Lab: areas and volumes, a multi-module C project

1 Objective

Extend the circle example from the *C build* session into a small project with three modules, and build it stage by stage with gcc. At the end of this lab you will have:

- extended the `area` module with new shapes;
- written a new module, `volume`, that reuses the functions of `area`;
- built the project with separate compilation and linking;
- inspected the object files and recognized a linker error.

> Estimated time: 60–90 minutes. Follow the Barr-C style guide, as in the course examples: function names start with the module name (`area_...`, `volume_...`), every header has an include guard, and braces go on their own line.

2 Requirements

- The tools of the *Lab setup* guide installed and working: `gcc --version` must work in a Windows command prompt.
- The slides of the *C build* session.

> In this lab you edit the files with Notepad and build them from the command prompt. Do not use the play button of VS Code: it builds just the active file, and this project has three `.c` files.

3 Starting point

1. Create a new folder for this lab and move into it:

```
C:\...> cd %USERPROFILE%\c_projects
C:\...> mkdir lab_shapes
C:\...> cd lab_shapes
```

> Open or create each file from the command prompt, inside the lab folder: `notepad area.h`. If the file does not exist, Notepad asks whether to create it: answer *Yes*. This way the file gets exactly the name you typed. If you save a new file from the *Save as* dialog instead, Notepad may name it `area.h.txt` and gcc will not find it. The status bar of Notepad shows the line and column of the cursor, which helps to locate the errors reported by gcc (e.g. `main.c:10:48`).

2. Create the files `area.h` and `area.c` with the code of the session. They are your starting point.

File `area.h`:

```
/** @file area.h
 *  @brief Area of some plane shapes.
 */
#ifndef AREA_H
#define AREA_H

double area_circle(double radius);

#endif /* AREA_H */
```

File `area.c`:

```
/** @file area.c
 *  @brief Area of some plane shapes.
 */
#include "area.h"

#define PI 3.14159265358979323846

/* Area of a circle of the given radius. */
double area_circle(double radius)
{
    return PI * radius * radius;
}
```

4 Part 1: extend the area module

Add two new shapes to the `area` module:

- `double area_rectangle(double width, double height)` returns the area of a rectangle.
- `double area_square(double side)` returns the area of a square.

1. Add the two prototypes to `area.h`, inside the include guard.
2. Write the two functions in `area.c`.

> `area_square()` **must call** `area_rectangle()` to do its job: a square is a rectangle with equal sides. Do not write `side * side`. Reusing functions instead of duplicating code is the point of this exercise.

3. Compile the module alone. The `-c` option stops after the compiling stage and produces the object file `area.o`:

```
C:\...\lab_shapes> gcc -c area.c -std=c99 -Wall -Wextra -pedantic
```

4. Fix every error **and every warning** until the command prints nothing.
5. List the symbols defined in `area.o` with the `nm` tool. You should see your three functions marked with `T` (code defined in this file):

```
C:\...\lab_shapes> nm area.o
0000000000000000 T area_circle
0000000000000025 T area_rectangle
0000000000000043 T area_square
```

> The addresses on the left can be different on your computer; the names and the letter `T` are what matter.

5 Part 2: write the volume module

Create a new module, `volume`, made of the files `volume.h` and `volume.c`. It computes the volume of three solids:

- `double volume_cylinder(double radius, double height)` returns the volume of a cylinder.
- `double volume_cuboid(double length, double width, double height)` returns the volume of a cuboid (a rectangular box).
- `double volume_cube(double side)` returns the volume of a cube.

All three solids follow the same rule: *volume = area of the base × height*. Each function **must call** the matching function of the `area` module to compute the area of its base (circle, rectangle or square).

1. Write `volume.h` with the three prototypes and an include guard named `VOLUME_H`.
2. Write `volume.c`. Think about which header files it needs to include.
3. Compile the module alone and fix every warning:

```
C:\...\lab_shapes> gcc -c volume.c -std=c99 -Wall -Wextra -pedantic
```

4. List its symbols:

```
C:\...\lab_shapes> nm volume.o
```

> Notice that `volume.o` compiles even though the code of the `area` functions is not in it. The compiler only needs their prototypes. Keep the output of `nm volume.o`: you will need it in Part 4.

