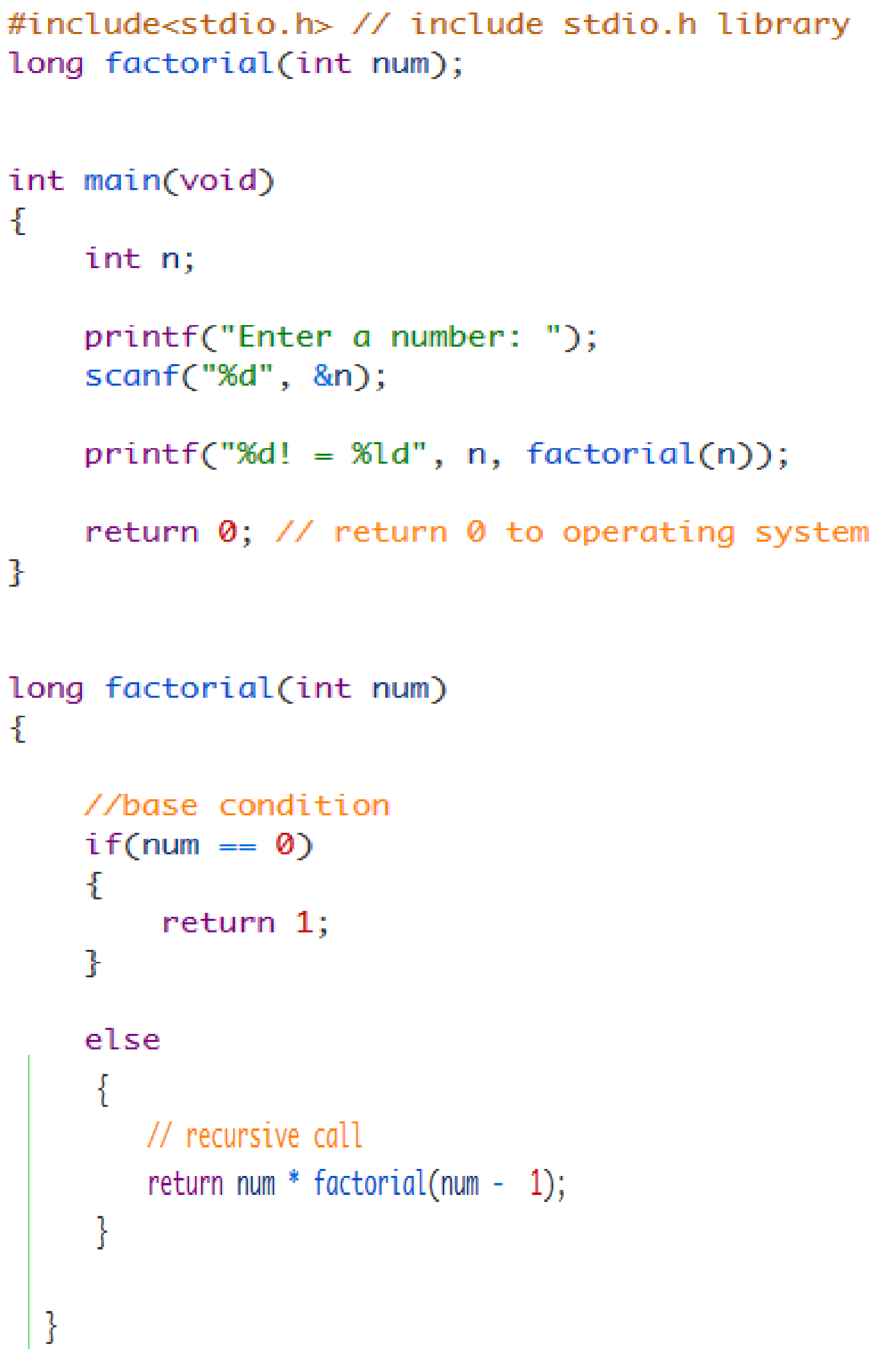
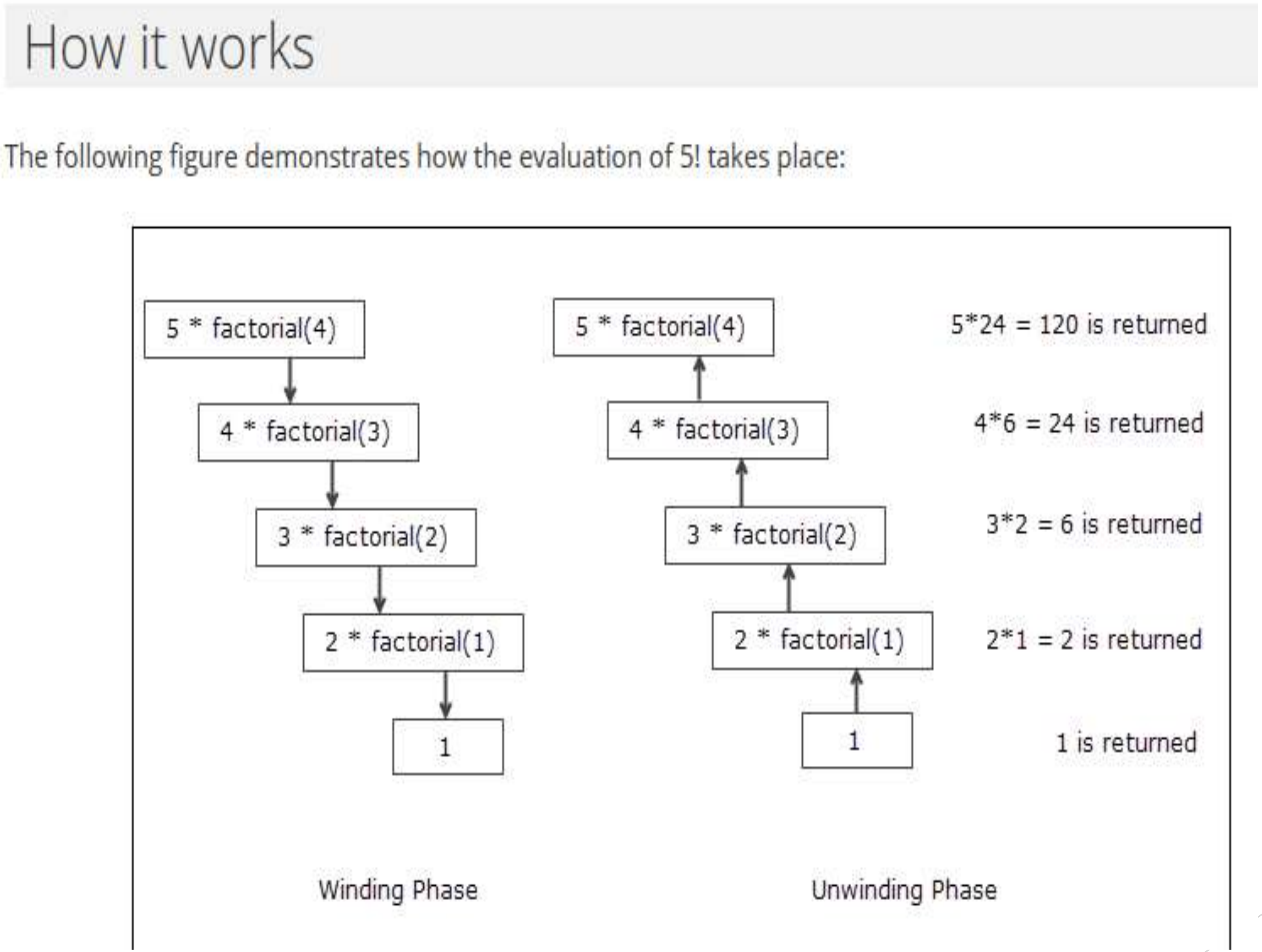
FACTORIAL USING RECURSION



```c
#include<stdio.h> // include stdio.h library
long factorial(int num);


int main(void)
{
    int n;

    printf("Enter a number: ");
    scanf("%d", &n);

    printf("%d! = %ld", n, factorial(n));

    return 0; // return 0 to operating system
}


long factorial(int num)
{

    //base condition
    if(num == 0)
    {
        return 1;
    }

    else
     {
        // recursive call
        return num * factorial(num -  1);
     }

  }
```



