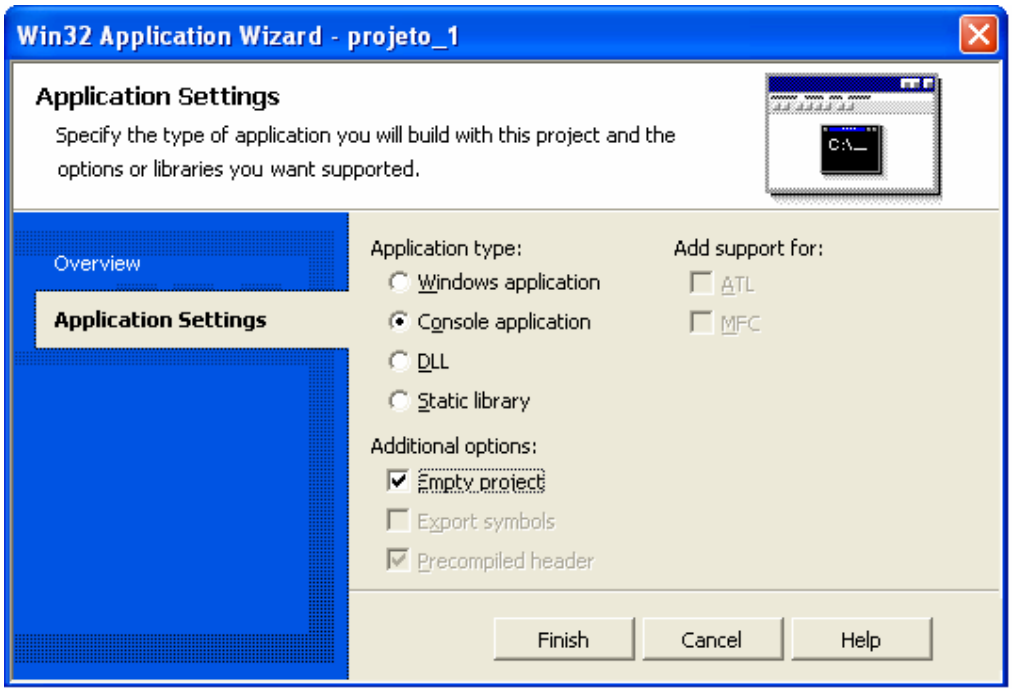
1 – Crie um novo projeto



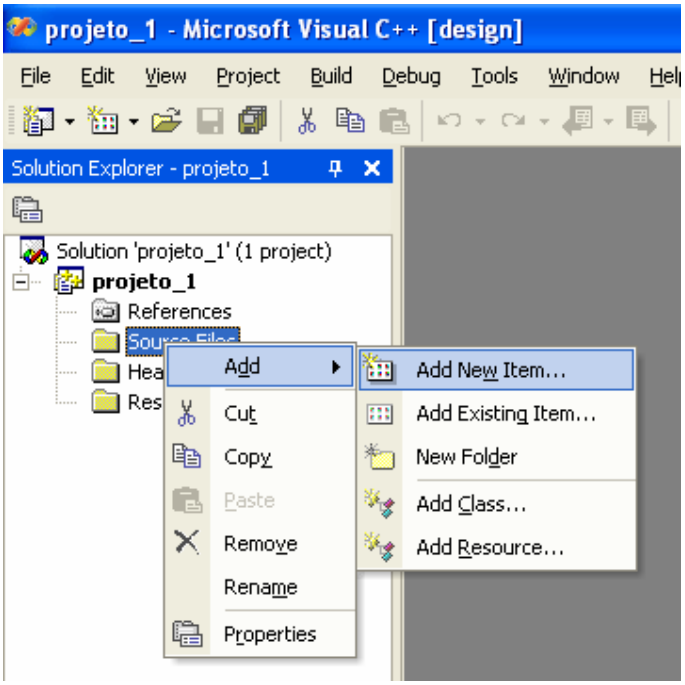
2 – Selecione *Win32 Console Project*. Especifique um nome e um diretório



