CS 137

Programming Principles

Chapter 3

Loops in C

Victoria Sakhnini

Table of Contents

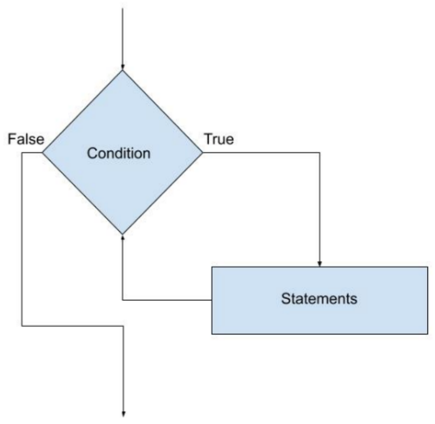
Introduction

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 passes to the line immediately following the loop.

Example

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

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?

Zero times — when the condition of while is false the first time it is checked.

? What is the maximal number of times the statements in the loop (conditional code) could be executed?

An infinite number of times — if the condition of the while is true and never becomes false.

Tricky: What is the output of the following program? (Trace manually before running.)

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

Tricky: What is the output of the following program? (Trace manually before running.)

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

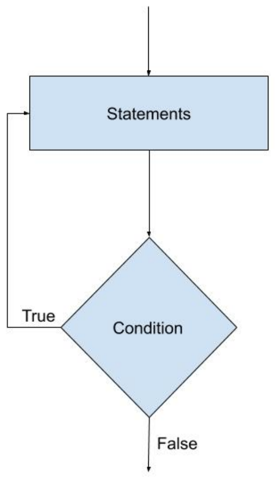
? What is the output of the following program? (Trace manually before running.)

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

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 rewritten using the do statement:

```
#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?

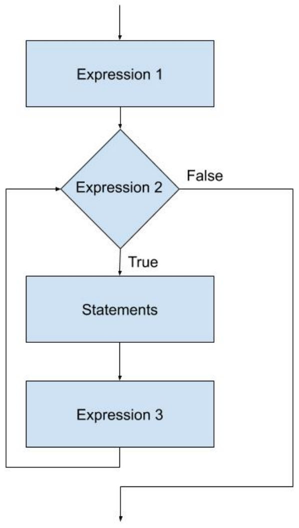
? Can we convert any while to a do statement and vice versa?

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 is only the for loop.



The last example rewritten using a for statement:

```
#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? (Trace manually before running.)

```
#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? (Trace manually before running.)

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

Tricky: What is the output of the following program? (Trace manually before running.)

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

* (only one)

? What is the output of the following program? (Trace manually before running.)

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

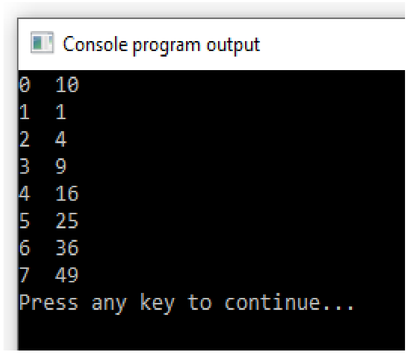
Multiple Expressions

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

⚠ 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.

Example 1:

```
#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;
}
```

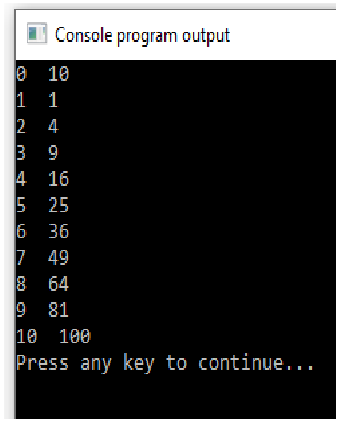


```
0  10
1  1
2  4
3  9
4  16
5  25
6  36
7  49
Press any key to continue...
```

One more example:

```
#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;
}
```

Tip: The condition should be one expression; you can use && or || though! The comma operator always ignores all operands (assuming no effect) except the very last one — so only the last condition in a comma-separated list is actually evaluated.



