CS 137

Loops in C

Fall 2025

Victoria Sakhnini

Table of Contents

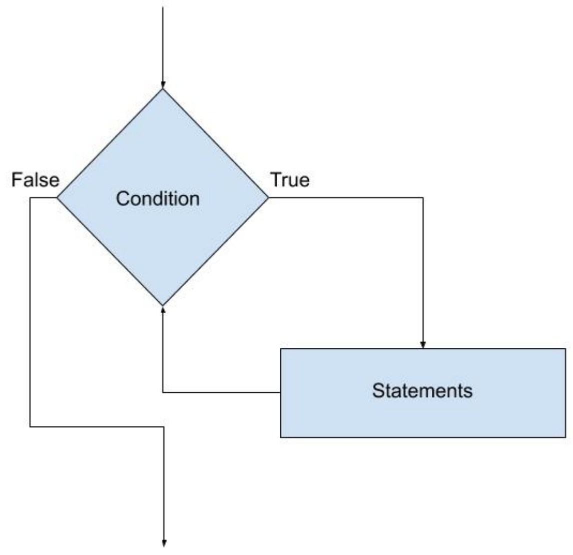
Introduction

One of the computers' key features is their ability to perform repeated calculations. `C` (as well as many programming languages) provides various control structures that allow the execution of a statement or group of statements multiple times.

The `while` statement

Syntax

```
while (condition) {
      statement(s);   // conditional code
}
```

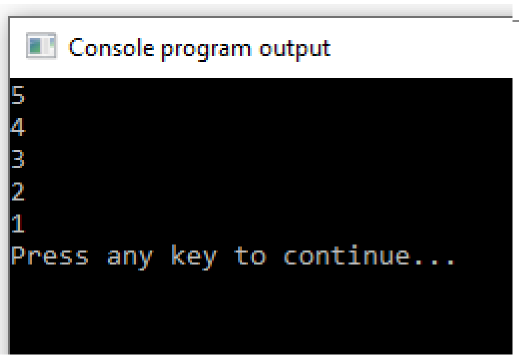


The condition may be any expression, and `true` is any nonzero value. The loop iterates while the condition is `true`.

When the condition becomes `false`, program control (that points to the following statement to be executed) passes to the line immediately following the loop.

Example:

```
1. #include <stdio.h>
2. int main(void)
3. {
4.   int i = 5;
5.   while (i != 0)
6.   {
7.        printf("%d\n", i);
8.        i = i - 1;
9.   }
10.  return 0;
11.  }
12.
```



```
5
4
3
2
1
Press any key to continue...
```

Tracing:

i	(i != 0)	output	comment
5			before the loop (line #4)
4	true	5	first iteration of while (lines 5-9)
3	true	4	second iteration of while
2	true	3	third iteration of while
1	true	2	fourth iteration of while
0	true	1	fifth iteration of while
	false		jump to after the while (line #10)

❓ What is the minimal number of times the statements in the loop (conditional code) could be executed[1]?

❓ What is the maximal number of times that the statements in the loop (conditional code) could be executed[2]?

🤔 What is the output of the following program? (To benefit the most, trace it manually before running the code to validate your answer.)

```c
#include <stdio.h>
int main(void)
{
    int i = 5;
    while (i != 0)
   {
      printf("%d\n", i);
   }
   return 0;
}

```

🤔 What is the output of the following program? (To benefit the most, trace it manually before running the code to validate your answer.)

```c
#include <stdio.h>
int main(void)
{
    int i = 5;
     while (i != 0)
          printf("%d\n", i);
          i = i - 1;

    return 0;
}

```

🤔 What is the output of the following program? (To benefit the most, trace it manually before running the code to validate your answer.)

```c
#include <stdio.h>
int main(void)
{
     int i = 5;
    while (i != 0){
          i = i - 1;
          printf("%d\n", i);
     }
     return 0;
}
```

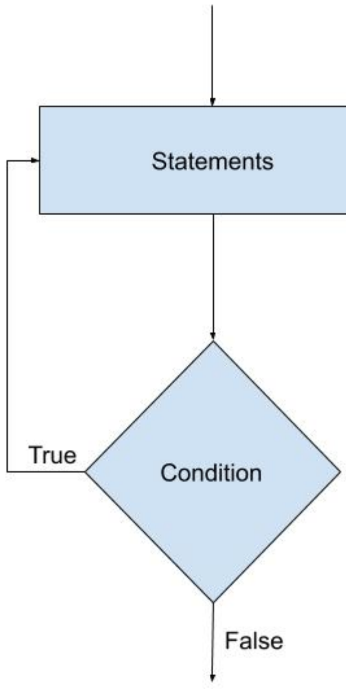
[1] zero times when the condition of while is `false` for the first time it is checked.
[2] infinite number of times if the condition of the while is `true` and never becomes `false`.

The do statement

Syntax

```
do {
    statement(s); // conditional code
} while(condition);
```

Notice that the condition appears at the end of the loop, so the statement(s) in the loop is (are) executed once before the condition is tested.



The last example can be written using the do statement as follows:

```c
#include <stdio.h>
int main(void)
{
    int i = 5;
    do
    {
        printf("%d\n", i);
        i = i - 1;
    } while (i != 0);
    return 0;
 }

```

When would you choose to use the `while` vs. `do` statement?

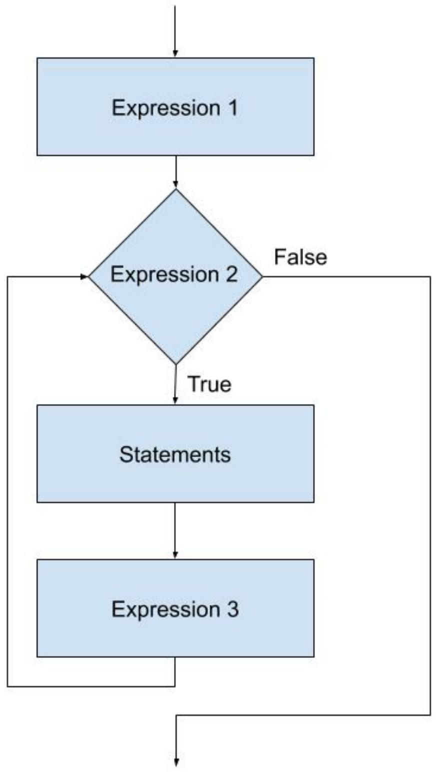
Can we convert any `while` to `do` statement and vice versa?