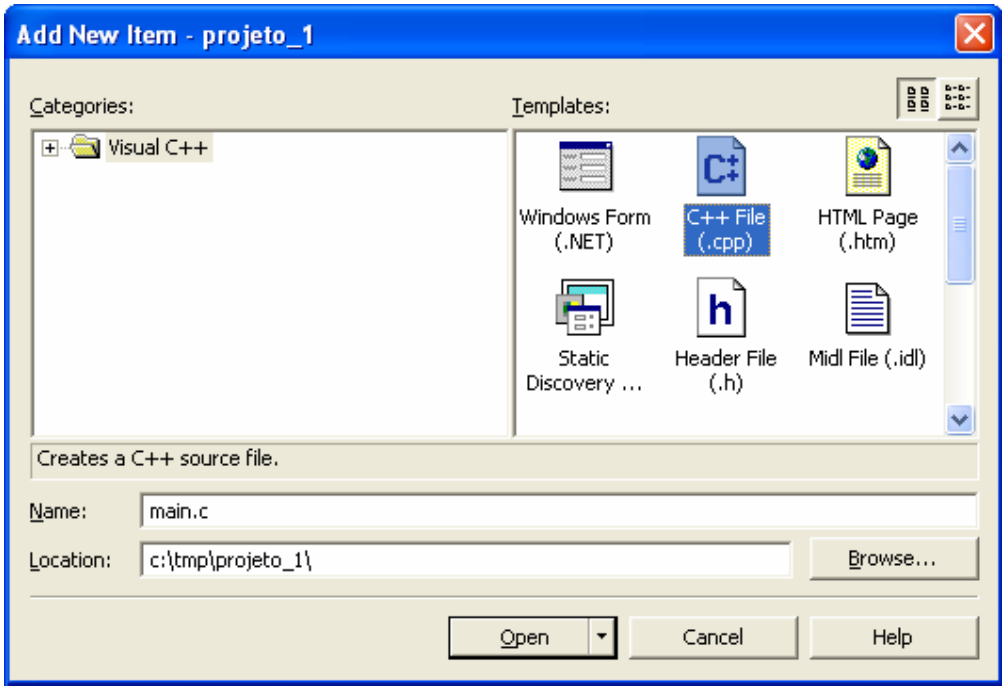
3 – Selecione *Empty project* na opção *Application Settings*



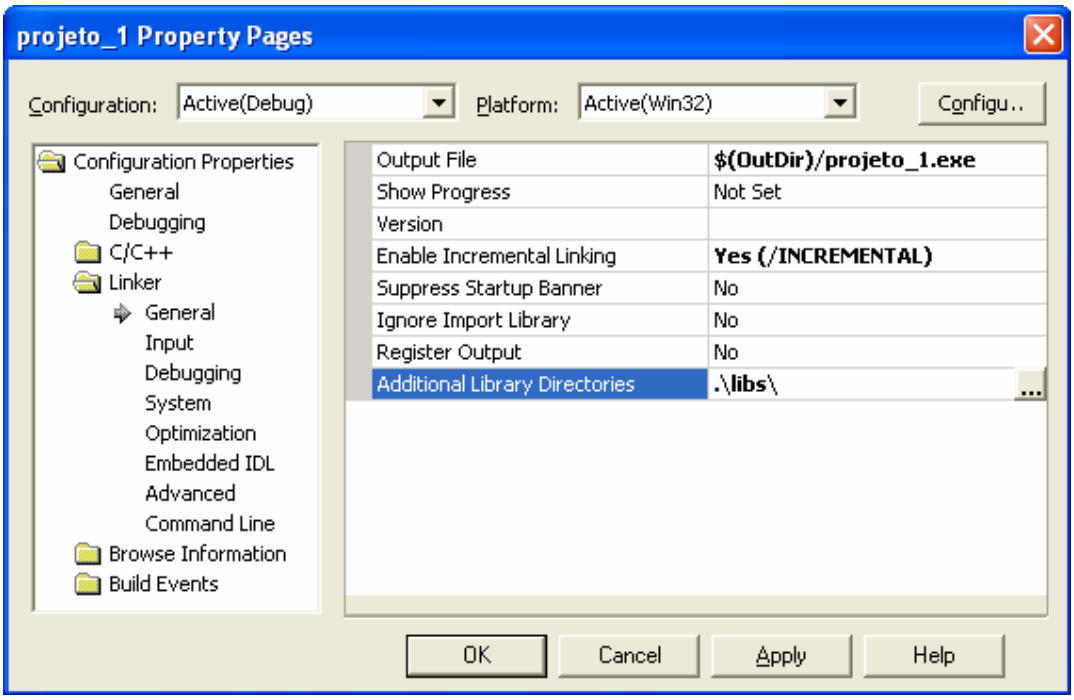
4 – Após adicione os arquivos de código (novo ou existente)