How it works

The following figure demonstrates how the evaluation of 5! takes place:

5 * factorial(4)

4 * factorial(3)

3 * factorial(2)

2 * factorial(1)

1

Winding Phase

5 * factorial(4)

4 * factorial(3)

3 * factorial(2)

2 * factorial(1)

1

Unwinding Phase

5*24 = 120 is returned

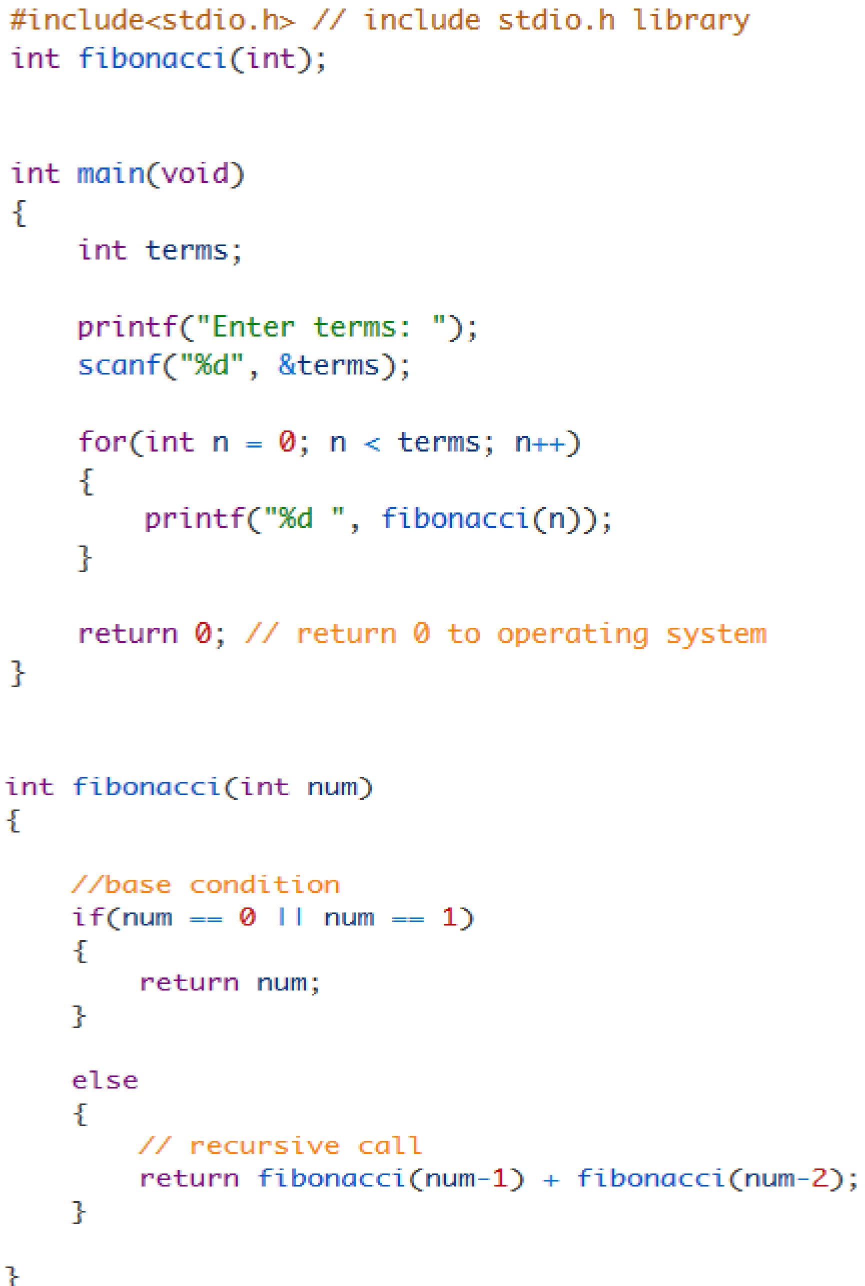
4*6 = 24 is returned

3*2 = 6 is returned

2*1 = 2 is returned

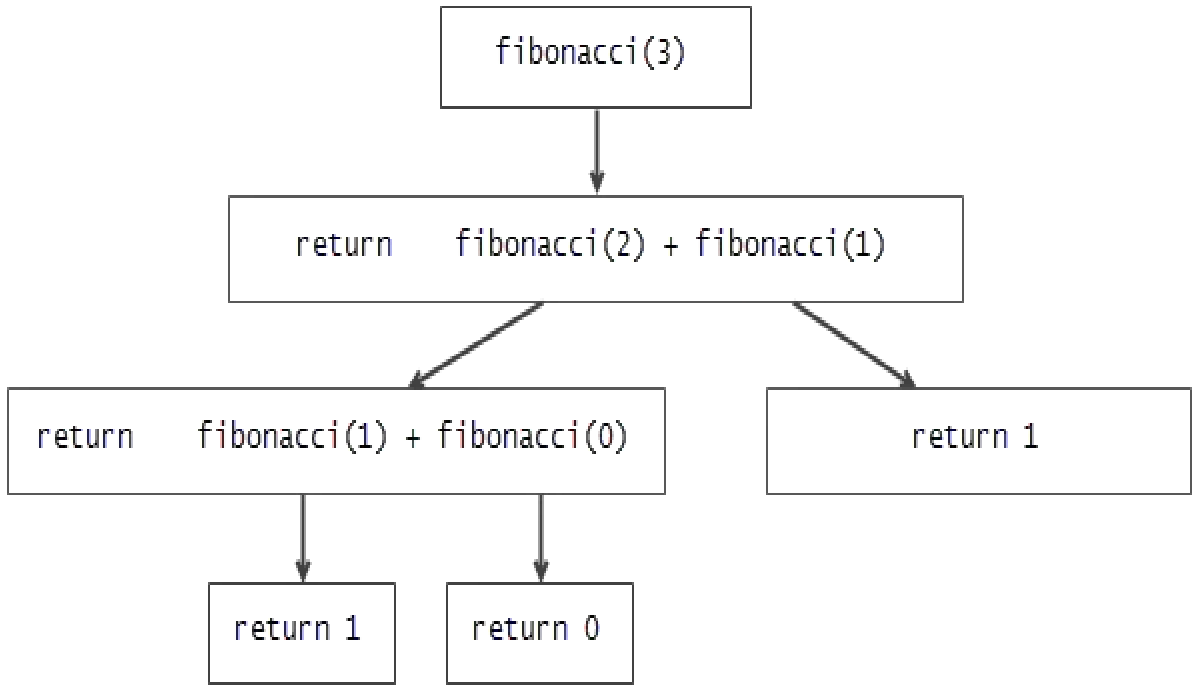
1 is returned

C Program to print Fibonacci Sequence using recursion



```c
#include<stdio.h> // include stdio.h library
int fibonacci(int);


int main(void)
{
    int terms;

    printf("Enter terms: ");
    scanf("%d", &terms);

    for(int n = 0; n < terms; n++)
    {
        printf("%d ", fibonacci(n));
    }

    return 0; // return 0 to operating system
}


int fibonacci(int num)
{

    //base condition
    if(num == 0 || num == 1)
    {
        return num;
    }

    else
    {
        // recursive call
        return fibonacci(num-1) + fibonacci(num-2);
    }

}
```

The following figure shows how the evaluation of `fibonacci(3)` takes place:



fibonacci(3)

return fibonacci(2) + fibonacci(1)

return fibonacci(1) + fibonacci(0)