The `for` statement

Syntax

```
for (expr1; expr2; expr3) {
       statements;
}
```

- `expr1` is usually an initializer before the loop is executed
- `expr2` is usually a condition
- `expr3` is usually an incrementer/decrementer.
- In `C99` and beyond, `expr1` can initialize a new variable whose scope (valid, can be used) is only the `for` loop.



The last example can be written using a `for` statement as follows:

```c
#include <stdio.h>
int main(void)
{
    for (int i = 5; i != 0; i--)
    {
        printf("%d\n", i);
    }
    return 0;
}
```

What is the output of the following program? (To benefit the most, trace it manually before running the code to validate your answer.)

```c
#include <stdio.h>
int main(void)
{
    for (int i=1; i!=6; i++)
    {
        printf("%d\n", i);
    }
    return 0;
}
```

What is the output of the following program? (To benefit the most, trace it manually before running the code to validate your answer.)

```c
#include <stdio.h>
int main(void)
{
    for (int i=1; i!=6; i++)
        printf("%d\n", i);
         return 0;
}
```

What is the output of the following program? (To benefit the most, trace it manually before running the code to validate your answer[3].)

```c
#include <stdio.h>
int main(void)
{
    for (int i=1; i!=6; i++);
        printf("*");
    return 0;
}
```

What is the output of the following program? (To benefit the most, trace it manually before running the code to validate your answer.)

```c
#include <stdio.h>
int main(void)
{
    for (int i=1; i!=6; i++)
         printf("%d",i);
         printf("**");

    return 0;
}
```

[3] * (only one)

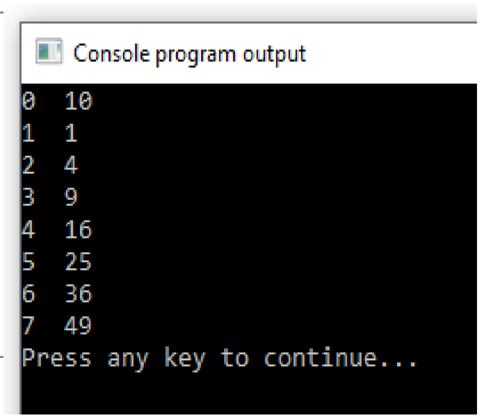
Multiple expressions

You can include multiple expressions in any of the fields of the `for` loop.

Example:

Be careful about the behaviour of multiple conditions in the middle part of `for`. Not recommended. Pay close attention to the output and try to understand why[4].

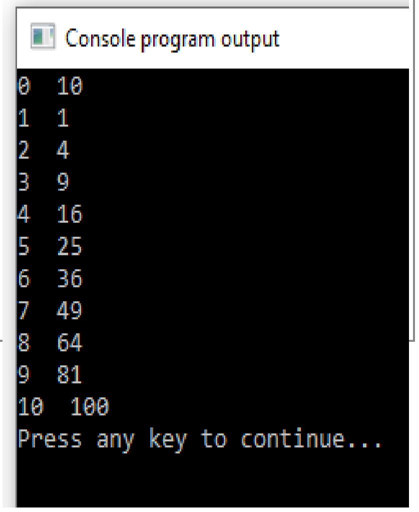
```c
#include <stdio.h>
int main(void)
{
    int i, j;
    for (i = 0, j = 10; i <= 10, j < 50; ++i, j = i * i)
    {
          printf("%d  %d\n", i, j);
    }
    return 0;
 }

```



```
0  10
1  1
2  4
3  9
4  16
5  25
6  36
7  49
Press any key to continue...
```

One more example:

```c
#include <stdio.h>
int main(void)
{
    int i, j;
    for (i = 0, j = 10; j < 50, i <= 10; ++i, j = i * i)
    {
          printf("%d  %d\n", i, j);
    }
    return 0;
 }

```



```
0  10
1  1
2  4
3  9
4  16
5  25
6  36
7  49
8  64
9  81
10  100
Press any key to continue...
```

Tip: The condition should be one expression; you can use `&&` or `||` though!

[4] This example was discussed to make a point that it is never appropriate to use comma to separate conditions. A condition is, by natural definition, an expression that is valuable specifically for its boolean result and not for its side effects (if any). The comma operator always ignores the results of all of its operands with the exception of the very last one. That immediately means that there's no meaningful way to specify a condition anywhere except for the very last position in the comma-separated sequence.
So for our example only the condition j<50 is executed

The **`break`** & **`continue`** statements

- Like `switch` statements, you can use the `break;` command to end a `while`, `do`, or `for` statement early.

- `C` also has a `continue` statement that allows you to exit the loop body early and go to just before the end of the loop body.

- For the `for` loop, a `continue` statement causes the conditional test and increment portions of the loop to execute. For the `while` and `do...while` loops, a `continue` statement causes the program control to pass the conditional test.

- There is also a `goto` command that can jump to any labeled part of the program.

- Using these statements almost always makes code more challenging to read, so most programmers avoid or limit the use of these statements as much as possible.

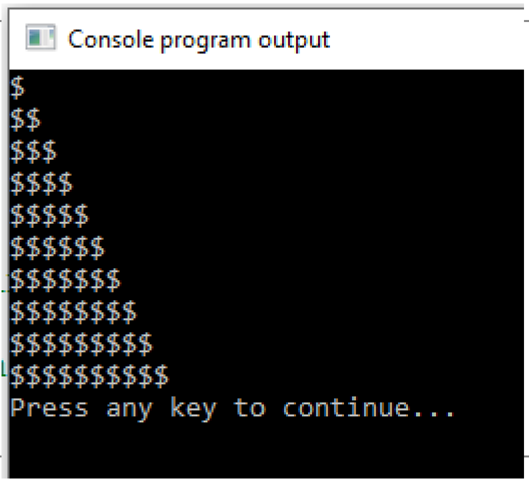
Nested loops

The C programming language allows us to use one loop inside another loop. The following examples illustrate the concept.