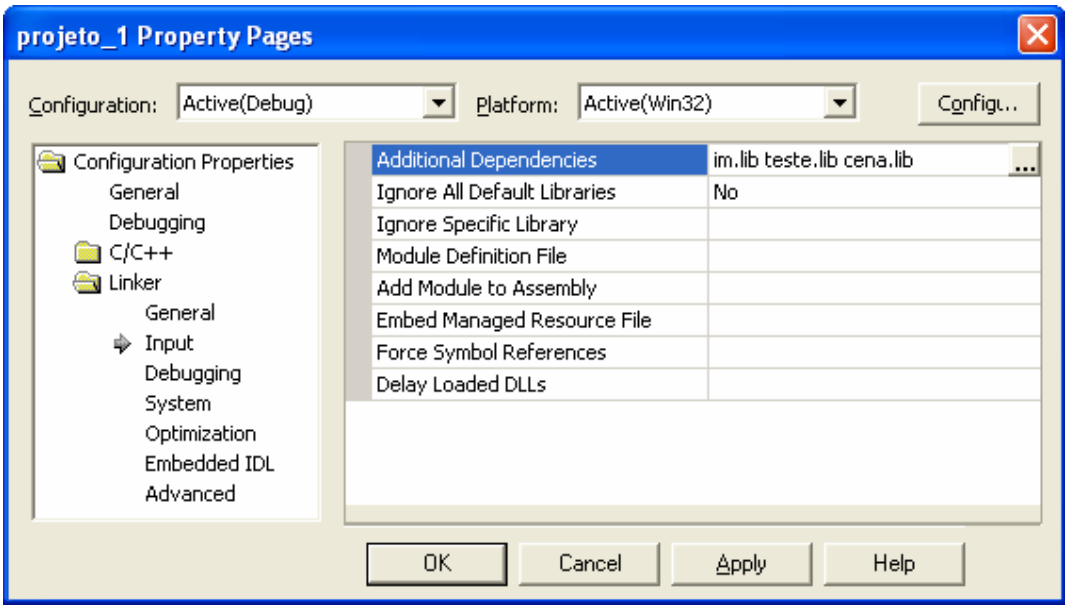


4.1 – Se for um novo, escolha C++ File e dê um nome ao arquivo



5 – Adição de bibliotecas (libs) - Menu Project/properties





Comandos

Ctrl+Shift+b → Compilar
F5 → Executar
F9 → Adição/remoção de breakpoint

Também possui uma excelente ferramenta de depuração (Menu Debug).

2. Utilização dos Compiladores gcc, g++

O compilador gcc é usado para compilar código C e o g++ para codigo C/C++. Recomenda-se o uso do g++, visto que compila os dois tipos de código. Ambos compiladores têm os seguintes argumentos de linha de comando:

```
D:\gcc --help
Usage: gcc [options] file...
Options:
  -pass-exit-codes         Exit with highest error code from a phase
  --help                   Display this information
  --target-help            Display target specific command line options
  (Use '-v --help' to display command line options of sub-processes)
  -dumpspecs               Display all of the built in spec strings
  -dumpversion             Display the version of the compiler
  -dumpmachine             Display the compiler's target processor
  -print-search-dirs       Display the directories in the compiler's search path

  -print-libgcc-file-name  Display the name of the compiler's companion library
  -print-file-name=<lib>   Display the full path to library <lib>
  -print-prog-name=<prog>  Display the full path to compiler component <prog>
  -print-multi-directory   Display the root directory for versions of libgcc
  -print-multi-lib         Display the mapping between command line options and
                           multiple library search directories
  -print-multi-os-directory Display the relative path to OS libraries
  -Wa,<options>            Pass comma-separated <options> on to the assembler
  -Wp,<options>            Pass comma-separated <options> on to the preprocessor

  -Wl,<options>            Pass comma-separated <options> on to the linker
  -Xassembler <arg>        Pass <arg> on to the assembler
  -Xpreprocessor <arg>     Pass <arg> on to the preprocessor
  -Xlinker <arg>           Pass <arg> on to the linker
  -save-temps              Do not delete intermediate files
  -pipe                    Use pipes rather than intermediate files
  -time                    Time the execution of each subprocess
```

```
  -specs=<file>            Override built-in specs with the contents of <file>
  -std=<standard>          Assume that the input sources are for <standard>
  -B <directory>           Add <directory> to the compiler's search paths
  -b <machine>             Run gcc for target <machine>, if installed
  -V <version>             Run gcc version number <version>, if installed
  -v                       Display the programs invoked by the compiler
  -###                     Like -v but options quoted and commands not executed
  -E                       Preprocess only; do not compile, assemble or link
  -S                       Compile only; do not assemble or link
  -c                       Compile and assemble, but do not link
  -o <file>                Place the output into <file>
  -x <language>            Specify the language of the following input files
                           Permissible languages include: c c++ assembler none
                           'none' means revert to the default behavior of
                           guessing the language based on the file's extension

Options starting with -g, -f, -m, -O, -W, or --param are automatically
 passed on to the various sub-processes invoked by gcc.  In order to pass
 other options on to these processes the -W<letter> options must be used.
```

