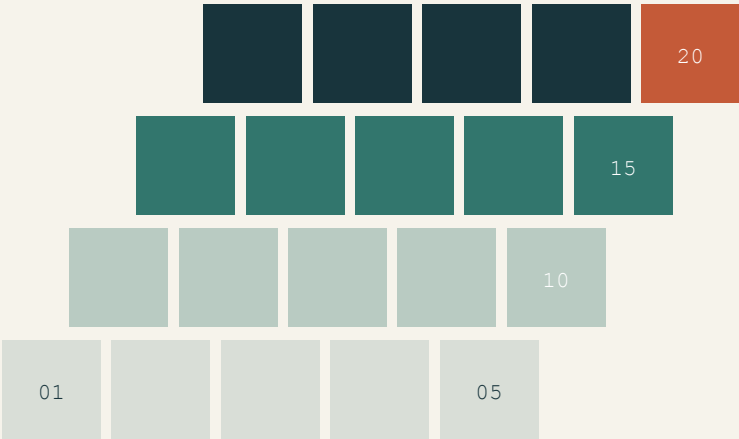


Προγραμματισμός με μικρά βήματα

20 πολύ εύκολες ασκήσεις
Από μια σταθερή απάντηση στο FizzBuzz



ΔΙΑΒΑΣΕ ΚΑΙ ΔΟΚΙΜΑΣΕ

Η σειρά σου

Πριν ξεκινήσεις · Η γλώσσα · Testing · Η διαδρομή

Βήμα	Μάθημα	Αρχή	Λύση
01	Μια σταθερή απάντηση	14	81
02	Ο ίδιος αριθμός	18	82
03	Ο επόμενος αριθμός	21	83
04	Δύο αριθμοί μαζί	24	84
05	Το διπλάσιο	27	85
06	Από λεπτά σε δευτερόλεπτα	30	86
07	Η διαφορά δύο αριθμών	33	87
08	Γύρω από το ορθογώνιο	36	88
09	Τι περισσεύει	39	89
10	Είναι μηδέν;	42	90
11	Είναι θετικός;	45	91
12	Έφτασε το όριο;	48	92
13	Ζυγός αριθμός	51	93
14	Απόσταση από το μηδέν	54	94
15	Ο μεγαλύτερος από τους δύο	57	95
16	Τρεις περιπτώσεις	61	96
17	Μέσα στα όρια	65	97
18	Χωρίς υπόλοιπο	69	98
19	Η πρώτη λέξη του FizzBuzz	72	99
20	Οι κανόνες του FizzBuzz	76	100

Οι τίτλοι και οι αριθμοί σελίδων είναι σύνδεσμοι.

ΑΠΟ ΤΟ ΜΗΔΕΝ

Πριν ξεκινήσεις

Ξεκινάς από το μηδέν και λύνεις είκοσι μικρές ασκήσεις σε C. Σε κάθε βήμα διαβάζεις ένα μάθημα, γράφεις τη δική σου συνάρτηση και τη δοκιμάζεις με το εργαλείο testing της γλώσσας. Οι λύσεις βρίσκονται στο τελευταίο μέρος.

Άνοιξε τον σωστό φάκελο

Άνοιξε τον φάκελο `c` στον editor σου. Στο Terminal γράψε `cd`, σύρε τον ίδιο φάκελο από το Finder και πάτησε Enter. Έλεγξε πού βρίσκεσαι:

```
SH
pwd
ls
```

Θα πρέπει να βλέπεις το `BOOK.md`, το PDF και τον φάκελο `exercises`. Οι εντολές του βιβλίου ξεκινούν από αυτόν τον φάκελο. Όταν μια εντολή χρειάζεται διαφορετική θέση, τα απαιτούμενα `cd` εμφανίζονται ρητά.

Διάβασε και γράψε

Το PDF και το `BOOK.md` περιέχουν την ίδια ύλη σε δύο μορφές. Διάλεξε μία για διάβασμα. Στην πρώτη άσκηση γράφεις στο `exercise.c`. Το `exercise_test.c` περιέχει τις δοκιμές της.

Αποθήκευε με `Command-S` πριν τρέξεις tests. Το `TODO` δείχνει το προσωρινό σώμα που θα αντικαταστήσεις. Κράτησε το όνομα της συνάρτησης και τις παραμέτρους της. Το βιβλίο εξηγεί ξεχωριστά πώς προσθέτεις δική σου δοκιμή.

Τι είναι συνάρτηση και test

Μια συνάρτηση δέχεται τιμές και επιστρέφει αποτέλεσμα. Μια συνάρτηση που τριπλασιάζει το 4 επιστρέφει 12. Η συγκεκριμένη τιμή που δίνεις λέγεται όρισμα. Το όνομα που τη δέχεται στον ορισμό της συνάρτησης λέγεται παράμετρος.

