# Makefiles !

**main.c**

```c
#include "printfuncs.h"

int main()
{
  int answer=42;
  // call a function
  // in another file

  printFuncInt(answer);

  return 0;
}
```

**printfuncs.c**

```c
#include <stdio.h>
#include "printfuncs.h"

void printFuncInt(int i)
{
  printf("Int: %d\n",i);
  return;
}
```

**printfuncs.h**

```c
/*
 * function definitions
for printfuncs
 *
*/

void printFuncInt(int);
```

```
gcc main.c printfuncs.c -o doit
```

**main.c**

```c
#include "printfuncs.h"

int main()
{
  int answer=42;
  // call a function
  // in another file

  printFuncInt(answer);
  printFuncStr(“Yes!”);

  return 0;
}
```

**printfuncs.c**

```c
#include <stdio.h>
#include "printfuncs.h"

void printFuncInt(int i)
{
  printf("Int: %d\n",i);
  return;
}

void printFuncStr(char *s)
{
  printf(“Str: %s\n”,s);
  return;
}
```

**printfuncs.h**

```c
/*
 * function definitions
for printfuncs
 *
*/

void printFuncInt(int);
void printFuncStr(char*);
```

```
gcc main.c printfuncs.c -o doit
```

**main.c**

```
#include "printfuncs.h"

int main()
{
  int answer=42;
  // call a function
  // in another file

  printFuncInt(answer);
  printFuncStr(“Yes!”);

  return 0;
}
```

**printfuncs.c**

```
#include <stdio.h>
#include "printfuncs.h"

void printFuncInt(int i)
{
  printf("Int: %d\n",i);
  return;
}

void printFuncStr(char *s)
{
  printf(“Str: %s\n”,s);
  return;
}
```

**printfuncs.h**

```
/*
 * function definitions
for printfuncs
 *
*/

void printFuncInt(int);
void printFuncStr(char*);
```

```
gcc -Wall -pedantic -std=c11 -ggdb main.c printfuncs.c -o doit
```

**main.c**

```
#include "printfuncs.h"

int main()
{
  int answer=42;
  // call a function
  // in another file

  printFuncInt(answer);
  printFuncStr(“Yes!”);

  return 0;
}
```

**printfuncs.c**

```
#include <stdio.h>
#include "printfuncs.h"

void printFuncInt(int i)
{
  printf("Int: %d\n",i);
  return;
}

void printFuncStr(char *s)
{
  printf(“Str: \”%s\”\n”,s);
  return;
}
```

**printfuncs.h**

```
/*
 * function definitions
for printfuncs
 *
*/

void printFuncInt(int);
void printFuncStr(char*);
```

```
gcc -Wall -pedantic -std=c11 -ggdb main.c printfuncs.c -o doit
```

**main.c**

```c
#include "printfuncs.h"

int main()
{
  int answer=TheAnswer;
  // call a function
  // in another file

  printFuncInt(answer);
  printFuncStr(“Yes!”);

  return 0;
}
```

**printfuncs.c**

```c
#include <stdio.h>
#include "printfuncs.h"

void printFuncInt(int i)
{
  printf("Int: %d\n",i);
  return;
}

void printFuncStr(char *s)
{
  printf(“Str: \”%s\”\n”,s);
  return;
}
```

**printfuncs.h**

```c
/*
 * function definitions
for printfuncs
 *
*/

void printFuncInt(int);
void printFuncStr(char*);

/* constants */
const int TheAnswer=42;
```

```
gcc -Wall -pedantic -std=c11 -ggdb main.c printfuncs.c -o doit
```

Adapted from https://web.stanford.edu/class/archive/cs/cs107/cs107.1174/guide_make.html

```
#
# A simple makefile for managing build of project composed of C source files.
#

# It is likely that default C compiler is already gcc, but explicitly
# set, just to be sure
CC = gcc

# The CFLAGS variable sets compile flags for gcc:
#  -ggdb        compile with debug information
#  -Wall        give verbose compiler warnings
#  -pedantic    issue all the warnings demanded by strict ISO C and ISO C++
#  -std=c11     use the C11 (2011) standard language definition
CFLAGS = -Wall -pedantic -std=c11 -ggdb

# The LDFLAGS variable sets flags for linker
#  -lm   says to link in libm (the math library)
LDFLAGS = -lm

# In this section, you list the files that are part of the project.
# If you add/change names of source files, here is where you
# edit the Makefile.
SOURCES = main.c printFuncs.c printFuncs.h
OBJECTS = $(SOURCES:.c=.o)
TARGET = doit

# The first target defined in the makefile is the one
# used when make is invoked with no argument. Given the definitions
# above, this Makefile file will build the one named TARGET and
# assume that it depends on all the named OBJECTS files.

$(TARGET) : $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LDFLAGS)

# Phony means not a "real" target, it doesn't build anything
# The phony target "clean" is used to remove all compiled object files.

.PHONY: clean

clean:
    @rm -f $(TARGET) $(OBJECTS) core
```

Note: if make knows you need a .o file AND it can see .c files with the same filename, make will *assume* you need it to run the gcc compiler to a .o file from the .c of the same name.

For example:
you need  fruit.o and there is a file named fruit.c.