```
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...
```

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, `continue` causes program control to pass to 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 their use.

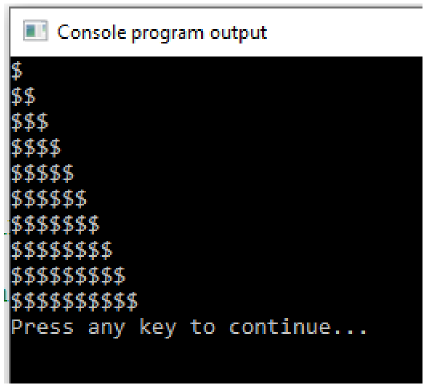
Nested Loops

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

As an exercise, trace each example manually. There is nothing new here — a loop is also a statement that must be executed before proceeding to the following statement.

Example 1

```
#include <stdio.h>
int main(void)
{
    int i, j;
    for (i = 1; i <= 10; i++)
    {
        for (j = 1; j <= i; j++)
        {
            printf("$");
        }
        printf("\n");
    }
    return 0;
}
```



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

Tracing:

i = 1: loop runs once → $ printed once, then \n

i = 2: loop runs twice → $$ printed, then \n

i = 3: loop runs 3 times → $$$ printed, then \n

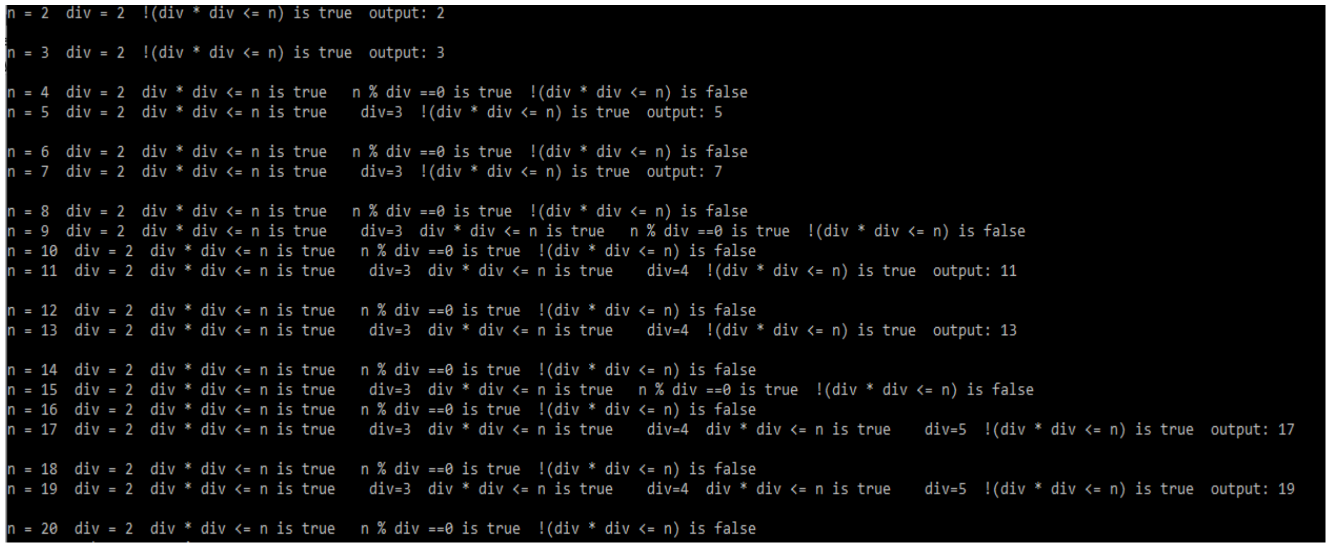
...

i = 10: loop runs 10 times → $$$$$$$$$$ printed, then \n

i = 11: i<=10 is false → exit

Example 2 — Printing Prime Numbers up to 100

```
#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;
}
```



```
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

The following problems show how to solve the same task using each of the three loop types. Try to solve each problem before looking at the code.