Para compilar código com estes compiladores, o mais aconselhado é criar arquivos Makefile. Outra solução, para Windows, é criar um arquivo .bat, como mostrado no seguinte exemplo.

```
@echo off
cls

echo.
echo Vai iniciar compilacao
g++ -O0 -g3 -Wall -c  *.cpp

echo.
echo Vai iniciar copia de arquivos
move *.o debug

echo.
echo Vai iniciar linkagem
g++ -o app.exe  debug/*.o -L"d:/app/Dev-Cpp/lib" –L"." –lmpc32 -lglut32 -lglu32 -lopengl32

pause
```

Para Unix, o Makefile pode ter a seguinte estrutura. Observe que deve haver uma tabulação antes de cada comando. O caractere `#` significa comentário.

```
all:
	#gera todos os arquivos objetos
	gcc -O0 -g3 -Wall -c *.c

	#ar r libmpc.a font.o mpc.o

	gcc -oapp app.o callbacks.o main.o -L"." -lmpc -lglut -lGLU -lGL -lm
clear:
	#apaga todos os arquivos objetos gerados
	rm *.a
	rm *.o
	rm app
```

Para crirar um arquivo lib, deve-se utilizar o programa ar. Inicialmente deve-se criar os arquivos .obj para então uní-los em um único arquivo .a (padrão do gcc e g++). Eis um exemplo de um arquivo .bat para criar uma lib para ser usada em projetos desenvolvidos em C++/C, que façam uso do g++:

```
g++ -O0 -c -Wall __*.cpp
move __*.o debug
ar -r  libmpc32.a ./debug/__*.o
rem ranlib libmpc32.a
```

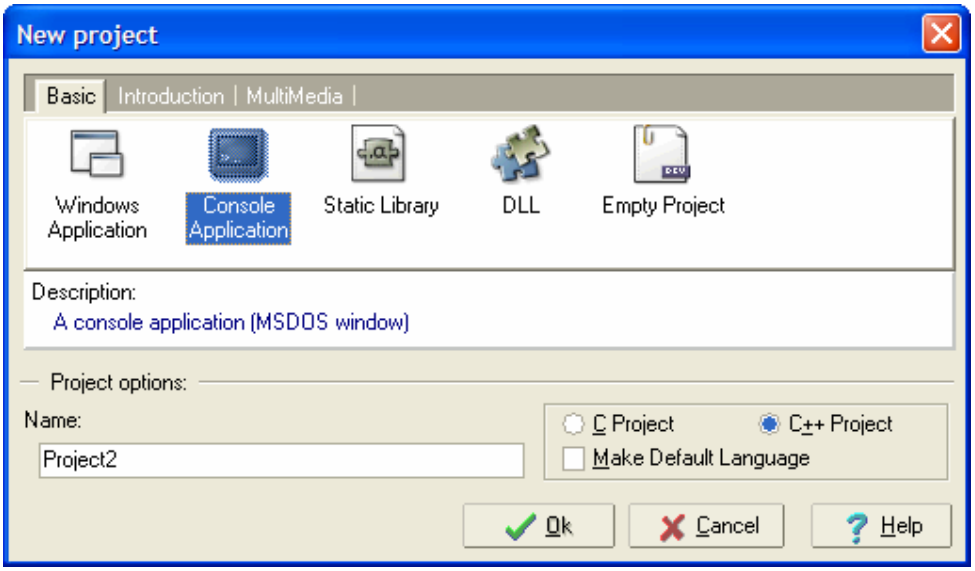
Regras Importantes:

- Para gerar uma lib para o gcc, deve-se compilar com o gcc. Os arquivos devem ter extensao .c
- Para gerar uma lib para o g++, deve-se compilar com o g++. Os arquivos podem ser c ou c++