6 Part 3: build and run the project

1. Create `main.c` with the following code. Do not change it: your modules must work with it as it is.

```c
/** @file main.c
 *  @brief Test program for the area and volume modules.
 */
#include <stdio.h>
#include "area.h"
#include "volume.h"

int main(void)
{
    double radius = 1.5;
    double length = 4.0;
    double width  = 2.5;
    double height = 3.0;
    double side   = 2.0;

    printf("Areas\n");
    printf("  circle    r = %.1f           : %8.3f\n",
           radius, area_circle(radius));
    printf("  rectangle %.1f x %.1f         : %8.3f\n",
           length, width, area_rectangle(length, width));
    printf("  square    side = %.1f        : %8.3f\n",
           side, area_square(side));

    printf("Volumes\n");
    printf("  cylinder  r = %.1f, h = %.1f  : %8.3f\n",
           radius, height, volume_cylinder(radius, height));
    printf("  cuboid    %.1f x %.1f x %.1f   : %8.3f\n",
           length, width, height, volume_cuboid(length, width, height));
    printf("  cube      side = %.1f        : %8.3f\n",
           side, volume_cube(side));

    return 0;
}
```

2. Compile `main.c` alone:

```
C:\...\lab_shapes> gcc -c main.c -std=c99 -Wall -Wextra -pedantic
```

3. Link the three object files into the executable `shapes.exe`:

```
C:\...\lab_shapes> gcc main.o area.o volume.o -o shapes
```

4. Run the program and check that the output is exactly this:

```
C:\...\lab_shapes> shapes
Areas
  circle    r = 1.5           :    7.069
  rectangle 4.0 x 2.5         :   10.000
  square    side = 2.0        :    4.000
Volumes
  cylinder  r = 1.5, h = 3.0  :   21.206
  cuboid    4.0 x 2.5 x 3.0   :   30.000
  cube      side = 2.0        :    8.000
```

Take a screenshot of the command prompt showing the build commands and the output of the program.

7 Part 4: questions

Answer these questions in a plain text file named `answers.txt`. Short answers are fine, but justify them.

1. In the output of `nm volume.o`, which symbols are marked with `U`? What does `U` mean, and which stage of the build will deal with them?
2. Why does `volume.c` include `area.h`, while `volume.h` does not need to?
3. Link the program again, but forget `area.o`:

   ```
   C:\...\lab_shapes> gcc main.o volume.o -o shapes
   ```

   Which stage fails? Copy the first lines of the error message and explain how you recognized the stage. Why are both `main.o` and `volume.o` mentioned?
4. Suppose you change only the value of `PI` in `area.c`. Which of the commands of Part 1 to Part 3 must you run again to get an updated `shapes.exe`? And if you change a prototype in `area.h`?
5. Thanks to `area_square()` calling `area_rectangle()`, how many functions would you need to modify if, one day, the area of a rectangle had to be computed differently (for example, with a correction factor)?

8 Going further (optional)

Add a fourth solid, `double volume_sphere(double radius)`, whose volume is 4/3 × π × r³. It needs the value of π, which is defined in `area.c`.

- Can `volume.c` use the `PI` defined in `area.c`? Why?
- Propose a solution that does not duplicate the value of π in two files, implement it and explain your choice in `answers.txt`.

9 Checklist and submission

Your lab is complete if:

- ☐ `area.c` and `volume.c` compile with `-std=c99 -Wall -Wextra -pedantic` without any warning.
- ☐ `area_square()` calls `area_rectangle()`, and each `volume_...()` function calls a function of the `area` module.
- ☐ `volume.h` has an include guard.
- ☐ `shapes.exe` prints exactly the expected output.
- ☐ `answers.txt` answers the five questions.

> Submit `area.h`, `area.c`, `volume.h`, `volume.c`, `answers.txt` and the screenshot of Part 3, as indicated by your instructor. Do not submit `.o` or `.exe` files.

10 Troubleshooting

- `fatal error: area.h: No such file or directory`**.** The header is not in the current folder, or its name is misspelled. Check with `dir`.
- `warning: implicit declaration of function ...`**.** The file uses a function whose prototype it has not seen: an `#include` is missing. Never ignore this warning.
- `undefined reference to ...`**.** This is a linker error: an object file is missing from the link command, or a function is declared but never defined (check the spelling).
- `multiple definition of ...`**.** The same function is defined in two `.c` files, or a `.c` file was included with `#include`. Only include `.h` files.
- **The output does not match.** Check the formula of each function and that you did not modify `main.c`.