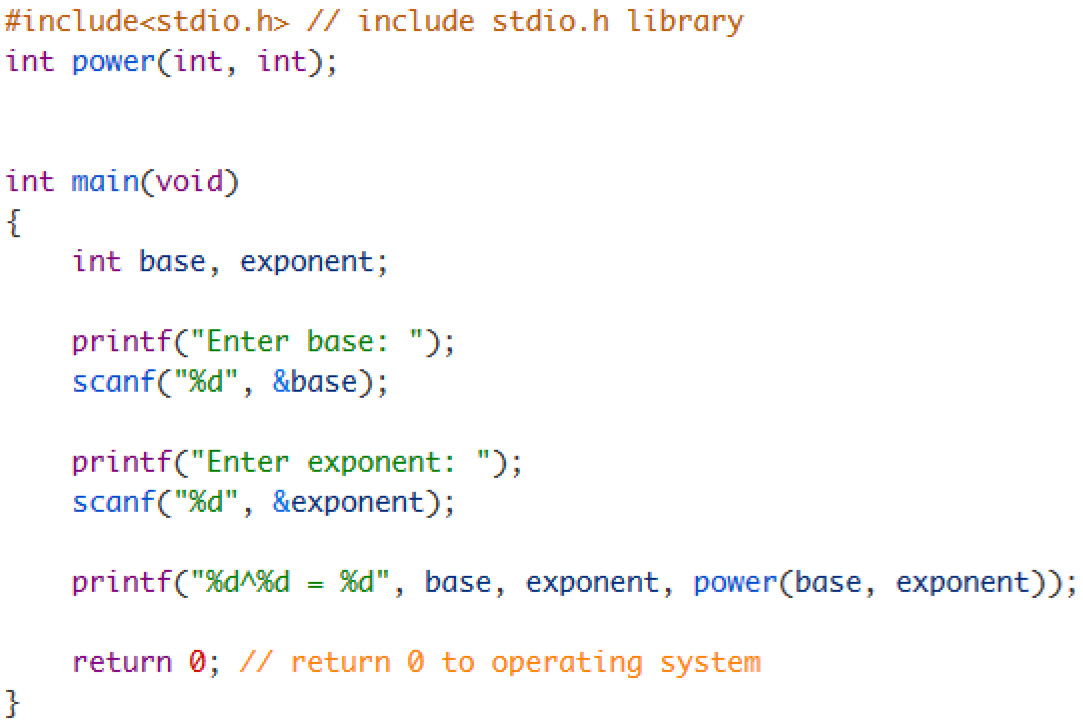
return 1

return 1

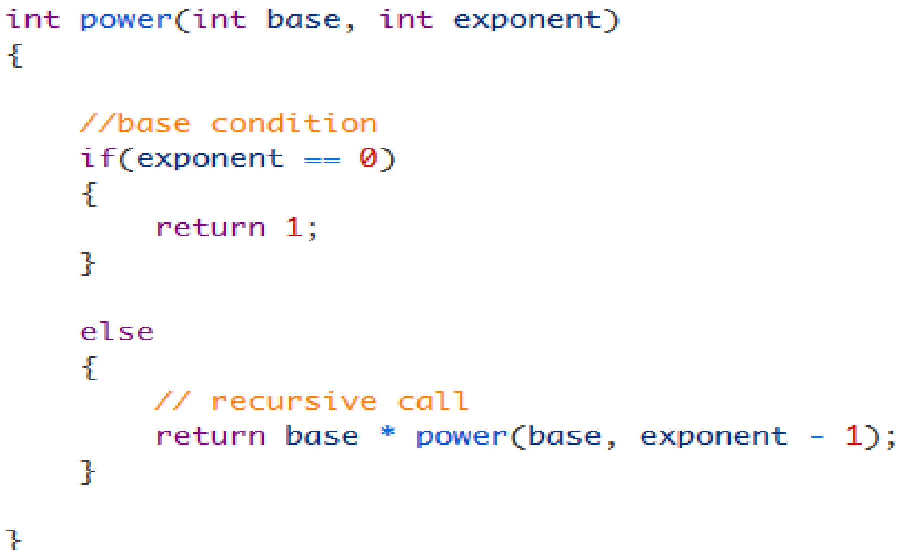
return 0

Evaluation of fibonacci(3)

C PROGRAM TO CALCULATE POWER USING RECURSION



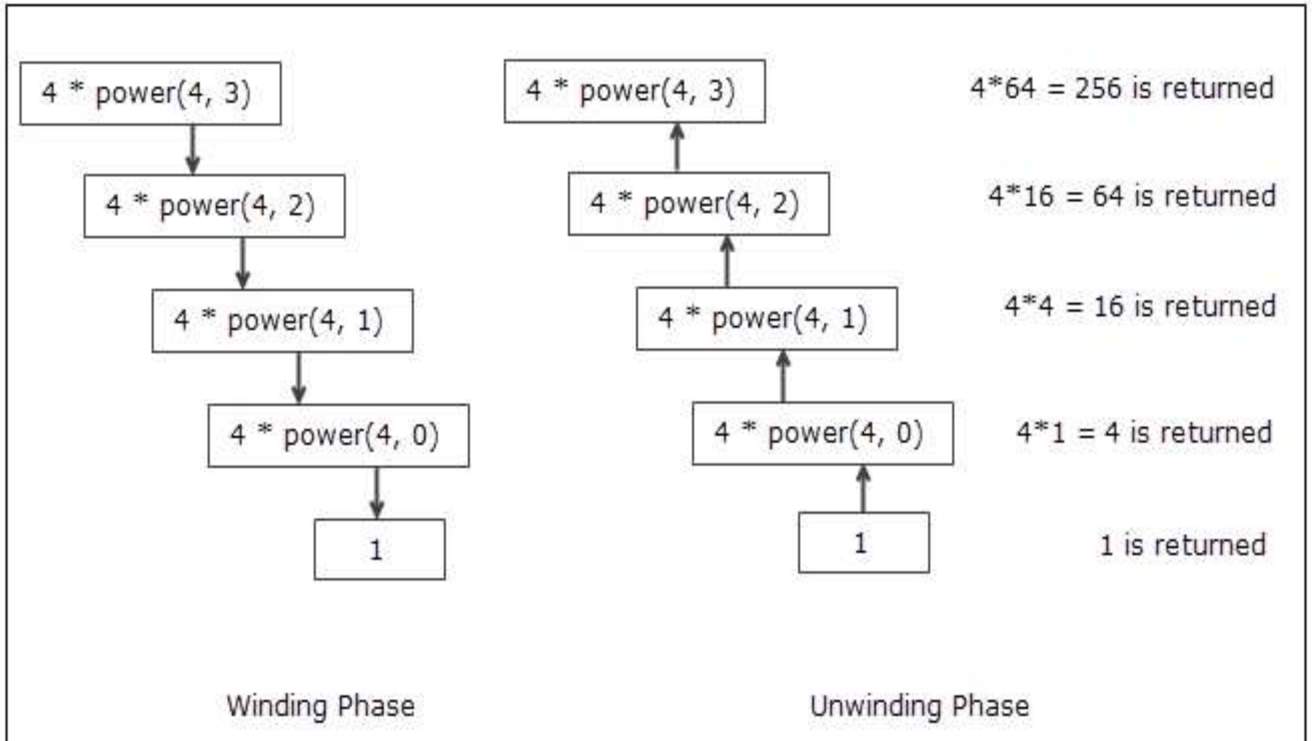
```c
#include<stdio.h> // include stdio.h library
int power(int, int);


int main(void)
{
    int base, exponent;

    printf("Enter base: ");
    scanf("%d", &base);

    printf("Enter exponent: ");
    scanf("%d", &exponent);

    printf("%d^%d = %d", base, exponent, power(base, exponent));

    return 0; // return 0 to operating system
}
```



```c
int power(int base, int exponent)
{

    //base condition
    if(exponent == 0)
    {
        return 1;
    }

    else
    {
        // recursive call
        return base * power(base, exponent - 1);
    }

}
```