As an exercise, try to trace each example and understand how it works manually. Again, there is nothing new here; it is the same process as long as you remember that a loop is also a statement that must be executed before proceeding to the following statement.

Example1:

```c
1.  #include <stdio.h>
2.  int main(void)
3.  {
4.      int i, j;
5.      for (i = 1; i <= 10; i++)
6.      {
7.              for (j = 1; j <= i; j++)
8.              {
9.                  printf("$");
10.             }
11.             printf("\n");
12.     }
13.     return 0;
14.  }
```



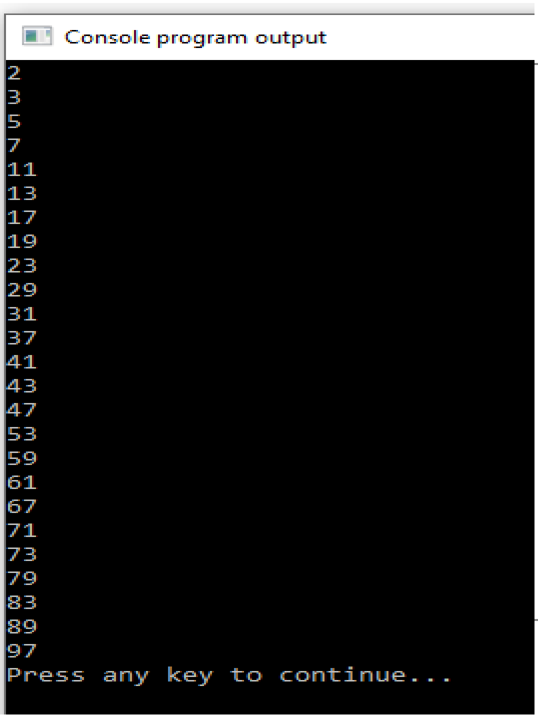
```
$
$$
$$$
$$$$
$$$$$
$$$$$$
$$$$$$$
$$$$$$$$
$$$$$$$$$
$$$$$$$$$$
Press any key to continue...
```

Tracing:

```
i = 1    (line 5)
i<=10 (true) (line 5)
        j runs from 1 to 1 => $ is printed once.  (lines7-10)
        \n  is printed. (line 11)
i++ => i = 2  (line 5)
i<=10 (true) (line 5)
        j runs from 1 to 2 => $ is printed twice   (lines7-10)
        \n  is printed. (line 11)
i++ => i = 3   (line 5)
i<=10 (true) (line 5)
        j  runs from 1 to 3 => $ printed 3 times   (lines7-10)
        \n is printed. (line 11)
.
.
.
i++ => i = 10   (line 5)
i<=10 (true) (line 5)
        j runs from 1 to 10 => $ is printed 10 times   (lines7-10)
        \n is printed. (line 11)
i++ => i = 11   (line 5)
i<=10 (false)
return 0;
```

Example2: printing prime numbers until 100

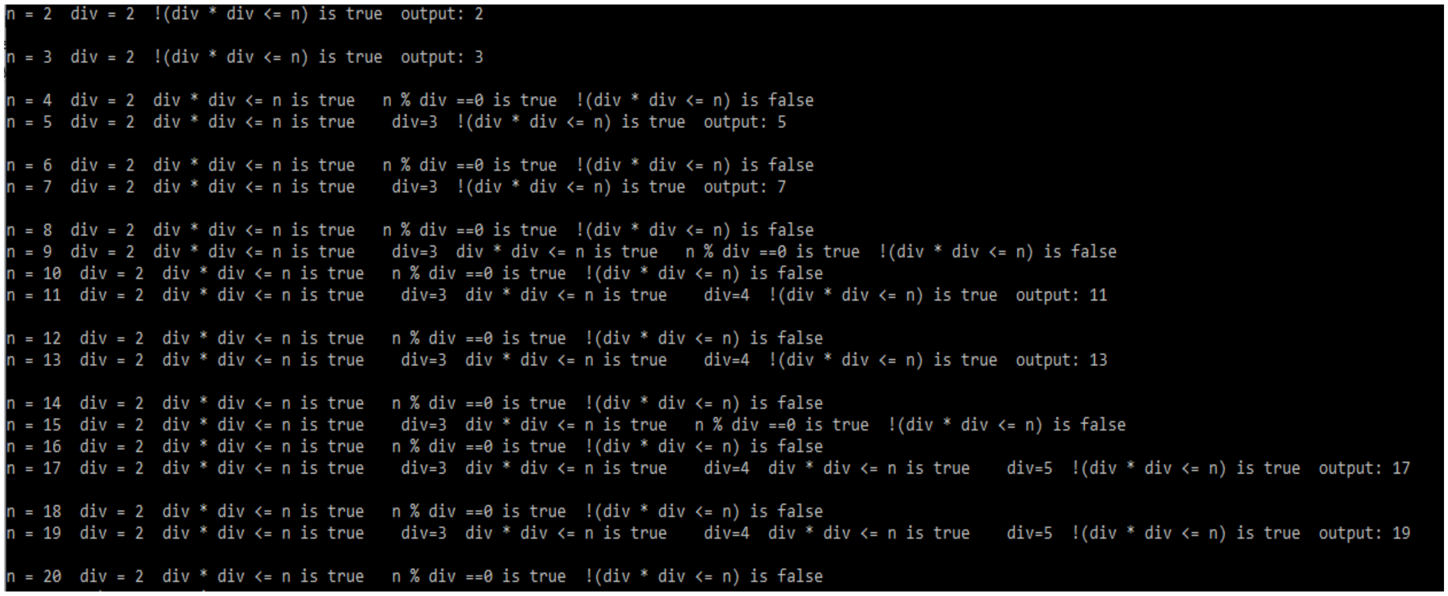
```c
#include <stdio.h>
int main(void)
{
    int div;
    for (int n = 2; n < 100; n++)
    {
        div = 2;
        while (div * div <= n)
        {
            if (n % div == 0)
                break;
            div++;
        }
        if (!(div * div <= n))
            printf("%d\n", n);
    }
    return 0;
}
```