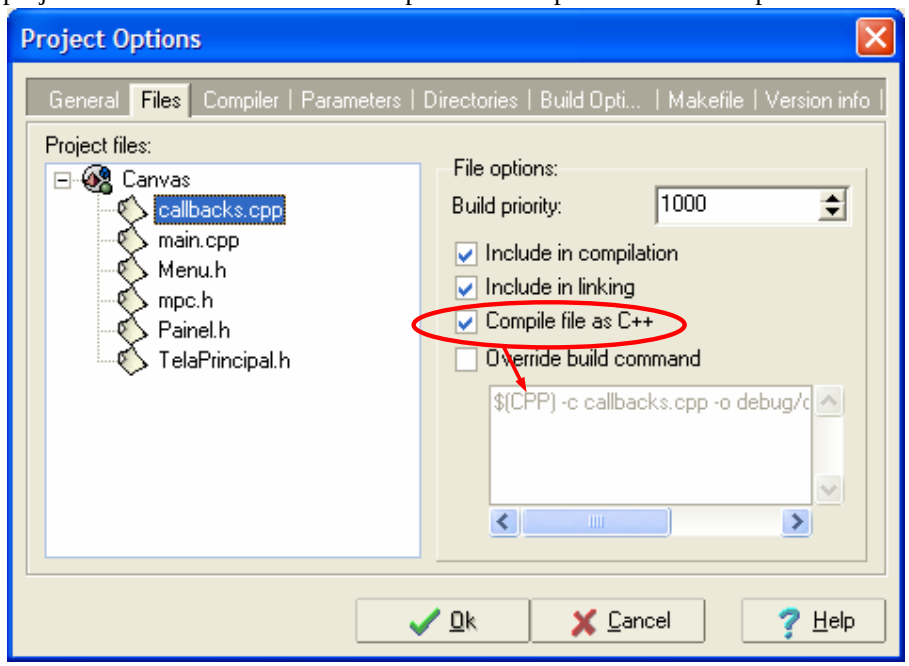
3. Utilização do Ambiente DEV-C++

O DEV-C++ é um ambiente de desenvolvimento integrado livre que utiliza os compiladores de licença GNU para compilar para os sistemas operacionais MS Windows ou MS-DOS (como por exemplo o gcc ou g++, apresentados anteriormente).

Ao criar um projeto (5 opções), deve-se selecionar o tipo de linguagem a ser utilizada.



Pode-se selecionar o compilador utilizado para compilar cada arquivo fonte. O compilador usado para linkagem é dado pelo tipo do projeto selecionado. Deve-se cuidar para nao compilar com um compilador e linkar com outro.



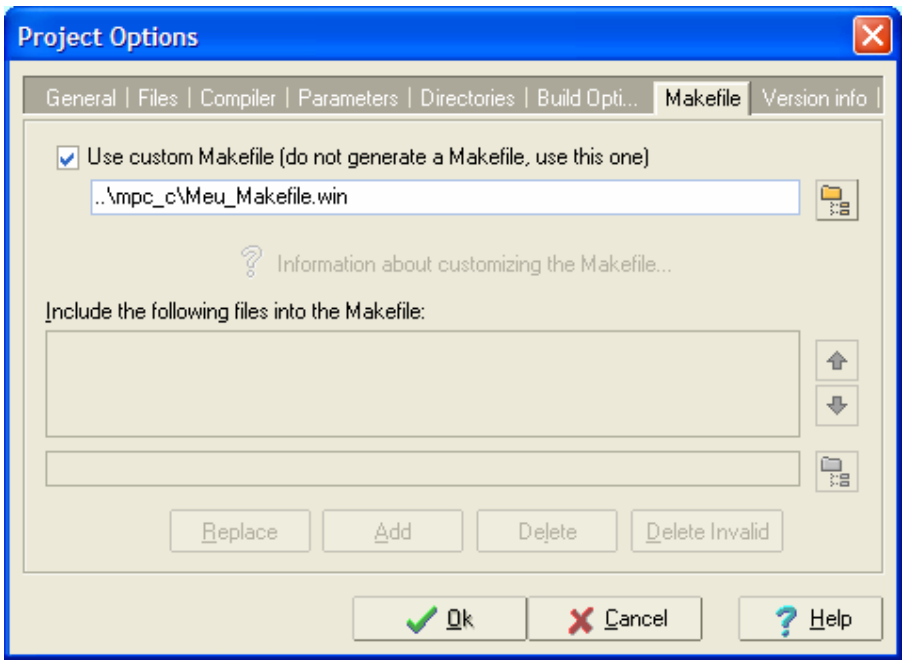
Antes de compilar um projeto com o DEV-C++, o arquivo Makefile.win será gerado. Por este arquivo pode-se estudar como gerar arquivos Makefile para qualquer plataforma. No seguinte exemplo, pode-se observar que está

sendo utilizado o compilador g++ para compilar e linkar o código. Se fosse um projeto com arquivos .c, seria utilizado o compilador gcc.

