Introduction to GCC and Header Files

Clare So

socm@mcmaster.ca

SE2AA4 Tutorial. McMaster University. January 17, 2007.

Compiling C Programs

- We have the following program `hello.c`

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

int main() {
   printf("Hi mom!  I can program in C!");
   return 0;
}
```


Compiling C Programs

- We can compile `hello.c` using `gcc`

```
[socm@birkhoff ~]$ gcc hello.c
[socm@birkhoff ~]$ a.out
Hi mom!  I can program in C!
[socm@birkhoff ~]$
```

- `-o` option can be used to specify the name of the executable

```
[socm@birkhoff ~]$ gcc hello.c -o hello
[socm@birkhoff ~]$ hello
Hi mom!  I can program in C!
[socm@birkhoff ~]$
```


How to Deal with Multiple Source Files?

- We have the following C procedure in `foo.c`

```
#include <math.h>

double mysterious_procedure(double r) {
   return pow(sin(r),2)+pow(cos(r),2);
}
```

How to Deal with Multiple Source Files?

- `mysterious_procedure` is invoked in `main.c`

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

#define PI 3.14

double mysterious_procedure(double r);

int main() {
   double result = mysterious_procedure(PI);
   printf("The mysterious result is %f\n",result);
   return 0;
}
```


Compiling Multiple Source Files

- We compile the source code and run the program

```
[socm@birkhoff ~]$ gcc -c foo.c
[socm@birkhoff ~]$ gcc -c main.c
[socm@birkhoff ~]$ gcc main.o foo.o -o main -lm
[socm@birkhoff ~]$ main
The mysterious result is 1.000000
[socm@birkhoff ~]$
```

- Important: `-lm` option is needed for programs using `math.h`

Stages of Compilation

- Four stages are involved to turn the source code into executable files

  Pre-process – strip out comments, expand `#include` etc
  Compile – parse the source code and produce assembly code
  Assemble – produce object files (aka `.o` files) from the assembly code
  Link – build an executable by linking together object files

- Note that
  - `-c` tells `gcc` to stop at assemble stage
  - `-o` tells `gcc` to link the object files

Why Use Header Files?

- You want to share your module but don't want other people to see its actual implementation
- Just give them the object files? No!
  - How can other people know the prototypes of the procedures?
- Solution: Give them the object files and the header files only

Contents of Header Files

- `#include` directive
- Macro definition
  (Example: `define`)
- Type definition
  (Example: `Cigar` struct from last tutorial)
- Prototypes of the procedures
  (Example: `double mysterious_procedure(double r)`)

Using Header Files for Our Example

- foo.h

```
#include <math.h>

double mysterious_procedure(double r);
```

- foo.c

```
#include "foo.h"

double mysterious_procedure(double r){
   return pow(sin(r),2)+pow(cos(r),2);
}
```


Using Header Files for Our Example

- main.c

```
#include <stdlib.h>
#include <stdio.h>
#include "foo.h"

#define PI 3.14

int main() {
   double result = mysterious_procedure(PI);
   printf("The mysterious result is %f\n",result);
   return 0;
}
```