Console program output

```
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
Press any key to continue...
```

Tracing (up to n=20):



```
n = 2  div = 2  !(div * div <= n) is true  output: 2

n = 3  div = 2  !(div * div <= n) is true  output: 3

n = 4  div = 2  div * div <= n is true   n % div ==0 is true  !(div * div <= n) is false
n = 5  div = 2  div * div <= n is true    div=3  !(div * div <= n) is true  output: 5

n = 6  div = 2  div * div <= n is true   n % div ==0 is true  !(div * div <= n) is false
n = 7  div = 2  div * div <= n is true    div=3  !(div * div <= n) is true  output: 7

n = 8  div = 2  div * div <= n is true   n % div ==0 is true  !(div * div <= n) is false
n = 9  div = 2  div * div <= n is true    div=3  div * div <= n is true   n % div ==0 is true  !(div * div <= n) is false
n = 10  div = 2  div * div <= n is true   n % div ==0 is true  !(div * div <= n) is false
n = 11  div = 2  div * div <= n is true    div=3  div * div <= n is true    div=4  !(div * div <= n) is true  output: 11

n = 12  div = 2  div * div <= n is true   n % div ==0 is true  !(div * div <= n) is false
n = 13  div = 2  div * div <= n is true    div=3  div * div <= n is true    div=4  !(div * div <= n) is true  output: 13

n = 14  div = 2  div * div <= n is true   n % div ==0 is true  !(div * div <= n) is false
n = 15  div = 2  div * div <= n is true    div=3  div * div <= n is true   n % div ==0 is true  !(div * div <= n) is false
n = 16  div = 2  div * div <= n is true   n % div ==0 is true  !(div * div <= n) is false
n = 17  div = 2  div * div <= n is true    div=3  div * div <= n is true    div=4  div * div <= n is true    div=5  !(div * div <= n) is true  output: 17

n = 18  div = 2  div * div <= n is true   n % div ==0 is true  !(div * div <= n) is false
n = 19  div = 2  div * div <= n is true    div=3  div * div <= n is true    div=4  div * div <= n is true    div=5  !(div * div <= n) is true  output: 19

n = 20  div = 2  div * div <= n is true   n % div ==0 is true  !(div * div <= n) is false
```

Extra examples

Let's see how we can solve problems using any of the three types of loops. Those are good exercises to learn how to manipulate each type of loop and to start sensing when to use each type and in which cases. Then, once you feel comfortable, try to solve those problems before looking at my code.

Problem 1

Write a program that requests any integer and prints the sum of the digits to the screen. For example, if the user enters `1234`, the answer 10 (=1+2+3+4) is printed.

Below is a solution written in three different ways.

```c
#include <stdio.h>
int main(void)
{
     int num;
     printf("Enter a positive integer.\n");
    scanf("%d", &num);
    int sum = 0;
    while (num > 0)
    {
          sum += num % 10;
          num = num / 10;
    }
   printf("The sum of digits is: %d\n", sum);
 }
```

```c
#include <stdio.h>
int main(void)
{
     int num;
     printf("Enter a positive integer.\n");
     scanf("%d", &num);
     int sum = 0;
     do
     {
          sum += num % 10;
          num = num / 10;
     } while (num > 0);
     printf("The sum of digits is: %d\n", sum);
 }
```

```c
#include <stdio.h>
int main(void)
{
     int num, sum;
     printf("Enter a positive integer.\n");
     scanf("%d", &num);
     for (sum = 0; num > 0; num = num / 10)
     {
          sum += num % 10;

     }
     printf("The sum of digits is: %d\n", sum);
 }
```

Problem 2

Write a program that asks the user for an integer and determines if it is a prime number. Print "Not Prime" if it is not prime and "Prime" otherwise. Recall: A prime number is an integer greater than 1 whose only positive divisors are 1 and itself.

Idea 1: Count all the divisors of the given number; if it is 2, we have a prime number.

```c
#include <stdio.h>
int main(void){
    int n, div = 1, cnt = 0;
    scanf("%d", &n);
    if (n <= 1)
          printf("Not Prime\n ");
    else {
         while (div <= n){
            if (n % div == 0)
                  cnt++;
            div++;
         }
         if (cnt == 2)
                printf("Prime\n");
         else
                printf("Not Prime\n");
     }
     return 0;
  }
```

comment	n	div	cnt	div<=n	n%div == 0
before the while	15	1	0		
iteration #1		2	1	true	true
iteration #2		3		true	false
		4	2	true	true
		5		true	false
		. . .			
iteration #14		15		true	false
iteration #15		16		true	false
out of the loop, cnt==2 is false thus the program prints Not Prime				false	

There are a few ways we can make this code run faster (fewer calculations)

- If we have a non-prime number, it has a divisor between the number itself and 1 (not including endpoints). If we find this, we can immediately break the code and print not prime
- If it has such a divisor d, there must be one less than $\sqrt{n}$ where n is the given number for one of d or n/d is bounded above by $\sqrt{n}$
- Let's rewrite the code to account for these changes.

Idea 2: Search for a single divisor, not `1` or `n`, and if we find one, return `false`.

```c
#include <stdio.h>
int main(void){
    int n, div = 2;
    scanf("%d", &n);
    if (n <= 1)
          printf("Not Prime\n");
    else {
         while (div * div <= n){
             if (n % div == 0)
                          break;
             div++;
         }
          if (div * div <= n)
              printf("Not Prime\n");
         else
              printf("Prime\n");
    }
    return 0;
  }
```

comment	n	div	div*div <=n	n%div == 0
before the while	15	2		
iteration #1		3	true	false
iteration #2			true	true*
* `break` is reached => leaving the loop executed `if (div * div <=n)` will print `Not Prime`				

The Euclidean Algorithm

How do we reduce $\frac{1080}{1920}$?

