

Warning: These notes are not complete, it is a Skelton that will be modified/add-to in the class. If you want to us them for studying, either attend the class or get the completed notes from someone who did

# CSE2301

## Functions and Compiler Directives

These slides are based on slies by Prof. Wolfgang Stuerzlinger

# Functions

- Function is a small program that may receives some data, perform some computations, and may return a value.

```
Return_type function_name(arguments declaration)
{
    declaration
    statements
}
```

- In C, call is by value (example).

## Declaration

- Before the use, functions must be declared.

```
int mysum(int a, int b);
```

Function may return a value using `return`

Retruning a value from a function that returns void is an error

Not returning a value from a function that returns a value is unpredictable.

## Scope

- Variables do exist within their block.
- For functions, all variables are created at the time the function is called, and vanishes after the function returns (automatic variables).
- If a variable is declared before main, then it is visible to all functions in the file, but could be overridden by declaring a variable by the same name in a function.

## Scope

- Usually, an external variable must be declared `extern` in any function that uses it
- If the function is in the same file as the variable declaration, no need for `extern`
- If a variable is declared `static` outside of `main`, it is not visible to other files (only in this file).

## Scope

- `static` could be used also with functions.
- If a variable in a function is declared `static`, the variable does not vanish after the function returns, it stays in the memory so the next call to the function will find the old value.
- `static` variables are initialized to 0.

## Preprocessor

- Preprocessor processes the file before compilation.

- It handles `#define` and `#include`

```
#define ON 1
```

```
#define OFF 0
```

```
#define square(x) ((x)*(x))
```

## Preprocessor

- Conditional inclusion

```
#if !defined(HDR) #ifdef HDR
```

```
#define HDR
```

```
//include the file here
```

```
#endif
```

```
#undef x undefines x
```

## Preprocessor

- Formal parameters are not replaced within quoted strings.
- `##` means concatenate
- If the parameter name is preceded by a `#` in the replacement text, the combination will be expanded into a quoted string with the parameter replaced by the actual argument

```
#define dprint(expr) printf(#expr " = %g \n", expr)
```

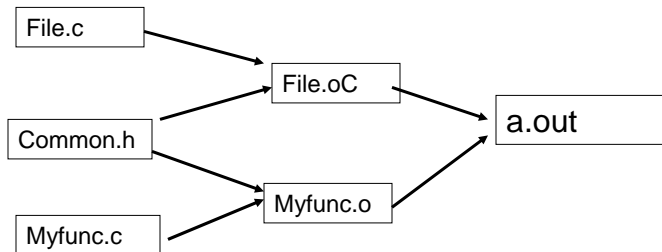
## Preprocessor

- `#ifdef` can also be used for conditional compilation.

## Compilation

- `cc file.c` produces `a.out`
- Compiling a C program
  - Converting the `.c` file into assembly `.s`
  - Compiling the assembly into a machine code (object code) `.o`
  - Linking the `.o` file to the code library and is named `a.out`
- What happens when you have several files

