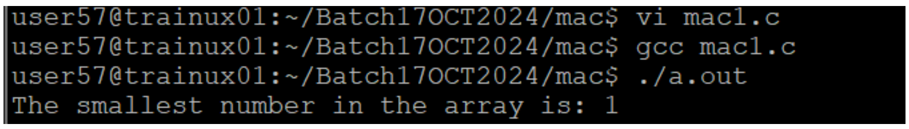
CONSTANTS AND MACROS ASSIGNMENT

1.Write a function macro to find the smallest number in an array of integers

Sol:

```
#include <stdio.h>

#define FIND_SMALLEST(arr, size) ({ \

   int min = arr[0]; \

   for (int i = 1; i < size; i++) { \

      if (arr[i] < min) { \

         min = arr[i]; \

      } \

   } \

   min; \

})

int main() {

   int arr[] = {5, 2, 9, 1, 7, 6};

   int size = sizeof(arr) / sizeof(arr[0]);

  int smallest = FIND_SMALLEST(arr, size);

  printf("The smallest number in the array is: %d\n", smallest);

 return 0;

}
```

Output:



```
user57@trainux01:~/Batch17OCT2024/mac$ vi mac1.c
user57@trainux01:~/Batch17OCT2024/mac$ gcc mac1.c
user57@trainux01:~/Batch17OCT2024/mac$ ./a.out
The smallest number in the array is: 1
```

2. What are the differences between macros and constant. Can you replace a constant with a macro and vice versa? Give examples for your statements.

Sol:

Macros and constants are both used to define values in C, but they differ significantly in how they are used and handled by the compiler.

Macros:

Defined using #define. It is a preprocessor directive.

Replaces the code wherever the macro is used during preprocessing (before the compilation phase).

Macros are globally scoped. They are not limited to functions or blocks.

Macros do not occupy memory. They are purely replaced by the preprocessor.

Constants:

Defined using const keyword or #define (if a literal constant).

Replaced by the compiler at runtime. The value remains in the final code as a constant literal.

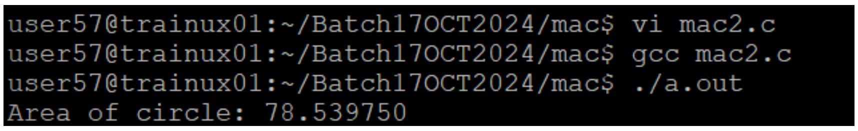
Constants have scope limited to where they are defined (usually within a function or file).

Constants can occupy memory depending on the type. For example, const int x = 5; takes memory.

Yes, you can replace a constant with a macro, but this is generally not recommended for simple constants because it loses type safety and clarity. A macro is a preprocessor directive and is replaced before compilation, while a constant is managed by the compiler with type safety.

```
#include <stdio.h>


#define PI 3.14159  // Macro for constant PI

// const double PI = 3.14159;

int main() {

   double radius = 5.0;

   double area = PI * radius * radius;  // Using the macro

   printf("Area of circle: %f\n", area);

   return 0;

}
```

Output:



```
user57@trainux01:~/Batch17OCT2024/mac$ vi mac2.c
user57@trainux01:~/Batch17OCT2024/mac$ gcc mac2.c
user57@trainux01:~/Batch17OCT2024/mac$ ./a.out
Area of circle: 78.539750
```

3. Refer macro below

#define MYPROD(x) (x *x)

WAP to invoke the above macro with inputs as below and display the result.

a. MYPROD(2+1)

b. MYPROD(6+1)

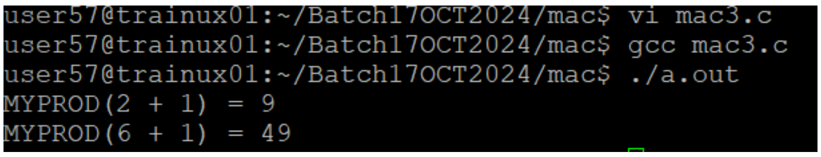
Do you get the expected answers as 9 and 49 in case a. and case b.?

If not modify the code to produce the expected results. in above case

Sol:

```
#include <stdio.h>

#define MYPROD(x) ((x) * (x))


int main() {

   int result1 = MYPROD(2 + 1);  // Expected: 9

   int result2 = MYPROD(6 + 1);  // Expected: 49


   printf("MYPROD(2 + 1) = %d\n", result1);

   printf("MYPROD(6 + 1) = %d\n", result2);


   return 0;

}
```

Output:



```
user57@trainux01:~/Batch17OCT2024/mac$ vi mac3.c
user57@trainux01:~/Batch17OCT2024/mac$ gcc mac3.c
user57@trainux01:~/Batch17OCT2024/mac$ ./a.out
MYPROD(2 + 1) = 9
MYPROD(6 + 1) = 49
```

4. Write macro definitions with arguments for calculation of area of a triangle and circle.

a. Use macros for both constants as well as formula evaluations.

