Advanced Software Techniques

Writing C/C++ code without Visual Studio

So far …

Always

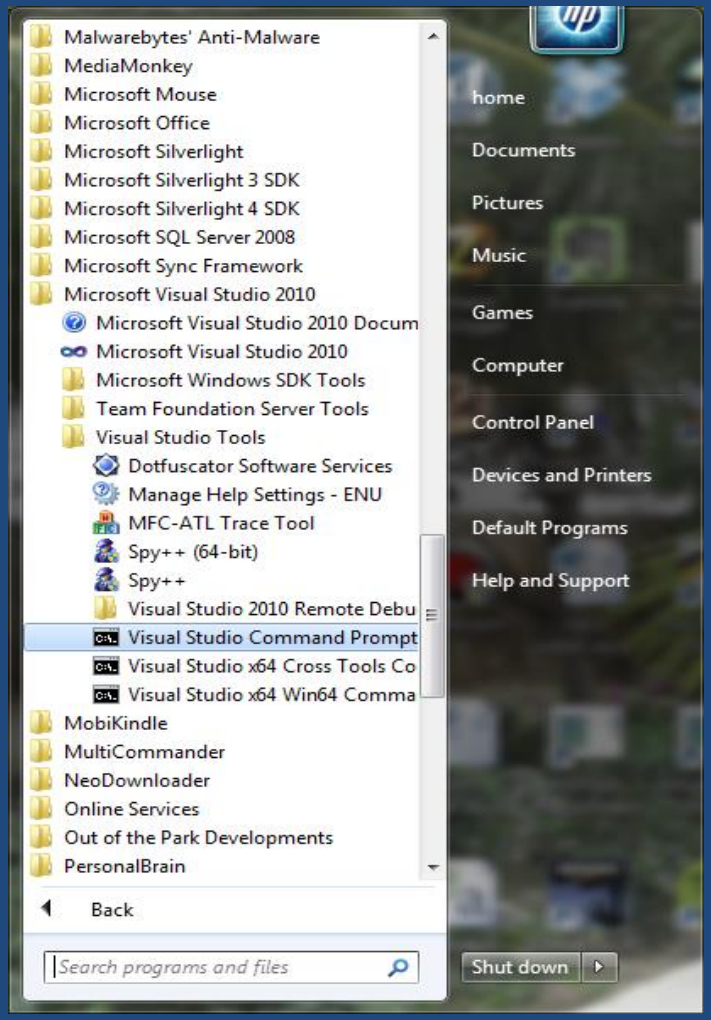
use

Visual Studio

That's not the way it used to be

And that's not
the way it has to be

Visual Studio Command Prompt



Alternative to ...

You can use a normal command prompt but ...

you'd have to use the vsvars32.bat file

(found in C:\Program Files (x86)\Microsoft Visual Studio 10.0\common7\tools) or C:\Program Files\Microsoft Visual Studio 10.0\common7\tools)

Let's create a C source file

Start up

notepad

or any other text editor

NOT

Microsoft Word

(it's not a text editor)

Create

times.c

and save it

somewhere that you can find it

```
#include "timestable.h"
int main (void)
{
int x = 0;
int table = 0;
char buffer[100] = "";
```

```
printf ("Enter a times table: ");
fgets (buffer, 100, stdin);
table = atoi (buffer);
printf ("%d Times Table\n\n",
                table);
```

```
for (x = 1; x <= 10; x++)
{
  printf ("%2d x %2d = %d\n",
            x, table, x * table);
}
return 0;
}
```

All on one page for easy pasting ...

```
#include "timestable.h"
int main (void)
{
int x = 0;
int table = 0;
char buffer[100] = "";
	printf ("Enter a times table: ");
	fgets (buffer, 100, stdin);
	table = atoi (buffer);
	printf ("%d Times Table\n\n", table);
	for (x = 1; x <= 10; x++)
	{
		printf ("%2d x %2d = %d\n", x, table, x * table);
	}
	return 0;
}
```