Ένα test καλεί τη συνάρτηση με γνωστή είσοδο και συγκρίνει την απάντηση με εκείνη που περιμένει. Η επιστροφή τιμής και η εμφάνιση στην οθόνη είναι διαφορετικές πράξεις. Εδώ γράφεις συναρτήσεις που επιστρέφουν τιμές· το εργαλείο testing εμφανίζει αν οι δοκιμές πέτυχαν.

Τι σημαίνει μια αποτυχία

Το αρχικό `TODO` προκαλεί επίτηδες αποτυχία. Πρώτα το αντικαθιστάς με δική σου υλοποίηση και μετά εξετάζεις τα αποτελέσματα των tests. Ένα σφάλμα σύνταξης εμφανίζεται πριν μπορέσει να ελεγχθεί η συμπεριφορά της συνάρτησης.

Όταν αποτυγχάνει μια σύγκριση, διάβασε το όνομα του test, την είσοδο, την αναμενόμενη απάντηση και την πραγματική. Διόρθωσε την προσπάθειά σου και ξανατρέξε την ίδια εντολή. Όταν όλα περάσουν, εξήγησε γιατί η λύση σου δουλεύει. Οι δοκιμές καλύπτουν συγκεκριμένες περιπτώσεις και όρια· η εκφώνηση περιγράφει τη συνολική απαίτηση.

Πότε προχωράς

Προχώρα όταν μπορείς να εξηγήσεις τι δέχεται η συνάρτηση, τι επιστρέφει και γιατί περνούν τα παραδείγματα. Αν χρειαστεί να διαβάσεις την έτοιμη λύση, κλείσε την και ξαναγράψε την προσπάθειά σου από τη δική σου εξήγηση.

ΕΓΚΑΤΑΣΤΑΣΗ ΚΑΙ ΣΥΝΤΑΞΗ

Η C με μια ματιά

Στη C γράφεις αρχεία `.c` και τα μεταγλωττίζεις σε εκτελέσιμο πρόγραμμα. Εδώ χρησιμοποιούμε τον compiler Clang που παρέχει η Apple. Οι ασκήσεις έχουν έτοιμο το πρόγραμμα των tests, οπότε συμπληρώνεις μόνο τη συνάρτηση που υπολογίζει την απάντηση.

Εγκατάσταση στο macOS

Άνοιξε Terminal και έλεγξε.

```
SH
cc --version
```

Αν λείπουν τα εργαλεία ανάπτυξης, η επίσημη οδηγία της Apple δίνει την εντολή.

```
SH
xcode-select --install
```

Ολοκλήρωσε την εγκατάσταση στο παράθυρο που ανοίγει και ξανατρέξε `cc --version`. Αν έχεις ήδη Xcode με διαθέσιμα τα εργαλεία του, δεν χρειάζεται δεύτερη εγκατάσταση. Οι εντολές του βιβλίου επιλέγουν C17.

Διάβασε μια μικρή συνάρτηση

```
C
1 int example(void) {
2     return 7;
3 }
```

Το πρώτο `int` δηλώνει ακέραιο αποτέλεσμα. Το `example` είναι το όνομα. Το `(void)` σημαίνει ότι δεν δέχεται παραμέτρους. Τα άγκιστρα περικλείουν το σώμα και το `return 7;` επιστρέφει την τιμή 7. Μια παράμετρος όπως `int n` δηλώνει ακέραιο με όνομα `n`.

Για λογικές τιμές χρησιμοποιούμε `bool`, `true` και `false` από το ενσωματωμένο header `<stdbool.h>`. Το `exercise.h` το περιλαμβάνει ήδη. Το `.h` είναι αρχείο δηλώσεων:

περιγράφει το όνομα, τις παραμέτρους και τον τύπο αποτελέσματος της συνάρτησης, ώστε να την γνωρίζουν τα tests. Το `#include "exercise.h"` εισάγει αυτές τις δηλώσεις.

Στις ασκήσεις με κείμενο η έτοιμη επικεφαλίδα έχει `const char *`. Αυτός είναι δείκτης προς χαρακτήρες που δεν τροποποιούμε μέσω του δείκτη. Εδώ επιστρέφουμε μόνο σταθερά κείμενα μέσα σε διπλά εισαγωγικά. Δεν χρειάζεται να δεσμεύσεις μνήμη ή να φτιάξεις πίνακα χαρακτήρων.