Similarly, how do we compute the greatest common divisor of `1080` and `1920`, denoted by `gcd(1080, 1920)`?

It turns out geometrically, this has a neat interpretation.

Watch this video https://www.youtube.com/watch?v=AVrtH6m2wcU (1:49 minutes)

The video shows that at each step, we are dividing the larger number by the smaller number using the division algorithm (more on this in MATH 135). When Euclid performed this, he repeated subtraction (a division).
In fact, we have

$$1920 = 1080(1) + 840$$
$$1080 = 840(1) + 240$$
$$840 = 240(3) + 120$$
$$240 = 120(2) + 0$$

When dividing a number by another positive integer, the remainder seems to be the most important value at each step.

```c
#include <stdio.h>
int main(void)
{
     int a = 0, b = 0, r = 0;
    printf("Enter the first integer: ");
    scanf("%d", &a);
    printf("Enter the second integer: ");
    scanf("%d", &b);
    while (b != 0)
    {     // Note : Could also use while (b)
          r = a % b;
          a = b;
          b = r;
    }
    printf("The result is: %d\n", a);
    return 0;
 }
```

tracing for the input: `1920 1080`

Iterations of the while loop:

Iteration	a	b	r
0	1920	1080	0
1	1080	840	840 (=1920%1080)
2	840	240	240 (=1080%840)
3	240	120	120 (=840%240)
4	120	0	0 (=240%120)

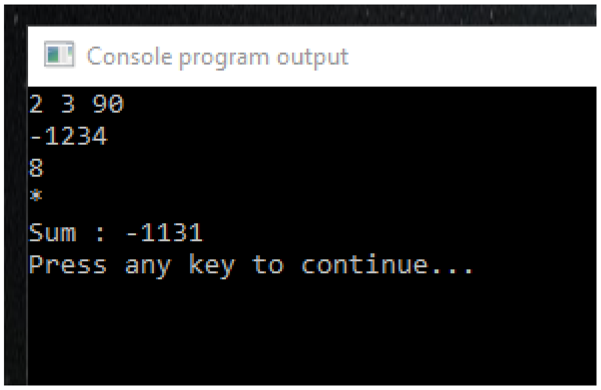
Back to `scanf()`

Imagine we want to read an arbitrary number of integers from the user and output the sum of those numbers. So, if the user inputs `5 4 10 8`, we will print out `27`. So far, we've only known how to read a preset number and order of things.

So far, we've only talked about how `scanf` modifies the variables we read into, but `scanf` is a function with a returned value. `scanf` returns the number of items it correctly reads. We can use that knowledge with a `while` loop to read all the user's integers.

Solution:

```c
#include <stdio.h>
int main(void)
{
    int d = 0;
     int sum = 0;
  // While read is successful
    while (scanf("%d", &d) == 1)
    {
          sum += d;
    }
    printf("Sum : %d\n", sum);
    return 0;
 }
```


Console program output

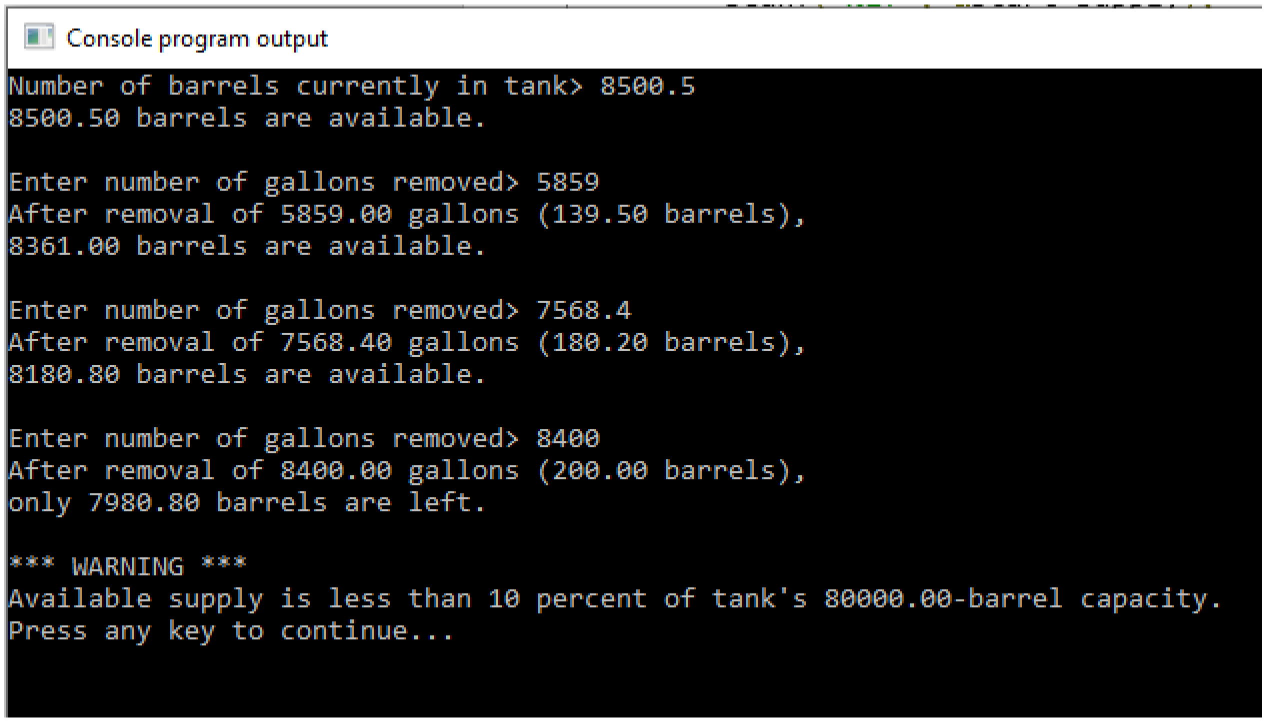
```
2 3 90
-1234
8
*
Sum : -1131
Press any key to continue...
```