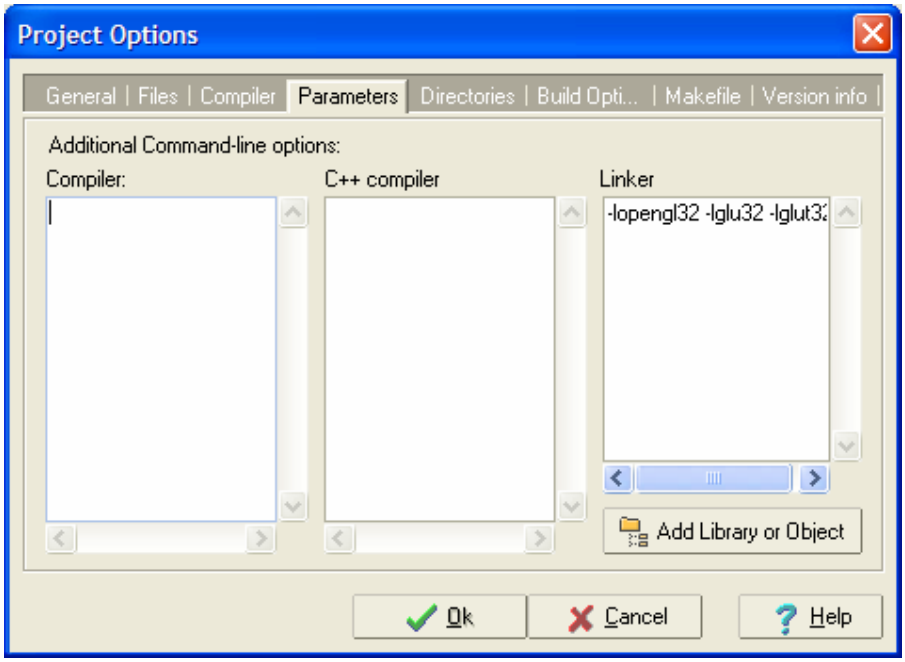
```
# Project: Canvas
# Makefile created by Dev-C++ 4.9.9.2

CPP  = g++.exe
CC   = gcc.exe
WINDRES = windres.exe
RES  =
OBJ  = debug/callbacks.o debug/main.o $(RES)
LINKOBJ  = debug/callbacks.o debug/main.o $(RES)
LIBS =  -L"d:/app/Dev-Cpp/lib" -L"." –lmpc32 -lopengl32 -lglu32 -lglut32  -fmessage-length=0
INCS =  -I"d:/app/Dev-Cpp/include"
CXXINCS     =          -I"d:/app/Dev-Cpp/lib/gcc/mingw32/3.4.2/include"          -I"d:/app/Dev-
Cpp/include/c++/3.4.2/backward"   -I"d:/app/Dev-Cpp/include/c++/3.4.2/mingw32"   -I"d:/app/Dev-
Cpp/include/c++/3.4.2"  -I"d:/app/Dev-Cpp/include"
BIN  = debug/app.exe
CXXFLAGS = $(CXXINCS)   -fmessage-length=0
CFLAGS = $(INCS)   -fmessage-length=0
RM = rm -f

.PHONY: all all-before all-after clean clean-custom

all: all-before debug/app.exe all-after


clean: clean-custom
       ${RM} $(OBJ) $(BIN)

$(BIN): $(OBJ)
       $(CPP) $(LINKOBJ) -o "debug\app.exe" $(LIBS)

debug/callbacks.o: callbacks.cpp
       $(CPP) -c callbacks.cpp -o debug/callbacks.o $(CXXFLAGS)

debug/main.o: main.cpp
       $(CPP) -c main.cpp -o debug/main.o $(CXXFLAGS)
```

Usa o g++ para linkar
Usa o g++ para compilar os arquivos

Pode-se também passar um arquivo Makefile personalizado para o dev que será usado para compilar o projeto.



Pode-se também definir libs a serem linkacas ao projeto. Estes argumentos podem ser visualizados no arquivo Makefile.win gerado antes da compilação.



O dev-c++ também tem recursos de depuração de código por meio da inclusão de breakpoints e watches. Para isso deve-se habilitar a geração de informação de depuração.