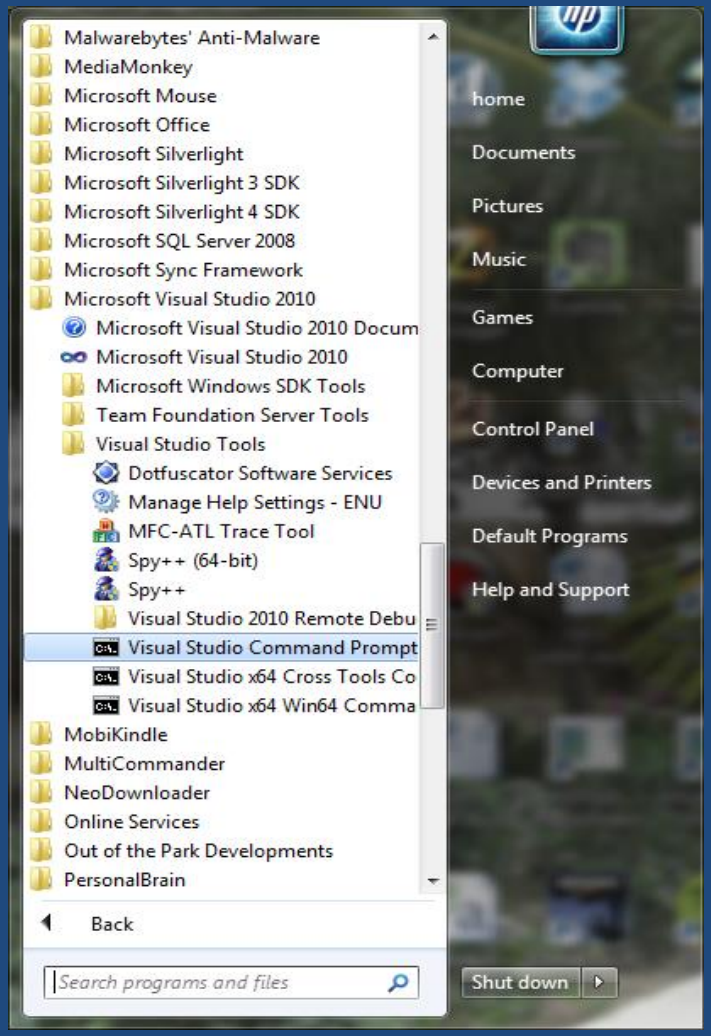
OK, did you make sure that you know where the source file is stored?