Tracing

d	sum	input	scanf (“%d”, &d) == 1	output
0				
	0			
2		2	true	
	2			
3		3	true	
	5			
90		90	true	
	95			
-1234		-1234	true	
	-1139			
8		8	true	
	-1131			
		*	false	
				-1131

Additional Examples

```
/*
 * Monitor gasoline supply in the storage tank. Issue warning when supply
 * falls below MIN PCT % of tank capacity.
 */

#include <stdio.h>

/* constant macros */
#define CAPACITY 80000.0 /* number of barrels tank can hold   */
#define MIN_PCT 10       /* warn when supply falls below this percent of capacity      */
#define GALS_PER_BRL 42.0        /* number of U.S. gallons in one barrel */

int main(void)
{
        double start_supply,     /* initial supply in barrels       */
               min_supply,       /* minimum number of barrels left without warning     */
               current,  /* current supply in barrels       */
               remov_gals,       /* amount of current delivery in    */
               remov_brls;       /*   barrels and gallons       */

        /* Compute minimum supply without warning       */
        min_supply = MIN_PCT / 100.0 * CAPACITY;

        /* Get initial supply and subtract amounts removed as long as
           minimum supply remains       */
        printf("Number of barrels currently in tank> ");
        scanf("%lf", &start_supply);
        for (current = start_supply; current >= min_supply; current -= remov_brls)
        {
                printf("%.2f barrels are available.\n\n", current);
                printf("Enter number of gallons removed> ");
                scanf("%lf", &remov_gals);
                remov_brls = remov_gals / GALS_PER_BRL;
                printf("After removal of %.2f gallons (%.2f barrels),\n", remov_gals,
remov_brls);
        }

        /* Issue warning        */
        printf("only %.2f barrels are left.\n\n", current);
        printf("*** WARNING ***\n");
        printf("Available supply is less than %d percent of tank's ", MIN_PCT);
        printf("%.2f-barrel capacity.\n", CAPACITY);

        return (0);
}
```


Console program output

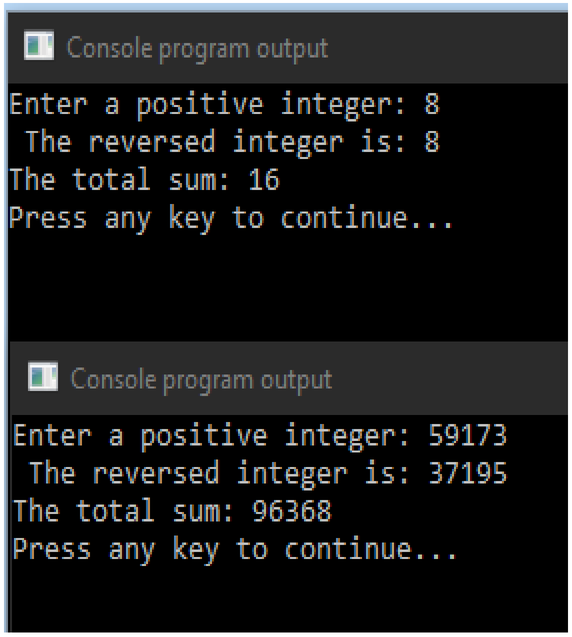
```
Number of barrels currently in tank> 8500.5
8500.50 barrels are available.

Enter number of gallons removed> 5859
After removal of 5859.00 gallons (139.50 barrels),
8361.00 barrels are available.

Enter number of gallons removed> 7568.4
After removal of 7568.40 gallons (180.20 barrels),
8180.80 barrels are available.

Enter number of gallons removed> 8400
After removal of 8400.00 gallons (200.00 barrels),
only 7980.80 barrels are left.

*** WARNING ***
Available supply is less than 10 percent of tank's 80000.00-barrel capacity.
Press any key to continue...
```

Extra Practice Problems

[Solve the question and test your solutions before you check my suggested ones.]

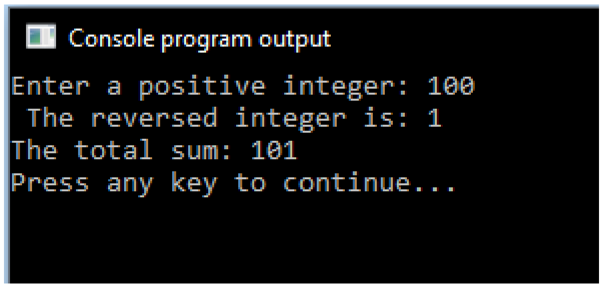
1) Write a program that reads a positive integer and creates a new value with the same digits as the input but in reverse order and prints the new integer in addition to the total sum of both integers[i].

Examples:



Console program output
```
Enter a positive integer: 8
 The reversed integer is: 8
The total sum: 16
Press any key to continue...
```

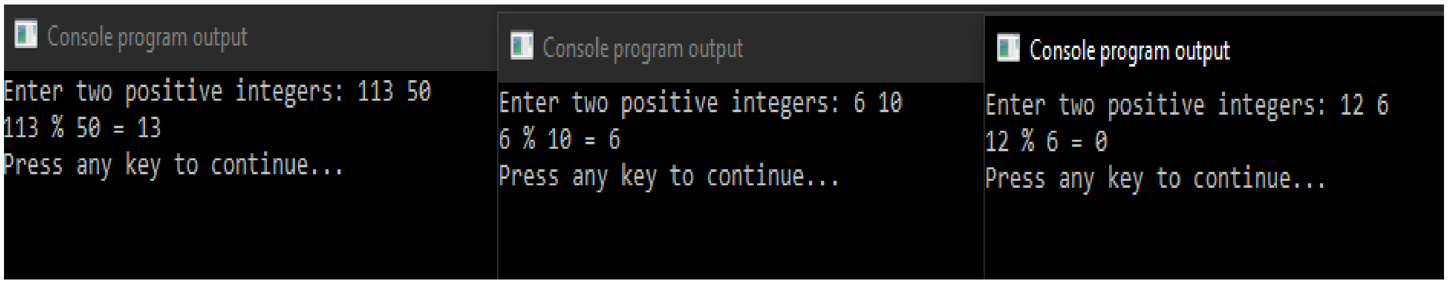
Console program output
```
Enter a positive integer: 59173
 The reversed integer is: 37195
The total sum: 96368
Press any key to continue...
```