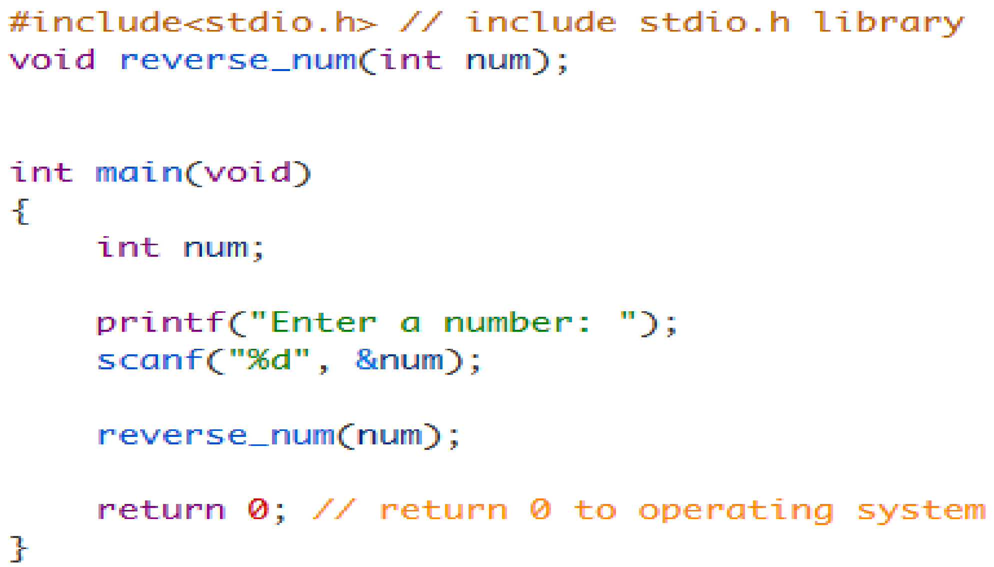
How it works

The following figure shows how the recursive evaluation of 4^4 takes place.

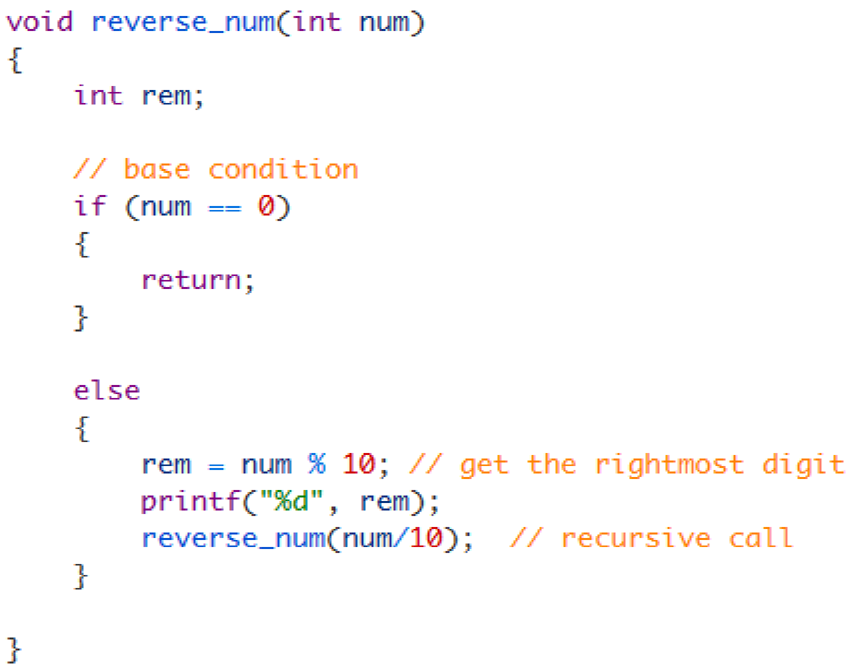


Calculation of 4^4 using recursion

C Program to to reverse the numbers of a digit using recursion.