In the same directory, create timestable.h using notepad

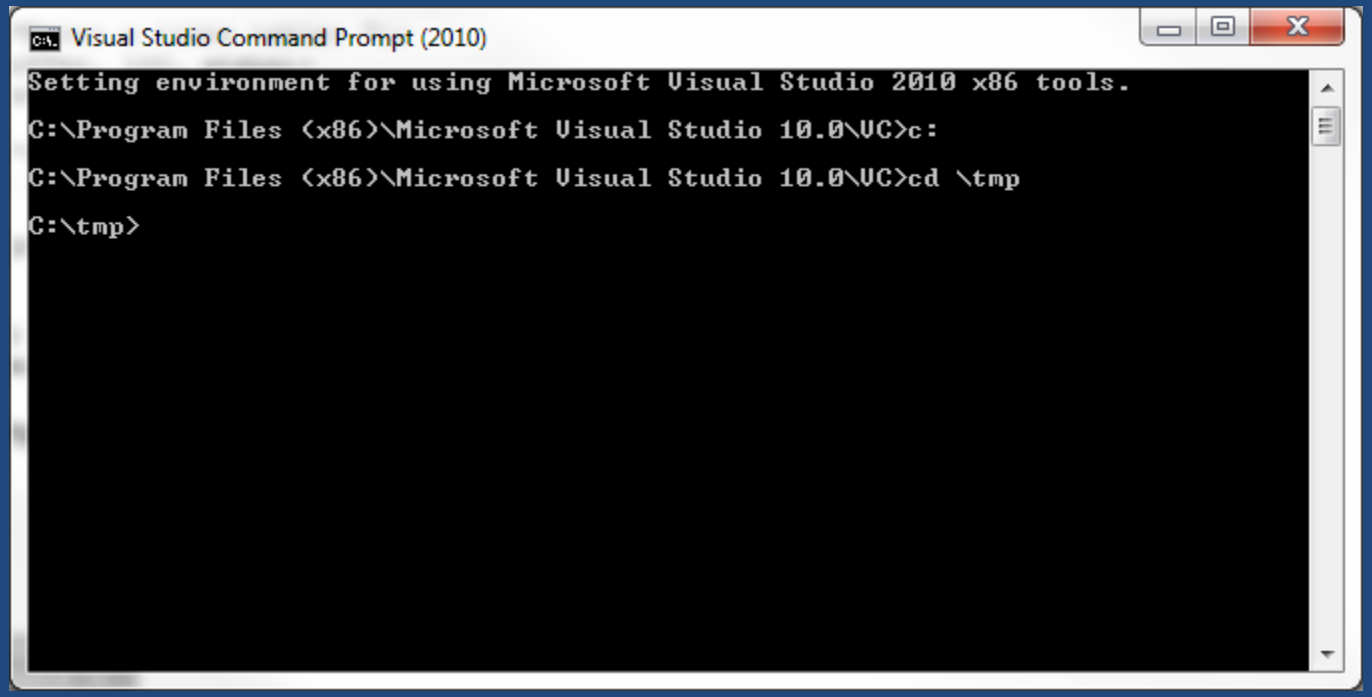
```
#include <stdio.h>

#include <stdlib.h>
```

Start up Visual Studio Command Prompt



Go to your directory



Now, let's compile it

Visual Studio Command Prompt (2010)

```
C:\tmp>cl times.c
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 16.00.40219.01 for 80x86
Copyright (C) Microsoft Corporation.  All rights reserved.

times.c
Microsoft (R) Incremental Linker Version 10.00.40219.01
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:times.exe
times.obj

C:\tmp>
```

cl.exe

cl stands for "compile and link"

It generates

times.exe

You can run your program by simply typing

```
times
```

Visual Studio Command Prompt (2010)

```
C:\tmp>times
Enter a times table: 3
3 Times Table

 1 x  3 = 3
 2 x  3 = 6
 3 x  3 = 9
 4 x  3 = 12
 5 x  3 = 15
 6 x  3 = 18
 7 x  3 = 21
 8 x  3 = 24
 9 x  3 = 27
10 x  3 = 30

C:\tmp>
```

How about making changes?

Just edit the file

and

compile-and-link (cl)

it again

But ...

More files?

More problems ...

unless ...

A makefile

can

help us

Use

NMAKE

Linux note:

make replaces nmake

Using makefile rules

Makefiles

contain

rules

Defining

how

to

create

something

Like this:

```
times.exe: times.c
    cl times.c
```

Each rule contains:

1) a dependency

2) an instruction

(and possibly more of each)

The dependency looks like this:

```
times.exe: times.c
```

meaning:

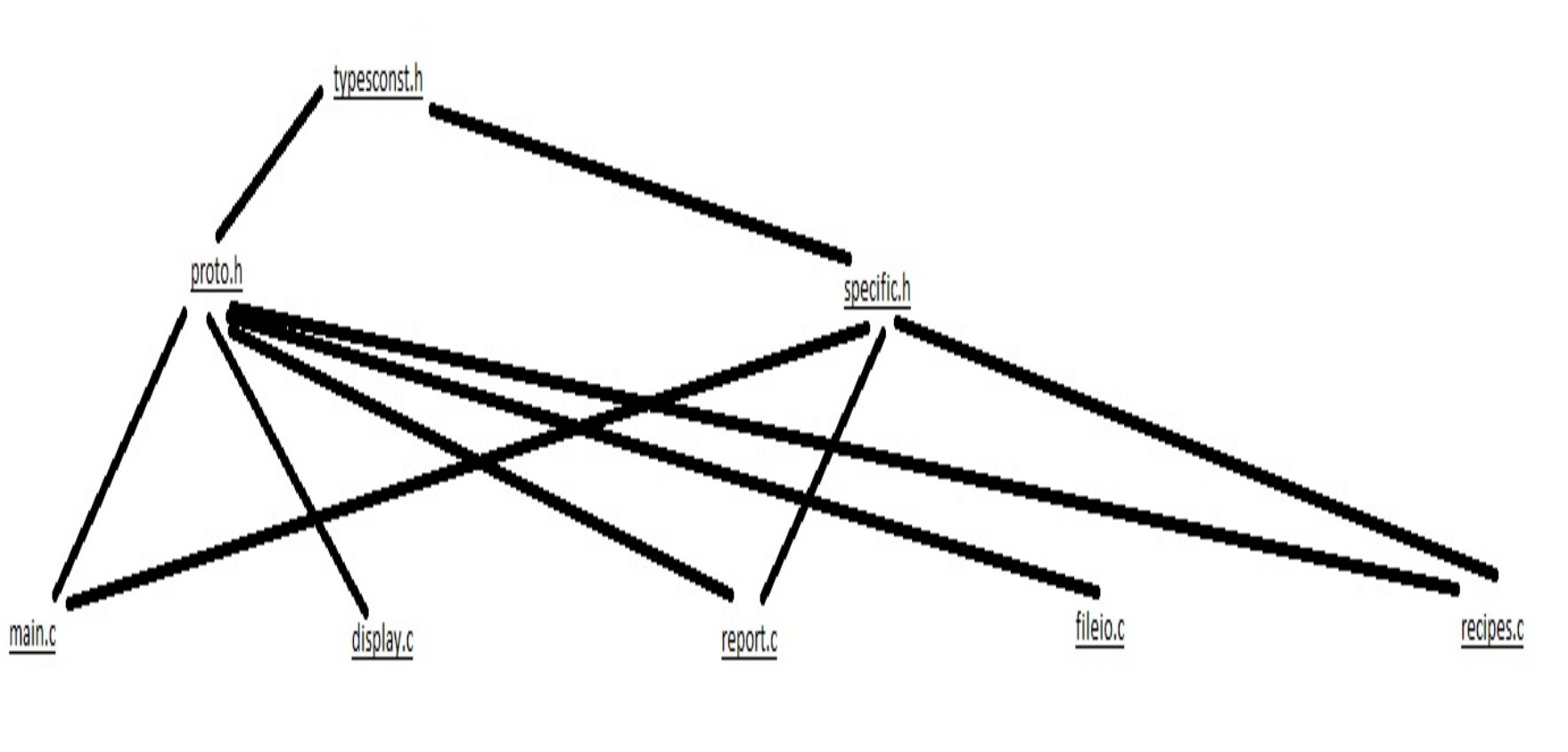
"times.exe is dependent on times.c and should be recreated if times.c changes"