Console program output
```
Enter a positive integer: 100
 The reversed integer is: 1
The total sum: 101
Press any key to continue...
```

2[ii]) Write a program that reads two positive integers, `a` and `b`, and prints the result of `a%b` without using `%` or `/`

Examples:

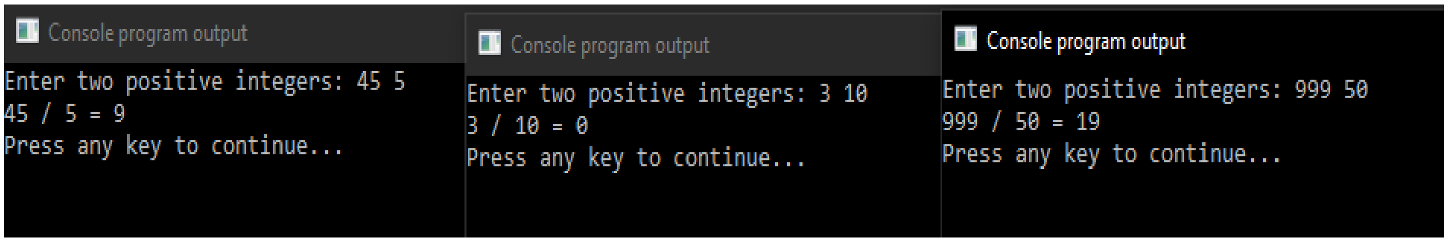


Console program output
```
Enter two positive integers: 113 50
113 % 50 = 13
Press any key to continue...
```

Console program output
```
Enter two positive integers: 6 10
6 % 10 = 6
Press any key to continue...
```

Console program output
```
Enter two positive integers: 12 6
12 % 6 = 0
Press any key to continue...
```

3) Write a program that reads two positive integers, `a` and `b`, and prints the result of `a/b` without using `%` or `/`



Console program output
```
Enter two positive integers: 45 5
45 / 5 = 9
Press any key to continue...
```

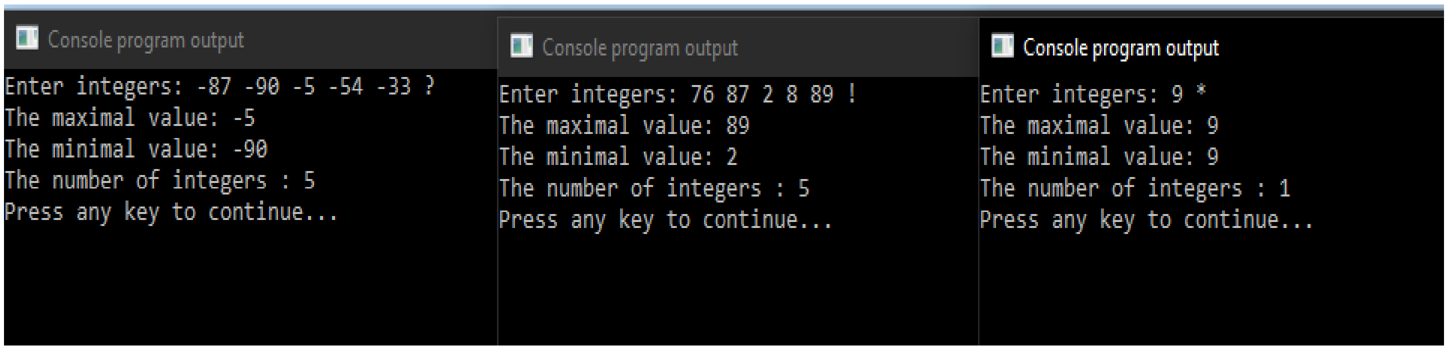
Console program output
```
Enter two positive integers: 3 10
3 / 10 = 0
Press any key to continue...
```

Console program output
```
Enter two positive integers: 999 50
999 / 50 = 19
Press any key to continue...
```

4[iii]) Write a program that reads an arbitrary number of integers (at least one) from the user and prints the following:

- The maximal value that was entered
- The minimal value that was entered
- The number of entered integers

Test cases:



```
Enter integers: -87 -90 -5 -54 -33 ?
The maximal value: -5
The minimal value: -90
The number of integers : 5
Press any key to continue...
```

```
Enter integers: 76 87 2 8 89 !
The maximal value: 89
The minimal value: 2
The number of integers : 5
Press any key to continue...
```

```
Enter integers: 9 *
The maximal value: 9
The minimal value: 9
The number of integers : 1
Press any key to continue...
```

5) Use the Euclidean algorithm to find the GCD of 1350 and 2275.

6) Write a program that reads an integer (assume it is positive) and prints whether it is a perfect number. A number (n) is considered perfect if the sum of all its divisors (other than itself) is n. For example, 6 is a perfect number since 1 + 2 + 3 = 6.

7) Write a program `butterfly.c`, that reads a positive number as the size of the butterfly and prints numbers in a butterfly shape as illustrated below.

```
butterfly(2) prints:
  1     1
  1  2  1
  1     1
```

```
butterfly(5) prints:
  1                       1
  1  2                 2  1
  1  2  3           3  2  1
  1  2  3  4     4  3  2  1
  1  2  3  4  5  4  3  2  1
  1  2  3  4     4  3  2  1
  1  2  3           3  2  1
  1  2                 2  1
  1                       1
```

You must use the format specifier `"%3d"` to print each integer.
For example: `printf("%3d", x);`
You must print a `newline` at the end of each line.

