

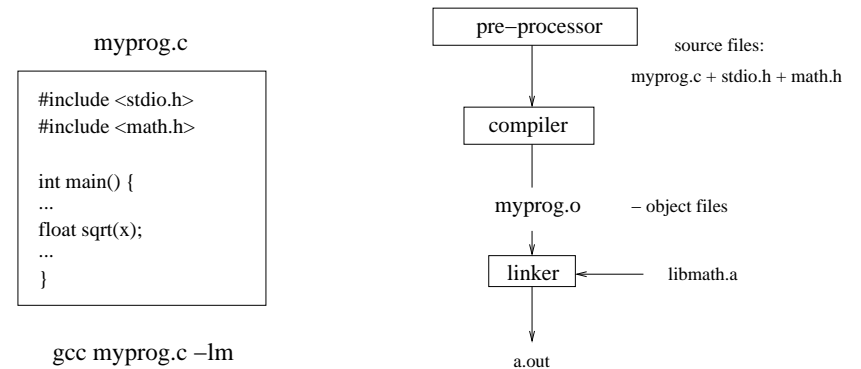
# Computer Programming: Skills & Concepts (CP1)

## Libraries and separate compilation

15th November, 2010

CP1-24 – slide 1 – 15th November, 2010

## The stages of compilation



CP1-24 – slide 3 – 15th November, 2010

## Compiling a C program

Is actually a three stage process...

- The 'C pre-processor' adds all the `#include` files and expands the `#define` statements.
- The 'C compiler' compiles the *source* files into *object* files.
- The 'Linker' links the object files with libraries into an *executable* that you can run.

```
gcc myprog.c -lm
```

CP1-24 – slide 2 – 15th November, 2010

## The pre-processor

```
#include <stdio.h>          /* These header files get added
#include <stdlib.h>         * directly into the program code
#include <math.h>           * by the pre-processor. */

#define SIZE 20             /* Pre-processor will put 20 everywhere SIZE
                           * appears in code (except inside quotes) */

int main() {
    int p, q;
    float x[SIZE], y[SIZE]; /* will get changed to x[20], y[20] */
    ....
    for (p=0; p < SIZE; p++) /* will get changed to have p < 20 */
    ...
}
```

CP1-24 – slide 4 – 15th November, 2010

## To do compilation only

To compile into an object file, and not link.

```
gcc -c myprog.c
```

A file is produced called myprog.o

To link object files:

```
gcc myprog.o -lm
```

executable file a.out is produced.

To produce a different name of executable:

```
gcc -o name myprog.o -lm
```

(To run just the pre-processor) **Not** usual to do this manually.

```
cpp myprog.c
```

CP1-24 – slide 5 – 15th November, 2010

## Functions in separate files

*A program prog1.c consists of its main function, with a single function func1(). Also the math library is used.*

Place function in a separate file func1.c. Compile both:

```
gcc -c prog1.c
gcc -c func1.c
```

Then link together into a.out

```
gcc func1.o prog1.o -lm
```

Why?

- function can easily be re-used elsewhere.
- No need to re-compile func1 if it hasn't changed (good for large files)!

CP1-24 – slide 7 – 15th November, 2010

## Some more compiler flags

Optimization:

-O: Compile the program for performance.

-O2/-O3: Aggressive optimisations. At the expense of compile time and memory usage.

```
gcc -O3 myprog.c -lm
```

De-bugging:

-g flag adds information to enable a debugger tool to work.

```
gcc -g myprog.c -lm
```

CP1-24 – slide 6 – 15th November, 2010

## A simple program

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
float func1(float y);

int main() {
    float x,y;
    y = 0.5;
    x = func1(y);
    printf("x was %f\n",x) ;
    return EXIT_SUCCESS;
}

float func1(float y) {
    float x;
    x = sin(y)*cos(y);
    return x;
}
```

CP1-24 – slide 8 – 15th November, 2010

## Split into 2 files

Make two files `prog1.c` and `func1.c`.

- ▶ `prog1.c` contains just the main body of original program;
- ▶ `func1.c` contains just the function `func1`, plus some `#include` statements;
- ▶ Must include the following at top of `prog1.c`:  
`extern float func1(float y);`

*CP1-24 – slide 9 – 15th November, 2010*

## Header file option

Make three files `prog1.c`, `func1.h`, and `func1.c`.

- ▶ `prog1.c` contains the main body of original program:
  - + also contains `#include "func1.h"`
  - but no longer has the extern definition for `func1`.
- ▶ `func1.c` contains just the function `func1`, plus some `#include` statements;
- ▶ `func1.h` is just the following declaration:  
`extern float func1(float y);`

*CP1-24 – slide 11 – 15th November, 2010*

## extern declaration

Indicates to the compiler that a variable or function is to be found in another file.  
*Will be resolved later by the linker.*

Only applies at global scope.  
*i.e only to global variables and functions.*

Where to put these extern function declarations?

- ▶ Can be messy with many functions in 1 file.
- ▶ We can use the pre-processor.

*CP1-24 – slide 10 – 15th November, 2010*

## Header files

Files containing function declarations are usually called *header files*.

Convention:

- `function1.h` contains function headers.
- `function1.c` contains the functions themselves.

To add functions to your program:

- `#include "function1.h"`
- `gcc function1.o myprog.o`

Might be many functions per file.

*CP1-24 – slide 12 – 15th November, 2010*

## Compilation (summary)

- ▶ Compilation is a three stage process.
- ▶ Can compile into object files separately.
- ▶ Multiple object files can be linked into a single program.
- ▶ Need to declare functions as 'extern'.
- ▶ Use of header files.

CP1-24 – slide 13 – 15th November, 2010

## A simple Makefile

```
func1.o:    func1.c func1.h project.h
gcc -c func1.c

func2.o:    func2.c func2.h project.h
gcc -c func2.c

program:    func1.o func2.o program.c project.h
gcc -o program func1.o func2.o program.c -lm

all:  program
```

- project.h has constants for the whole project. All files depend on it.
- func1.o depends on func1.c and func1.h.
- program depends on func1 and func2.

CP1-24 – slide 15 – 15th November, 2010

## make and Makefiles

make is a tool for automating the building of programs.

A Makefile consists of a number of rules.

One rule consists of:

- **target:** a target is a file(s) to be built.
- **dependencies:** a list of files that the target relies on.
- **commands:** how to build the target.

make <target\_file> will build the file based on the rules.

CP1-24 – slide 14 – 15th November, 2010

## Makefiles

- ▶ Very flexible, powerful - and complicated!
- ▶ MACROS - constants that can be defined
- ▶ Special macros: \$@ is the name of the file to be made:  

```
CFLAGS= -c
printenv: printenv.c
gcc $(CFLAGS) $@.c -o $@
```
- ▶ Makefiles can call any command, and can be used for a wide variety of tasks.

CP1-24 – slide 16 – 15th November, 2010