Problem 1 — Sum of Digits

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

Solution using while:

```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);
}
```

Solution using do:

```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);
}
```

Solution using for:

```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 — Primality Test

Write a program that asks the user for an integer and determines if it is prime. 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 divisors: if there are exactly 2, the number is prime.

```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;
}
```

Tracing for n = 15:

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 loop, cnt==2 is false → prints Not Prime				false	

There are a few ways to make this code run faster:

- If we have a non-prime number, it has a divisor between 1 and itself (exclusive). If we find one, we can immediately break and print Not Prime.
- If it has a divisor d, there must be one ≤ √n, since one of d or n/d is bounded above by √n.

Idea 2 — Search for a single divisor (up to √n); if found, the number is not prime.

```
#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;
}
```

Tracing for n = 15:

Comment	n	div	div*div<=n	n%div == 0
before the while	15	2		
iteration #1		3	true	false
iteration #2 — break is reached			true	true
executed if (div*div<=n) → prints Not Prime				

The Euclidean Algorithm

How do we reduce 1080/1920? Similarly, how do we compute gcd(1080, 1920)?

The Euclidean algorithm repeats the division algorithm — the remainder is the key value at each step:

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

```
#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)  // could also use: while (b)
    {
        r = a % b;
        a = b;
        b = r;
    }
    printf("The result is: %d\n", a);
    return 0;
}
```

Back to scanf()

Imagine reading an arbitrary number of integers and printing their sum. For example, if the user inputs `5 4 10 8`, print 27.

scanf is a function with a return value: it returns the number of items it correctly reads. We can use this with a while loop to read all integers.

```
#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;
}
```

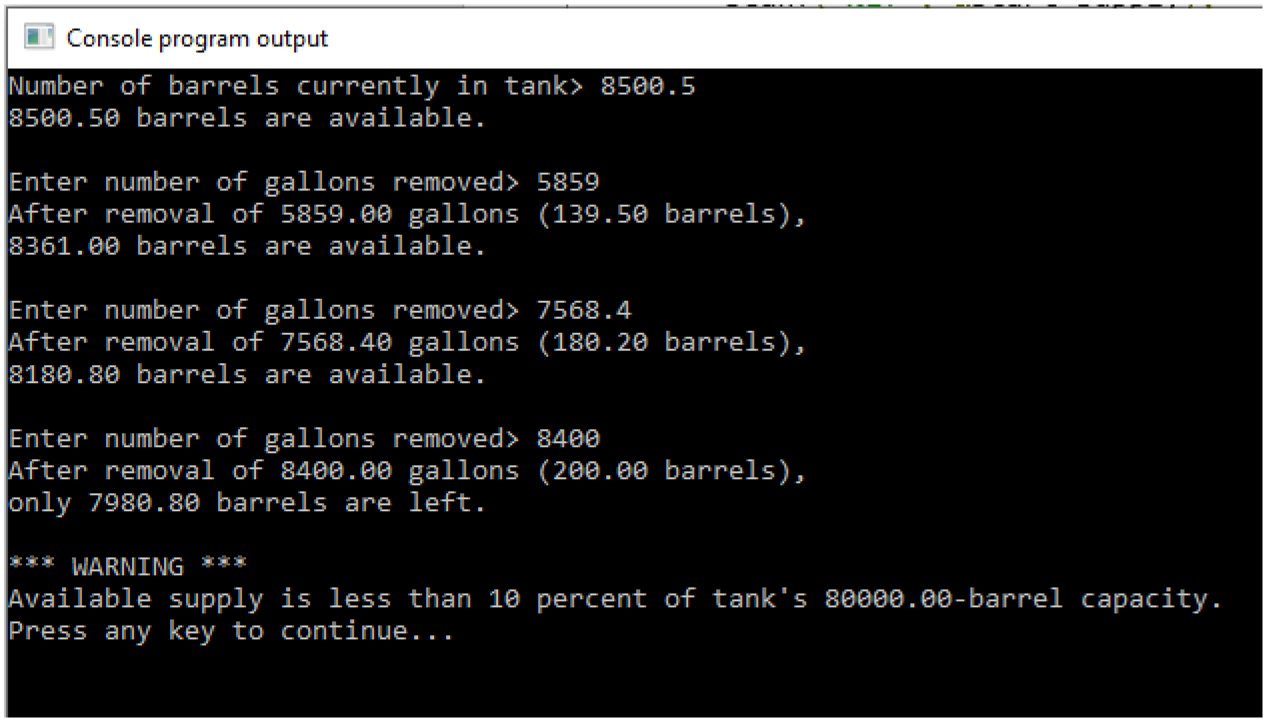
Tracing:

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 a storage tank. Issue a warning when supply falls below MIN_PCT % of tank capacity.