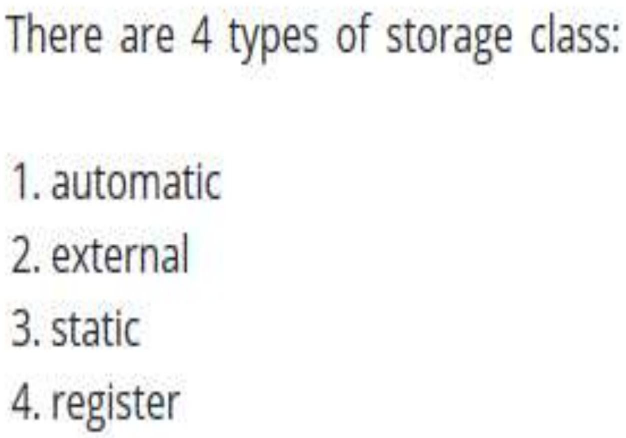


```c
#include<stdio.h> // include stdio.h library
void reverse_num(int num);


int main(void)
{
    int num;

    printf("Enter a number: ");
    scanf("%d", &num);

    reverse_num(num);

    return 0; // return 0 to operating system
}
```



```c
void reverse_num(int num)
{
    int rem;

    // base condition
    if (num == 0)
    {
        return;
    }

    else
    {
        rem = num % 10; // get the rightmost digit
        printf("%d", rem);
        reverse_num(num/10);  // recursive call
    }

}
```