## Compilation

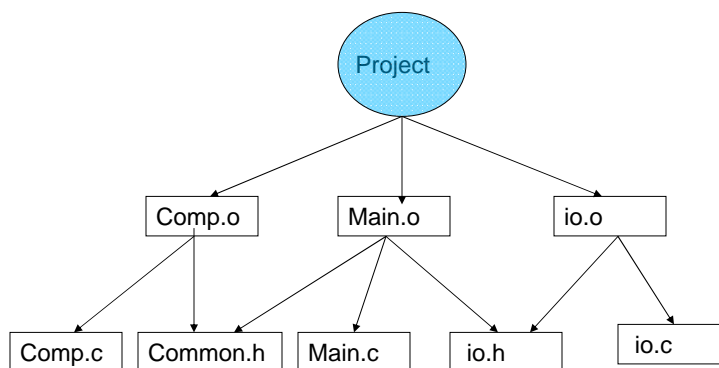


- We can produce the `.o` file using
- `cc -c File.c`

## Compilation

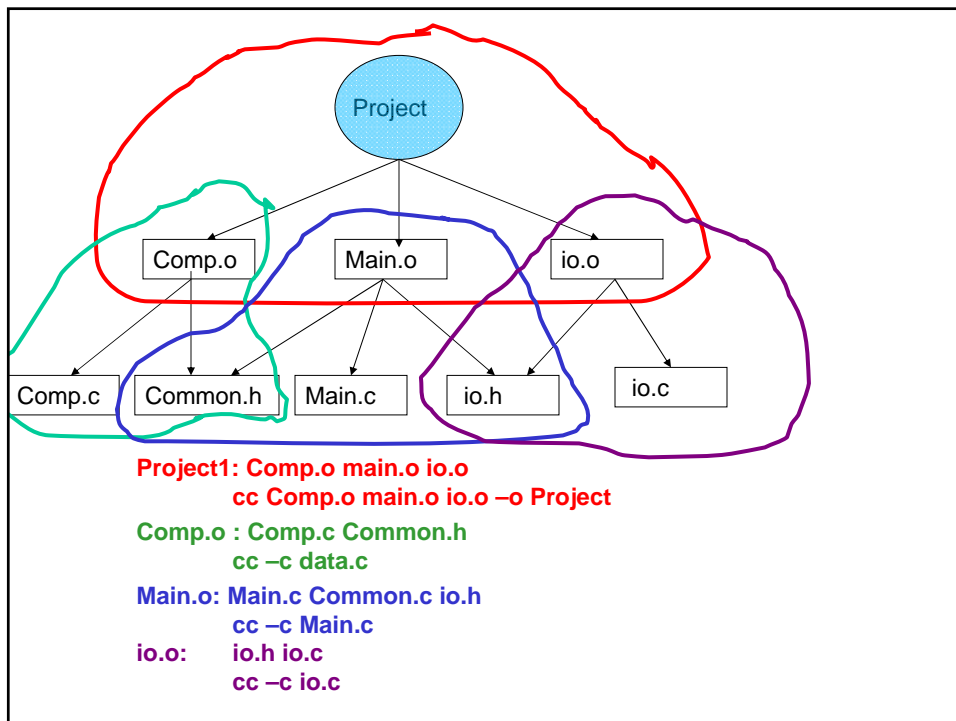
- Usually, your program will be divided into several files.
- Be careful with global variables and functions name.
- Every file will be compiled into its .o
- Finally, all the .o files can be compiled to produce a.out
- `Cc File.o Myfunc.o`

## Dependency Graph



# Compilation

- If we changed one file, there is no need to recompile all the files.
- Make files are used to manage that
- The name of the file is makefile or Makefile





## Makefile

- Each dependency is handled as  
Target: source files  
    Command (preceded by a tab)
- To compile just type **make**

## Macros in Makefile

```
OBJECTS = Comp.o Main.o io.o
Project: $(OBJECTS)
    gcc $(OBJECTS) -o Project
Comp.o: Comp.c Common.h
    gcc -c Comp.c
Main.o: Main.c Common.h io.h
    gcc -c Main.c
io.o: io.c io.h
    gcc -c io.c
```

## Standard C Library

- **assert.h**
  - **ctype.h**
  - **errno.h**
  - **math.h**
  - **limits.h**
  - **signal.h**
  - **stdarg.h**
  - **stddef.h**
  - **stdio.h**
  - **stdlib.h**
  - **string.h**
  - **time.h**
- assertions
  - character mappings
  - error numbers
  - math functions
  - metrics for ints
  - signal handling
  - variable length arg lists
  - standard definitions
  - standard I/O
  - standard library functions
  - string functions
  - date/type functions

## Standard C Libraries

- • Utility functions **stdlib.h**
  - **atof, atoi, rand, qsort, getenv,**
  - **calloc, malloc, free, abort, exit**
- • String handling **string.h**
  - **strcmp, strncmp, strcpy, strncpy, strcat,**
  - **strncat, strchr, strlen, memcpy, memcmp**
- • Character classifications **ctype.h**
  - **isdigit, isalpha, isspace, isupper, islower**
- • Mathematical functions **math.h**
  - **sin, cos, tan, ceil, floor, exp, log, sqrt**