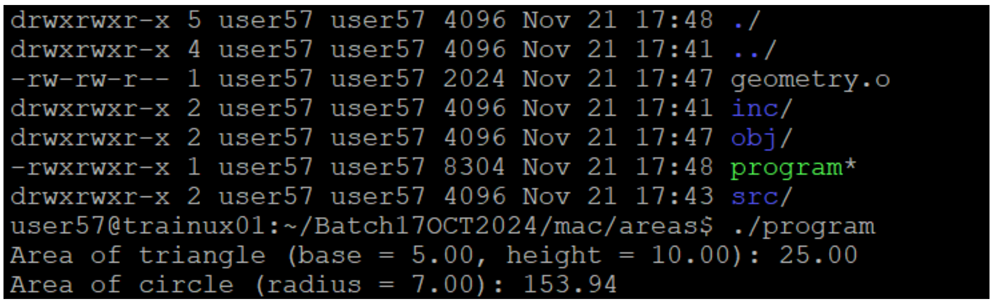
b. Store these macro definitions in a header file and invoke the macros from the main function

sol:

```
// geometry.h

#ifndef GEOMETRY_H
```

```c
#define GEOMETRY_H

#define PI  3.14159

#define AREA_OF_TRIANGLE(base, height) (0.5 * (base) * (height))

#define AREA_OF_CIRCLE(radius) (PI * (radius) * (radius))

#endif


// geometry.c

#include <stdio.h>

#include "geometry.h"

int main() {

    double base = 5.0;

    double height = 10.0;


    double radius = 7.0;

    double triangle_area = AREA_OF_TRIANGLE(base, height);

    printf("Area of triangle (base = %.2f, height = %.2f): %.2f\n", base, height, triangle_area);


    double circle_area = AREA_OF_CIRCLE(radius);

    printf("Area of circle (radius = %.2f): %.2f\n", radius, circle_area);


    return 0;

}
```

Output:



```
drwxrwxr-x 5 user57 user57 4096 Nov 21 17:48 ./
drwxrwxr-x 4 user57 user57 4096 Nov 21 17:41 ../
-rw-rw-r-- 1 user57 user57 2024 Nov 21 17:47 geometry.o
drwxrwxr-x 2 user57 user57 4096 Nov 21 17:41 inc/
drwxrwxr-x 2 user57 user57 4096 Nov 21 17:47 obj/
-rwxrwxr-x 1 user57 user57 8304 Nov 21 17:48 program*
drwxrwxr-x 2 user57 user57 4096 Nov 21 17:43 src/
user57@trainux01:~/Batch17OCT2024/mac/areas$ ./program
Area of triangle (base = 5.00, height = 10.00): 25.00
Area of circle (radius = 7.00): 153.94
```

5. Define a macro name MYPRINT as below.

#define MYPRINT(x) printf(x)

Use the above macro conditionally only if a macro CUST_PRINT is defined , otherwise not to be used.

For eg refer the code and comments below.

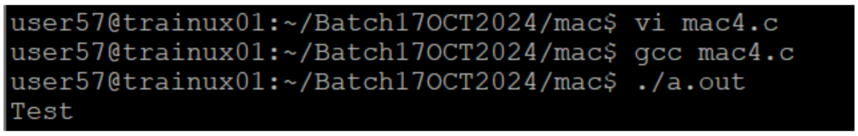
```
int main()

{

MYPRINT("Hello World"); // will be displayed only when CUST_PRINT is defined

printf("Test"); // will be displayed always irrepective of CUST_PRINT

return 0;

}
```

Add the code to demonstrate the above behaviour.

Sol:

```
#include <stdio.h>

#ifdef CUST_PRINT

  #define MYPRINT(x)

#else

  #define MYPRINT(x)

#endif

int main()

{

  MYPRINT("Hello World\n");  // Will be displayed only when CUST_PRINT is defined

   printf("Test\n");  // Will be displayed always, regardless of CUST_PRINT

   return 0;

}
```

Output:



```
user57@trainux01:~/Batch17OCT2024/mac$ vi mac4.c
user57@trainux01:~/Batch17OCT2024/mac$ gcc mac4.c
user57@trainux01:~/Batch17OCT2024/mac$ ./a.out
Test
```

6. Identify and use the macros to display

a. file name

b. function name

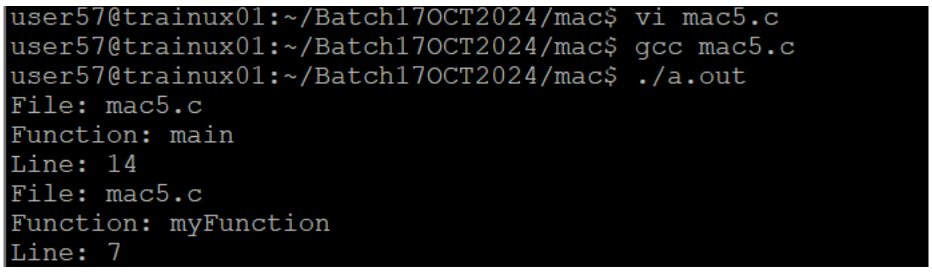
c. line of code

Show the usage with a code example

Sol:

```
#include <stdio.h>

void myFunction() {
    printf("File: %s\n", __FILE__);
    printf("Function: %s\n", __FUNCTION__);
    printf("Line: %d\n", __LINE__);

int main() {
    printf("File: %s\n", __FILE__);
    printf("Function: %s\n", __FUNCTION__);
    printf("Line: %d\n", __LINE__);
    myFunction();
    return 0;
}
```

Output:



```
user57@trainux01:~/Batch17OCT2024/mac$ vi mac5.c
user57@trainux01:~/Batch17OCT2024/mac$ gcc mac5.c
user57@trainux01:~/Batch17OCT2024/mac$ ./a.out
File: mac5.c
Function: main
Line: 14
File: mac5.c
Function: myFunction
Line: 7
```