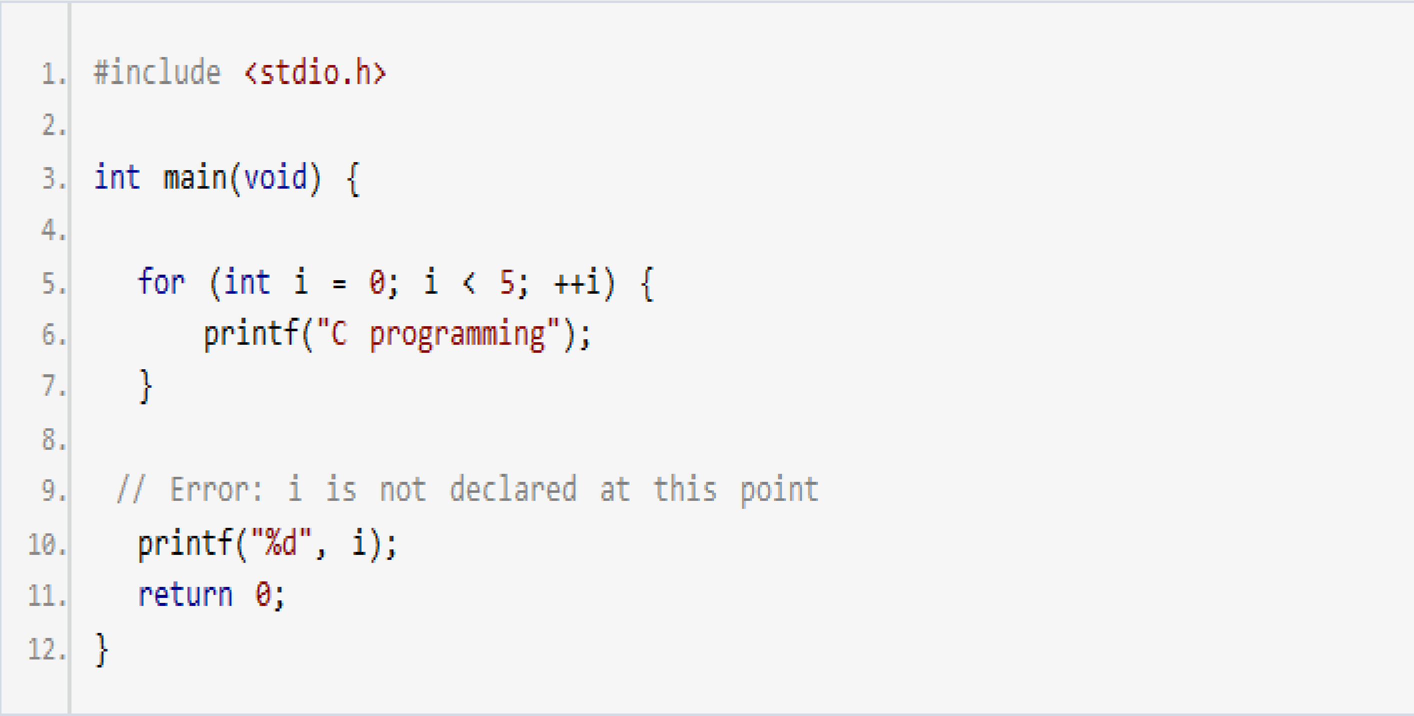
C Storage Class



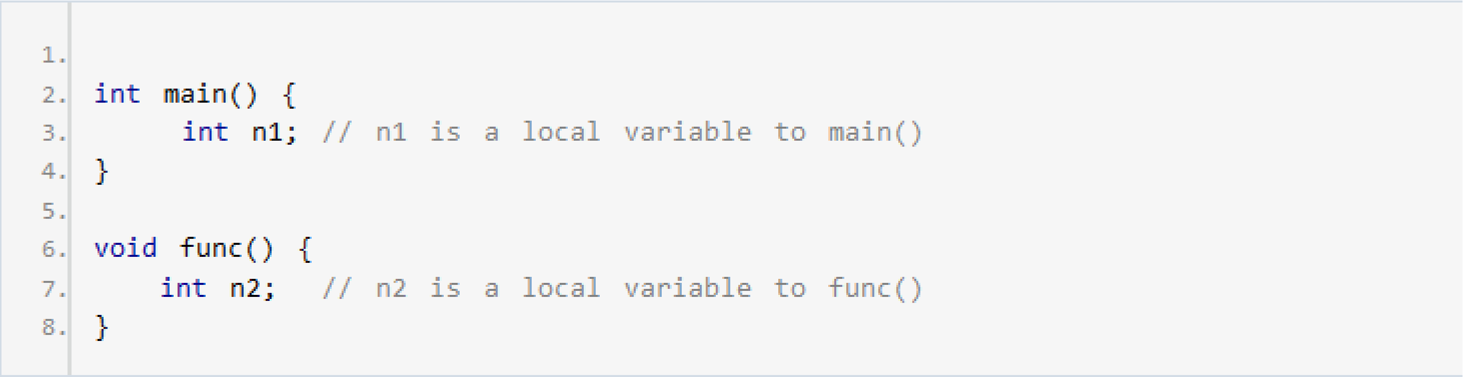
There are 4 types of storage class:

1. automatic
2. external
3. static
4. register

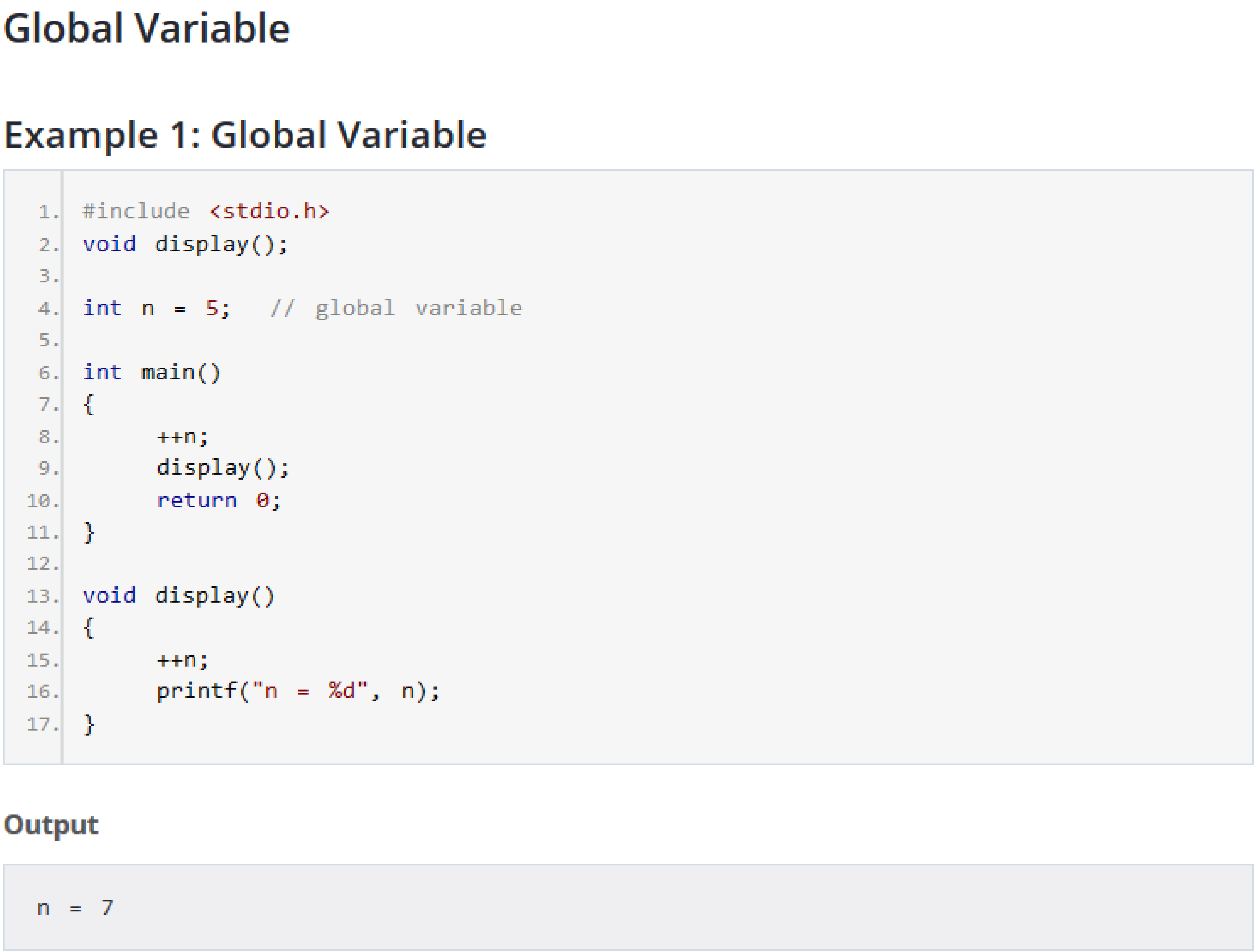
Local Variable



```c
#include <stdio.h>

int main(void) {

   for (int i = 0; i < 5; ++i) {
      printf("C programming");
   }

 // Error: i is not declared at this point
   printf("%d", i);
   return 0;
}
```



```c

int main() {
     int n1; // n1 is a local variable to main()
}

void func() {
    int n2;   // n2 is a local variable to func()
}
```