```c
#include <stdio.h>

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

int main(void)
{
    double start_supply, min_supply, current, remov_gals, remov_brls;

    min_supply = MIN_PCT / 100.0 * CAPACITY;

    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);
    }

    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 questions and test your solutions before you check the suggested ones.

1.

Write a program that reads a positive integer and creates a new value with the same digits in reverse order.
Print the reversed integer and the total sum of both integers.
Example: 1234 → reversed: 4321, sum: 5555

2.

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

3.

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

4.

Write a program that reads an arbitrary number of integers (at least one) from the user and prints:
- The maximal value entered
- The minimal value entered
- The number of integers entered

5.

Use the Euclidean algorithm to find the GCD of 1350 and 2275. Show your work.

6.

Write a program that reads a positive integer and prints whether it is a perfect number.
A number n is perfect if the sum of all its proper divisors (divisors other than n) equals n.
Example: 6 is perfect because 1 + 2 + 3 = 6.

7.

Read a positive number as the size of the butterfly and print numbers in a butterfly shape.
You must use the format specifier "%3d" to print each integer, and print a newline at the end of each line.

```
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
```

8.

Read a positive number as the size and draw a box with an X inside.
The box dimensions are (2*size + 1) × (2*size + 1).
When size = 1:   When size = 3:

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

9.

Write a program that adds digits of a number repeatedly until the result is less than 10.
Example: fun(12399987654020) returns 2 because:
 1+2+3+9+9+9+8+7+6+5+4+0+2+0 = 65
 6+5 = 11
 1+1 = 2

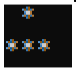
10.

A robot starts at origin (0, 0) and receives movement commands: U (up), D (down), L (left), R (right), E (end).
The robot must stay within boundary (X1, Y1) to (X2, Y2) and avoid N obstacles.
Input: boundary coordinates, number of obstacles, obstacle coordinates, then a command string.
Output: the robot's final position.
Example: boundary (0,0)–(4,4), obstacles (1,2),(3,3),(2,0), commands "URDLLERERUUDE" → output: (0, 0)

11.

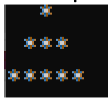
Read a positive integer n and print a triangle of n rows as illustrated in the examples provided below.

For input 2:



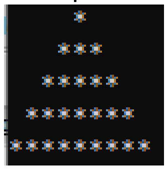
```
 *
***
```

For input 3:



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

For input 5:



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

12.

a) int triangle(int n) — returns the nth triangle number (sum of first n positive integers).
Restriction: must NOT use %, /, or * in your solution.

b) int tetrahedral(int n) — returns the nth tetrahedral number (sum of first n triangle numbers).
Restriction: must NOT use %, /, or * in your solution.

Test cases:

```
assert(triangle(2)==3);
assert(triangle(3)==6);
assert(triangle(5)==15);
assert(tetrahedral(2)==4);
assert(tetrahedral(3)==10);
```

13.

Given an 8-digit serial number where 1=Red, 2=Green, 3=Blue, print the fraction of each colour.
Example input: 11231123 → R = 4/8, G = 2/8, B = 2/8

14.

Write void hollow_square(int w) that prints a hollow square of side w with both diagonals drawn in stars.
Example — hollow_square(5):

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