Module 6 – Managing C Programs

Laboratory Exercises

Objectives
The aim of this lab mainly to become familiar with the tools available for managing the compilation of multi-file C programs, e.g. the make utility and makefiles.

Activities
Depending on the compiler you are using it may have a software project manager utility built into an integrated development environment. On the Melbourne campuses of RMIT, we use a Unix environment and the make utility.

The following is a briefly introduction to makefiles and the make utility. If you are studying off campus, then you should find out how to use the project manager facilities within your compiler for achieving the same end.

Typical make utilities these days are quite sophisticated in what they can do. We don't aim to present examples of all or even most of what you can do with make.

In addition to the materials provided, here is a simple example of how Makefiles work. You can Save these files in a temporary directory and try them out if you like.

[readme]

```
Lab exercises -- Make utility
1. Copy files to your work space
2. Try following this session transcsript:

%ls
complex.c main.c readme
complex.h makefile
makefile.annotated

%make
gcc -Wall -ansi -pedantic -c main.c
gcc -Wall -ansi -pedantic -c complex.c
gcc -o complex-main main.o complex.o

                  Make automatically compiles main.c
                  complex.c and then links them together

%ls complex-main
complex-main*

                  Here is out executable

%./complex-main
0+0i
1+2i
1+1i
2+4i              And here it is running
3+6i
4+8i
5+10i
```

```
%touch main.c     This updates the time-stamp on the source
                  file, ''simulating'' changes being made
                  to the file.

%make
gcc -Wall -ansi -pedantic -c main.c
gcc -o complex-main main.o complex.o

                  Now when we call make, it only recompiles
                  main.c (the changed file) and relinks the
                  updated main.o with the existing complex.o

%touch complex.c  Now ''simulating'' an update to complex.c

%make
gcc -Wall -ansi -pedantic -c complex.c
gcc -o complex-main main.o complex.o

                  make only recompiles those files that need
                  recompiling. The dependency lists in the
                  makefile tell make which files, if changed,
                  will result in other files needing to be
                  recompiled.

%touch complex.h  Now ''simulating'' an update to complex.h

%make
gcc -Wall -ansi -pedantic -c main.c
gcc -Wall -ansi -pedantic -c complex.c
gcc -o complex-main main.o complex.o

                  Since both main.c and complex.c depended
                  on complex.h, make recompiled them both
                  before relinked, and creating an executable.

The End.
```

[makefile]

```
#
# This is a simple makefile for the complex data type example
# When you learn more about make and makefiles you will discover
# that it is possible to write considerably more 'concise' makefiles.
#
# A fully annotated version of the makefile can be found in
# 'makefile.annotated'
#

all: main.o complex.o
      gcc -o complex-main main.o complex.o

main.o: main.c complex.h
      gcc -Wall -ansi -pedantic -c main.c

complex.o: complex.c complex.h
      gcc -Wall -ansi -pedantic -c complex.c
#
# The End
#
```

[makefile.annotated]

```
#
# This is a simple makefile for the complex data type example.
# When you learn more about make and makefiles you will discover
# that it is possible to write considerable more 'concise' makefiles.
#

all: main.o complex.o
      gcc -o complex-main main.o complex.o

# the line above indicates how the entire multi-file program is
# to be compiled and linked. We'd read the line above as: In
# order to build all we need an up to date main.o and complex.o
# if they are up to date, then use the gcc command to link the
# object files together with the standard library to create an
# executable.
# note also that a 'tab' is used to indent the line containing
# the shell commands (such as the call to gcc) required.

main.o: main.c complex.h
      gcc -Wall -ansi -pedantic -c main.c

# to determine if main.o is up to date the date-stamp on the
# main.o file is compared with the date-stamps on the files
# main.c and complex.h (these are the files that main.o depends
# on -- if they have changed, then main.c needs to be recompiled
# to produced an up to date main.o) If recompilation is required
# the call to gcc with -c is used to compile to the .o level
# rather than trying to create an executable -- which of course
# would not be possible, and main.c and complex.h by themselves
# do not make a complete program.

complex.o: complex.c complex.h
      gcc -Wall -ansi -pedantic -c complex.c

# and finally, complex.o depends on complex.c and complex.h if
# either of these files have changed more recently than complex.o
# then the make utility will use the gcc command to recompile
# complex.c to the .o level
```

[main.c]

```
#include <stdio.h>
#include <stdlib.h>
#include "complex.h"
#define CSIZE 4

int main(void)
{
   Complex c1, c2, Carr[CSIZE];
   int i;

   InitComplex(&c1);
   DisplayComplex(&c1);
   printf("\n");

   SetComplex(&c2, 1, 2);
   DisplayComplex(&c2);
   printf("\n");
```

```
   AddToComplex(&c1, &c2);
   DisplayComplex(&c1);
   printf("\n");

   for(i=0; i < CSIZE; i++)
   {
      SetComplex(&Carr[i], i+1, i+2);
      AddToComplex(&Carr[i], &c1);
   }

   for(i=0; i < CSIZE; i++)
   {
      DisplayComplex(&Carr[i]);
      printf("\n");
   }

   return EXIT_SUCCESS;
}
```

[complex.c]

```
/*
* complex.c
*/
#include <stdio.h>
#include "complex.h"

void InitComplex(Complex * cp)
{
   cp->real = 0;
   cp->img = 0;
}

void SetComplex(Complex * cp, int real, int img)
{
   cp->real = real;
   cp->img = img;
}

void AddToComplex(Complex * dst, Complex * src)
{
   dst->real += src->real;
   dst->img += dst->real;
}

void DisplayComplex(Complex * cp)
{
   printf("%d+%d%c", cp->real, cp->img, 'i');
}
```

[complex.h]

```
/*
* complex.h
*/
typedef struct complex
{
   int real;
```

```
   int img;
} Complex;

void InitComplex(Complex *);
void SetComplex(Complex *, int, int);
void AddToComplex(Complex *, Complex *);
void DisplayComplex(Complex *);
```