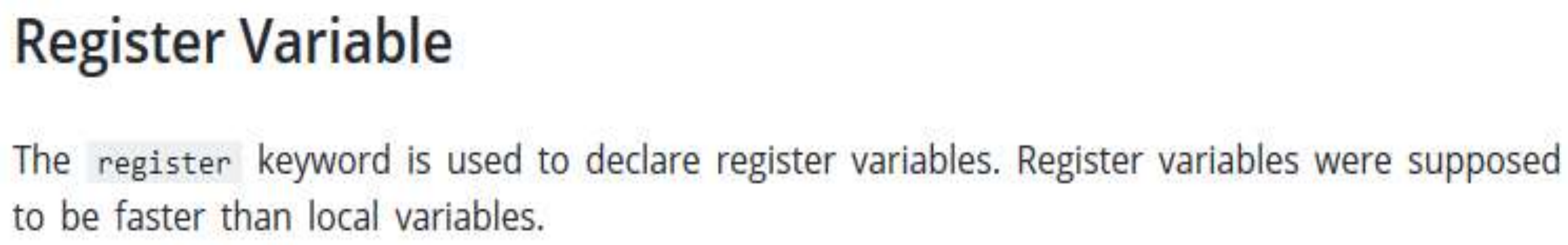
Global Variable

Example 1: Global Variable

```c
#include <stdio.h>
void display();

int n = 5;  // global variable

int main()
{
    ++n;
    display();
    return 0;
}

void display()
{
    ++n;
    printf("n = %d", n);
}
```

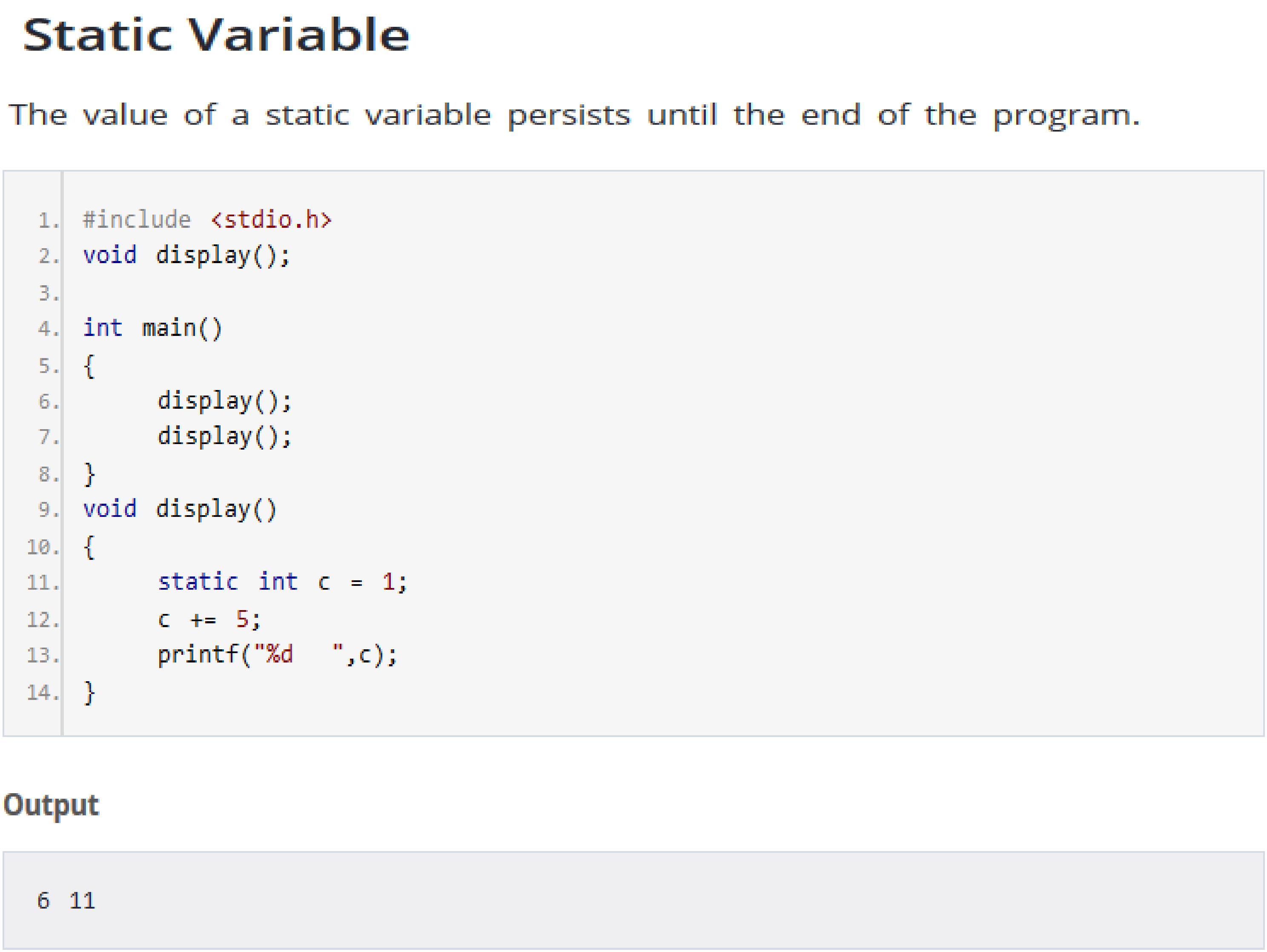
Output

```
n = 7
```



Register Variable

The `register` keyword is used to declare register variables. Register variables were supposed to be faster than local variables.



Static Variable

The value of a static variable persists until the end of the program.

```c
#include <stdio.h>
void display();

int main()
{
      display();
      display();
}
void display()
{
      static int c = 1;
      c += 5;
      printf("%d   ",c);
}
```

Output

```
6 11
```