

CSE1310 INLAB 03

We will start with a simple C program to calculate the area of a circle. We will be using omega for working out all of our C programs and “vi” or “pico” as our editor. Log onto omega and type:

vi circle.c

or

pico circle.c

This command opens a new file called circle.c in the “vi” or “pico” editor.

Some coding necessities:

The C programs that we will be writing for this class must end with the “.c” suffix. It is also necessary to ensure that all programs have meaningful variable declarations and are documented properly using comments. Failure to use comments in your program will result in your program will result in points being deducted.

Type in the following program inside the circle.c file:

```
/* circle.c - This is the program header for a program that calculates the area of a circle.
```

```
*/
```

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

```
#define PI 3.14159 /* mathematical constant pi */
```

```
int main (void)
```

```
{
```

```
    double radius; /* radius of the circle */
```

```
    double area; /* area of the circle */
```

```
    printf(“Enter a value for radius of the circle: ”);
```

```
    scanf(“%lf”, &radius);
```

```
    area = PI * (radius * radius);
```

```
    printf(“The area of the circle with radius %f is %f\n”, radius, area);
```

```
    return (0);
```

```
}
```

Save the program and exit the “vi” or “pico” editor.

Now, for the above program, **perform a design and analysis based on what the inputs and expected outputs are, and what the processing steps involved are.** Represent these by drawing a flowchart diagram. Once you finish, signal the grader to check your work.

Checkpoint 1

Now, compile the program “circle.c” using the command:

gcc circle.c

If an error message appears, go back to the file “circle.c” using the “vi” or “pico” editor and fix the errors accordingly. Remember that if you make any changes to your program, you must save the changes and re-compile the program before executing it. Unless another filename has been specified for the executable form of this program, the program is executed by typing the following command:

a.out

When your program runs without errors, signal the grader to check your work.

Checkpoint 2

Through Checkpoint 1 and Checkpoint 2, we were able to print some output onto the screen based on the user's choice of input values so that "a.out" had to be executed each time a new radius value was entered. But in contrast to this, if we fix the value of the radius, then "a.out" will be executed only once during program execution. So, in order to calculate the area for a different radius value, we need to change the value of radius in the program. This reduces the execution time, but each time the radius is changed in the program, we need to compile the program again. So the execution and compile times vary if we change the way the input is given to the program. Create another file using "vi" or "pico" called "circle1.c" and type the following. This program is a modification of "circle.c".

Save the program and quit "vi" or "pico".

```
/* circle1.c — This program calculates the area of a circle. It is modified from circle.c */
#include <stdio.h>
#define PI 3.14156 /* mathematical constant PI */
#define CONST_RADIUS 4.55 /* constant value for radius */
int main(void)
{
    double area; /* area of the circle */
    area = PI * (CONST_RADIUS * CONST_RADIUS);
    printf("The area of the circle with radius %f is: %f\n", CONST_RADIUS, area);
    return (0);
}
```

Compile the program again using the command:

```
gcc circle1.c
```

and execute it. Make sure the program runs without any errors. When you have finished, signal the grader to check your work.

Checkpoint 3

In the file circle1.c, change the value of CONST_RADIUS from 4.55 to 6.333. Then save the program, compile and execute it as you did in the previous checkpoints. Observe what you get as output of the program, it should be a different value. With the output on the screen, signal the grader to check your work.

Checkpoint 4

What we are going to do now is to create errors in the program circle.c. Provide answers to the following questions through a table provided in the next page after you perform the tasks listed I through 8 in this checkpoint. Make sure to compile **and execute each task independently in order**.

Questions:

1. Were there any errors when you compiled the program, when you executed the program or were there no errors?
2. If there were errors, what were the error messages?

Tasks:

1. Remove the “%” from the scanf statement. Compile and run the program.
2. Place the “%” in scanf and remove the “&”.
3. Place the “&” back and remove the beginning “ from the string in printf.
4. Place the “ back in printf. Now replace the “double” data type for variable “area” with the “int” data type.
5. Replace the “double” with “int” and then remove the #include before the main function.
6. Place the #include back, and remove the semicolon from any one of the statements in the program.
7. Place the semicolon back from wherever it was removed, and place an = sign after “#define”.
8. Remove the = sign in the define statement and remove */ from the opening comment which describes the program.
9. Finally place the */ back.

Note: After introduction of an error, compile the program using “gcc”, record the error in the table given below. After fixing each error, compile to ensure that the program is error-free before going to the next task.

Error	Error during Compile Time?	Error during Run Time?	No Error	Error Message Given
1				
2				
3				
4				
5				
6				
7				
8				

When you complete the table, signal the grader to check your work.

Checkpoint 5

We will now expand on “circle.c” and make sure it runs without error. Modify the code for “circle.c” as follows:

1. Declare another variable called ‘radius1’ of type “double”, add a printf statement that will prompt for this new radius value and another scanf that will store this value.
2. Also, declare four “double” variables named total1, total2, total3, and total.

Compile the program using “gcc” and execute it using “a.out”. If there is an error, there is a good chance that the message that is given will be in our table above. When the program is working signal the grader.

Checkpoint 6

Enter the following code at the end of the program “circle.c”, i.e. just above the return statement.

```
total = radius + radius1;  
printf(“ The sum of radius and radius1 is: %f”, total);
```

Also, comment out the printf statement used to print the area of the circle.

Compile and execute the program. If there is an error fix it. Next do the following:

1. Enter the code that will allow **radius1** to be subtracted from **radius** and store the answer in **total1**.
2. Enter the code that will allow **radius** to be multiplied by **radius1** and store the answer in **total2**.
3. Enter the code that will allow **radius1** to be divided by **2** and store the answer in **total3**.
4. Enter the code that will add **total** and **total1**, subtract that amount from **total2** and then divide that answer by **total3**. The final answer should be stored in the variable **total**.
5. Using values 12.5 for **radius** and 6.5 for **radius1**, display the results of 1,2,3 and 4 to the screen.

When you have finished, signal the grader.

Checkpoint 7

For the rest of the semester, anytime you hand in a lab to your instructor, it must contain a header at the top of your program. The header should contain the following:

1. Your name.
2. Your lab instructors name.
3. Your lab section (also include 1310).
4. The duration of the lab.
5. Lab number.
6. Lab due date.
7. Brief description of the objective of the lab.

Add header to circle.c and remember the header is treated like a comment so you must encapsulate it with a /* comment */. When you have completed your header signal the grader.

Checkpoint 8

The End!