8) Write a program `draw_xbox.c` to read a positive number as the size of the box and draw (print) a fancier box with an"`X`" inside of the box.
1) The size of the box must be positive.
2) The dimensions of the xbox will be (2*size + 1).
For example, when size = 1, it prints a 3 x 3 box that appears as:

```
+-+
|X|
+-+
```

When size = 3, it prints a 7 X 7 box that appears as:

```
+-----+
|\   /|
| \ / |
|  X  |
| / \ |
|/   \|
+-----+
```

By reviewing these two examples carefully, you should be able to draw(print) an xbox of any positive size

9)

Input an integer N, find the sum of the following series up to N:

$$\sum_{i=1}^{\infty} = \frac{2^{i-1}}{(i+1)!} = \frac{1}{2!} + \frac{2}{3!} + \frac{4}{4!} + \frac{8}{5!} + \ldots$$

10) You are developing a program for a robotics competition with an advanced robot. The robot can move in four directions: Up, Down, Left, and Right. It starts at the origin (0, 0) and receives a sequence of commands as input. The commands are represented as characters: 'U' for Up, 'D' for Down, 'L' for Left, and 'R' for Right.

Additionally, there are obstacles in the robot's path. Obstacles are represented as coordinates (X, Y), and the robot cannot move to a location with an obstacle. The robot should also stay within a specified boundary, defined as a rectangular region with coordinates (X1, Y1) to (X2, Y2).
Write a C program that takes the following inputs:

- The boundary coordinates (X1, Y1, X2, Y2).
- The number of obstacles, N.
- N pairs of obstacle coordinates.
- A sequence of commands as a string.
- The character 'E' to signify the end of input.

Your program should execute the commands while avoiding obstacles and staying within the specified boundary. If a command would take the robot outside the boundary or into an obstacle, it should ignore that command and move to the next one. Finally, print the robot's final position.

<u>Example Input:</u>

```
Boundary: (0, 0) to (4, 4)
Number of obstacles: 3
Obstacles: (1, 2), (3, 3), (2, 0)
Commands: "URDLLERERUUDE"
```

<u>Output :</u> `(0, 0)`

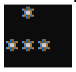
<u>Demo :</u>

```
Enter boundary coordinates (X1, Y1, X2, Y2): 0 0 4 4
Enter the number of obstacles: 3
Enter obstacle coordinates (X Y): 1 2
3 3
2 0
Enter a sequence of commands (E to end): URDLLERERUUDE
Final position of the robot: (0, 0)
```

Your program should execute the commands while avoiding obstacles and staying within the boundary and then print the robot's final position.

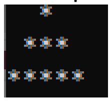
11) Create a program `triangle.c`, which reads a positive integer `n` and prints a triangle of `n` rows as illustrated in the following examples:

For input 2:



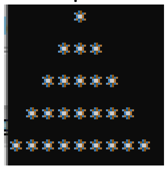
```
 *
***
```

For input 3:



```
  *
 ***
*****
```

For input 5:



```
    *
   ***
  *****
 *******
*********
```

12) Create a C program `simplex.c` that contains the following:

a) `int triangle(int n);`
The function returns the `n`th triangle number.
Assumptions: `n` is a positive integer.
Restrictions: You must NOT use any of `% / *` in your solution.
Definition: The `n`th triangle number is equal to the sum of the first `n` positive integers.

b) `int tetrahedral(int n);`
The function returns `n`th tetrahedral number.
Assumptions: `n` is a positive integer.
Restrictions: You must NOT use any of `% / *` in your solution.
Definition: The `n`th tetrahedral number is equal to the sum of the first `n` triangle numbers.

The following code will help you with testing

```
#include <stdio.h>
#include <assert.h>
// The rest of the Code is here
int main(void)
{
assert(triangle(2)==3);
assert(triangle(3)==6);
assert(triangle(5)==15);
assert(tetrahedral(2)==4)
assert(tetrahedral(3)==10);
}
```

13) Assume that you are given a picture containing certain amounts of red, green, and blue color. The picture has an 8-digit serial number encoding to represent the color constituents. Red is represented by 1, green by 2, and blue by 3. The 3 numbers can be in any order and not necessarily clumped together. Write a function `rgb` that takes in a serial number and prints the fraction of each colour for a picture.
example input : `11231123`
output: `R = 4/8, G = 2/8, B =2/8`

13) Write a function `void hollow_square(int w);` that prints a hollow square with side length `w` with diagonals. The square should have a boundary of stars (*), but its diagonals (top-left to bottom-right and top-right to bottom-left) should also contain stars. Inside the square (other than diagonals and edges), there should be spaces.

Example:

`hollow_square(5)` prints:

```
* * * * *
* *   * *
*   *   *
* *   * *
* * * * *
```

14) Write a function that takes a `long int` and returns an int <10 following the process below.
Add the digits of the `long int`; if the answer is >9, repeat until the total sum of digits is <10.
Example: `fun(12399987654020)` returns `2` because
`1+2+3+9+9+9+8+……+2+0` is `65`
`6+5=11`
`1+1` is `2`

Answers

i

```
#include <stdio.h>
int main(void)
{
    int num, val, inp;
    printf("Enter a positive integer: ");
    scanf("%d", &inp);
    val = 0;
    num = inp;
    while (num > 0)
    {
        val = val * 10 + num % 10;
        num = num / 10;
    }
    printf(" The reversed integer is: %d\n", val);
    printf("The total sum: %d\n", val + inp);
    return 0;
}
```

ii

```
#include <stdio.h>
int main(void)
{
    int a, b;
    printf("Enter two positive integers: ");
    scanf("%d %d", &a, &b);
    printf("%d %% %d = ", a, b);
    while (a >= b)
    {
        a = a - b;
    }
    printf("%d\n", a);
    return 0;
}
```

iii

```c
#include <stdio.h>
int main(void)
{
    int num, min, max, cnt;

    printf("Enter integers: ");
    scanf("%d", &num);
    min = max = num;
    cnt = 1;
    while (scanf("%d", &num) == 1)
    {
        if (num > max)
            max = num;
        if (num < min)
            min = num;
        cnt++;
    }

    printf("The maximal value: %d\n", max);
    printf("The minimal value: %d\n", min);
    printf("The number of integers : %d\n", cnt);
    return 0;
}
```