The instruction is:

```
cl times.c
```

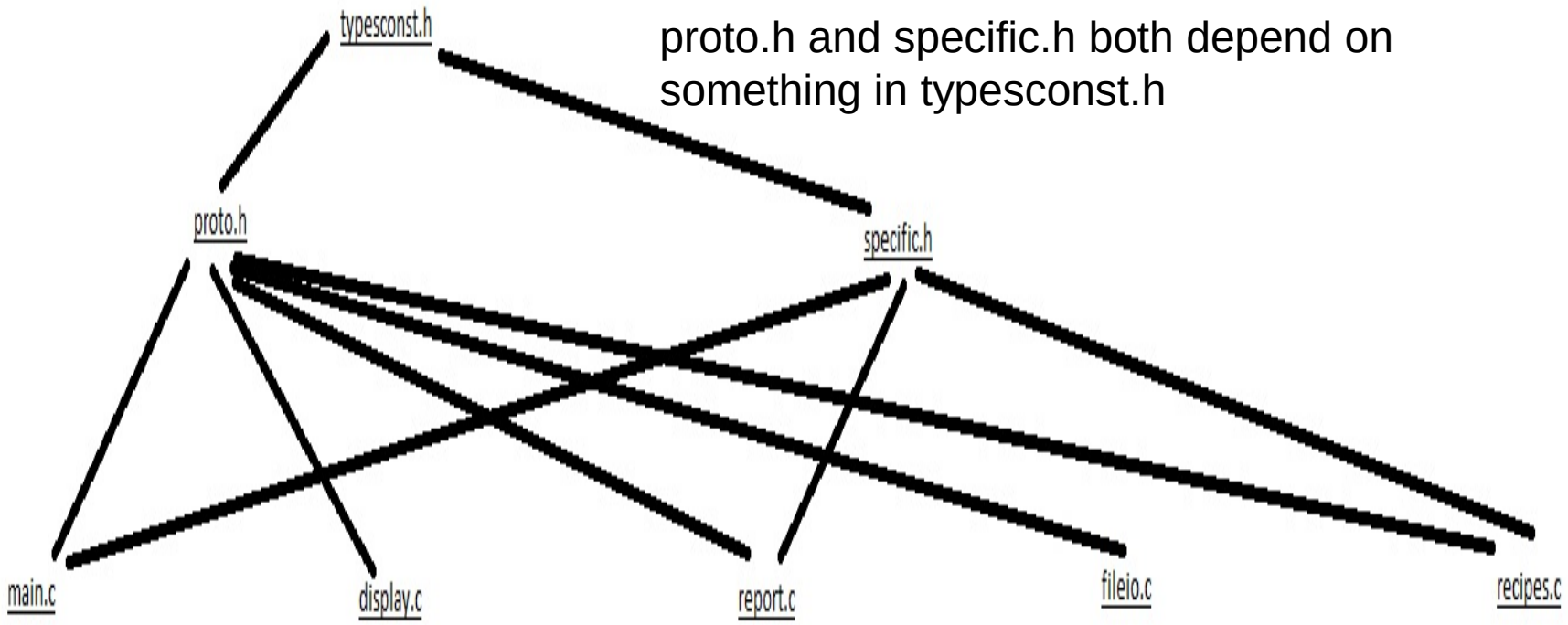
In most common projects, there's more:

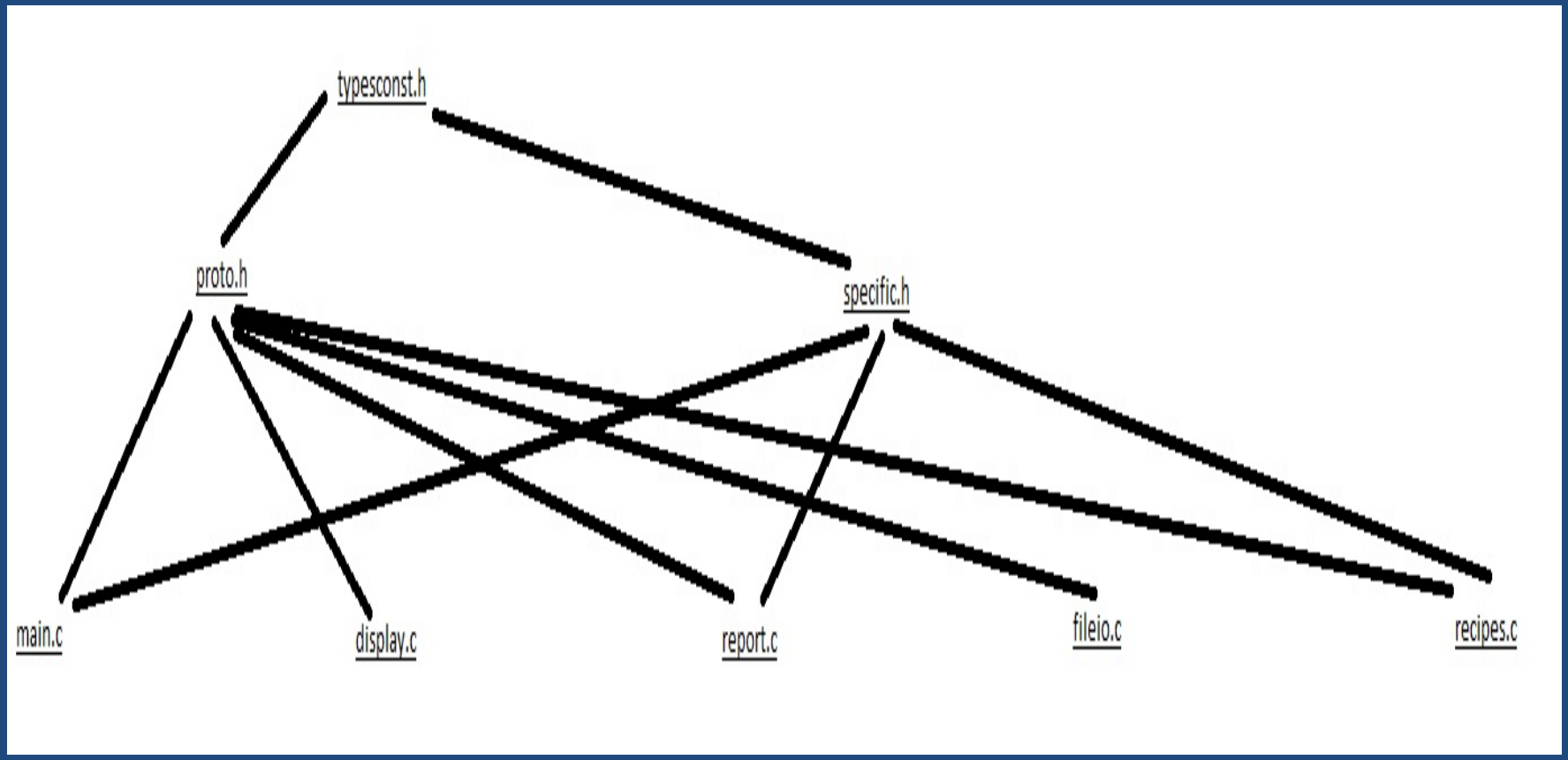
Small example



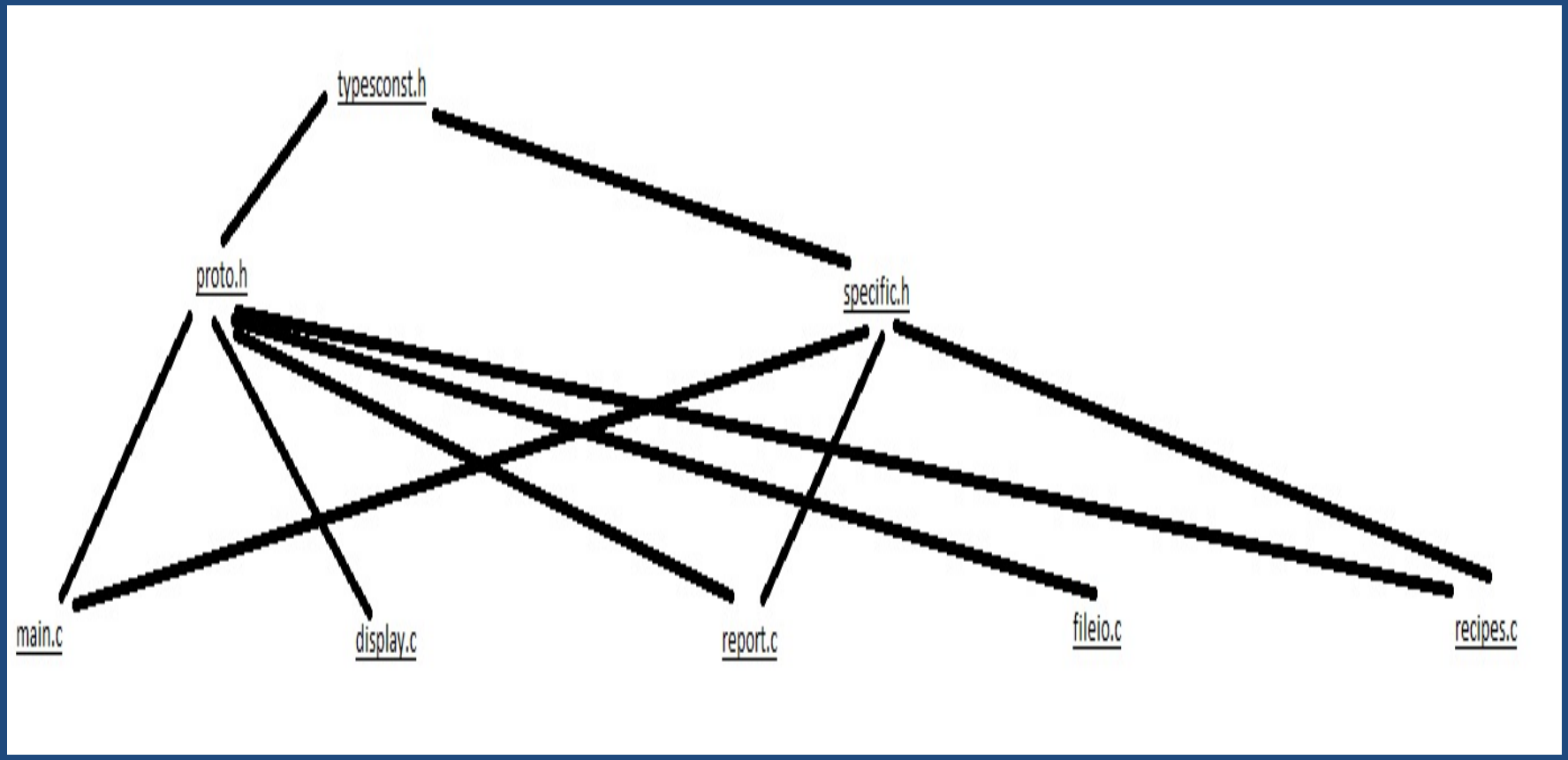
Explanation:

proto.h and specific.h both depend on something in typesconst.h





All C source files depend on proto.h



But only main.c, report.c, and recipes.c depend on specific.h

If you change something in specific.h,

ONLY

main.c, report.c, and recipes.c are affected

display.c and fileio.c
are not affected

So why bother recompiling display.c and fileio.c

This was automatic

in

Visual Studio

We can do the same
by creating rules
in a makefile

Warning!

If you use NOTEPAD
to create a makefile,
it will create
makefile.txt

Simple example

```
application.exe: main.obj input.obj output.obj
    cl main.obj input.obj output.obj /Feapplication.exe

main.obj: main.c prototypes.h
    cl main.c /c

input.obj: input.c prototypes.h special.h
    cl input.c /c

output.obj: output.c prototypes.h special.h
    cl output.c /c
```

Rules:

1) Dependencies
2) Instructions

Dependencies have:

1) a dependent target
2) a colon
3) a list of files the target is dependent on

Revisited

```
application.exe: main.obj input.obj output.obj
    cl main.obj input.obj output.obj /Feapplication.exe

main.obj: main.c prototypes.h
    cl main.c /c

input.obj: input.c prototypes.h special.h
    cl input.c /c

output.obj: output.c prototypes.h special.h
    cl output.c /c
```

Instructions are indented by tabs

and

end with a blank line

Revisited

```
application.exe: main.obj input.obj output.obj
    cl main.obj input.obj output.obj /Feapplication.exe

main.obj: main.c prototypes.h
    cl main.c /c

input.obj: input.c prototypes.h special.h
    cl input.c /c

output.obj: output.c prototypes.h special.h
    cl output.c /c
```

For this particular instruction set:

/Feapplication.exe

means

"call the executable application.exe, not main.exe"

/c means:

"don't link,

just compile"

Both of these are
part of
the cl command,
not nmake

In the quiz,
you'll be creating a
makefile