



C Minicourse – Day 3



Outline

- Header files
- Makefiles
- Input / Output functions
- `const` / `static` keywords
- `gcc` basics
- Common errors (segfaults, etc.)
- Tips for compiling code



Header Files

- Use sparingly, only when necessary
- Use as an interface to a particular .c file
- Contain
 - `#define` statements
 - `extern` function prototypes
 - `typedef` statements



Example Header File

```
#ifndef __HELLO_WORLD__  
#define __HELLO_WORLD__  
  
#define TIMES_TO_SAY_HELLO 10  
  
extern void printHello(void);  
  
#endif // __HELLO_WORLD__
```



Header File Do's

- Always enclose header files with `#ifndef`, `#define`, and `#endif` preprocessor commands
- Make sure function prototypes match functions
- Double-check semi-colons; they are necessary after function prototypes



Header File Don'ts

- Don't `#include` files if you can avoid it; you should use `#include` in `.c` files instead
- Don't write functions inside your header file unless you have a good reason
- Don't use a single header file for multiple `.c` files



Makefiles

- Makefiles save you typing
- Can reduce compile time by only compiling changed code
- Can be extremely complicated
- Are hard to “do right” sometimes
- But existing Makefiles are easy to edit
- And it’s easy to write quick-n-dirty ones



Sample Makefile

```
EXEC_NAME = helloWorld
```

```
all: main.o
```

```
    gcc -g -o $(EXEC_NAME) main.o
```

```
main.o: main.c
```

```
    gcc -g -c main.c -o main.o
```

```
clean:
```

```
    rm -rf main.o $(EXEC_NAME)
```



More Makefiles

- Rule names like “all” and “clean” and “main.o”
- “all” and “clean” are special – most rule names should be filenames
- Dependencies are listed after the colon following the rule name
- Operations are listed underneath; must be preceded by a tab



Input / Output functions

- `printf` – takes a format string and an “unlimited” number of variables as arguments; prints to `stdout`
- `scanf` – similar to `printf`, but it takes pointers to variables and fills them with values received from `stdin`
- `read / write` – lower level calls for getting and printing strings



printf/scanf example

```
int main(int argc, char **argv) {  
    int a = 8;  
    char b = 'b';  
    int c;  
  
    printf("The int a is %d\n", a);  
    printf("a is %d. b is %c\n", a, b);  
    scanf("Enter c:\n%d", c);  
}
```



`const/static` keywords

- `const` tells compiler that data should not be changed.
- `static` limits the usage of a variable or function to a particular `.c` file; also can be used with variables
 - `static` functions cannot be declared `extern` in header files
 - `static` variables retain their values between function calls



gcc basics

- gcc is C compiler
 - -Wall turns on all warnings
 - -g adds debugging info
 - -o specifies output name
 - -c just builds a .o (object) file

```
gcc -Wall hello.c -o helloWorld
```



Common errors

- Segfaults
 - Indicates a problem with memory access; dereferencing a `NULL` pointer can cause this
- Linker errors
 - Usually a problem with a Makefile and/or `gcc` commands
- Multiple definitions of functions
 - Can sometimes be caused by incorrect `#ifndef`, `#define` statements in header files



Tips for compiling code

- If you get several compiling error message, fix the first error first (seems intuitive, no?)... Later errors can be caused by the first one.
- Don't just look at the line number that `gcc` reports. Look above the line for missing semicolons or other problems.