Τα μικρά αριθμητικά όρια των εκφωνήσεων κρατούν τους υπολογισμούς μέσα στο εύρος του `int` στο Mac. Τα αποτελέσματα αριθμών είναι τιμές, όχι κείμενα που εμφανίζουμε με `printf`.

ΤΟ ΕΡΓΑΛΕΙΟ TESTING

Unity

Η C έχει το `assert` της τυπικής βιβλιοθήκης, αλλά δεν περιλαμβάνει πλήρες testing framework. Το Unity προσφέρει ονομασμένα tests, assertions για διαφορετικούς τύπους και συνοπτική αναφορά. Είναι μικρό και γραμμένο σε C. Εναλλακτικές είναι το `cmocka`, που δίνει έμφαση και στα `mocks`, και το `Criterion`, που υποστηρίζει αυτόματη καταχώριση tests και απομόνωση διεργασιών. Η C χρησιμοποιείται σε διαφορετικά περιβάλλοντα, από `embedded` συσκευές μέχρι εφαρμογές υπολογιστή. Η επιλογή framework εξαρτάται και από τον στόχο μεταγλώττισης.

Στο βιβλίο χρησιμοποιούμε **Unity**. Η μεταγλώττιση εμφανίζεται ρητά με `cc`, ώστε να φαίνεται πώς συνδέονται η συνάρτηση, το αρχείο tests και η βιβλιοθήκη.

Πηγές: `Unity`, οδηγός `Unity`, `cmocka`, `Criterion`.

UNITY

Ετοίμασε τη μεταγλώττιση

Από τον φάκελο `c`, έλεγξε τον compiler:

```
SH
cc --version
```

Το `lib/unity` περιέχει τα αρχεία της βιβλιοθήκης Unity και την άδειά της. Το `unity.h` δηλώνει τα εργαλεία testing και το `unity.c` περιέχει την υλοποίησή τους. Το `unity_internals.h` υποστηρίζει εσωτερικά το framework.

UNITY

Διάβασε το πρώτο test

Γράφεις στο `exercises/01-hello/exercise.c`. Το `exercise.h` δηλώνει τη συνάρτηση που πρέπει να υλοποιήσεις. Το `exercise_test.c` περιέχει:

C

```
1  #include "unity.h"
2  #include "exercise.h"
3
4  void setUp(void) {}
5  void tearDown(void) {}
6
7  void test_01_exact_greeting(void) {
8      TEST_ASSERT_EQUAL_STRING("Hello!", solve());
9  }
10
11  int main(void) {
12      UNITY_BEGIN();
13      RUN_TEST(test_01_exact_greeting);
14      return UNITY_END();
15  }
```

Το `#include "unity.h"` φέρνει τα assertions του framework. Το `TEST_ASSERT_EQUAL_STRING` συγκρίνει κείμενα. Για ακεραίους χρησιμοποιούμε `TEST_ASSERT_EQUAL_INT` και για λογικές τιμές `TEST_ASSERT_TRUE` ή `TEST_ASSERT_FALSE`.

Τα `setUp` και `tearDown` καλούνται πριν και μετά από κάθε test. Εδώ μένουν κενά. Μέσα στη `main`, το `UNITY_BEGIN` ξεκινά την καταγραφή, το `RUN_TEST` εκτελεί μια ονομασμένη δοκιμή και το `UNITY_END` εμφανίζει το σύνολο και δίνει κωδικό εξόδου.

UNITY

Μεταγλώττισε και τρέξε

Ξεκίνα από τον φάκελο `c`:

```
SH
cd exercises/01-hello
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το πρώτο `cd` μπαίνει στην άσκηση. Το `cc` μεταγλωττίζει τη συνάρτηση, τα tests και το Unity σε ένα εκτελέσιμο. Το `-I` δείχνει πού βρίσκονται τα headers, το `-o` ονομάζει το εκτελέσιμο και το `-std=c17` επιλέγει τη διάλεκτο C17. Οι επιλογές προειδοποιήσεων βοηθούν να εντοπίσεις λάθη πριν τρέξει ο κώδικας.

Η ανάποδη κάθετος ενώνει την εντολή με την επόμενη γραμμή. Μετά από επιτυχημένη μεταγλώττιση, το `./exercise_tests` τρέχει τις δοκιμές. Το τελευταίο `cd ../../` επιστρέφει στον φάκελο `c`.

Το αρχικό `TODO` σταματά την εκτέλεση. Αφού συμπληρώσεις τη λύση, το Unity αναφέρει πόσα tests πέρασαν και πόσα απέτυχαν. Σε λάθος αποτέλεσμα δείχνει τη γραμμή του test και την αναμενόμενη και πραγματική τιμή.

UNITY

Πρόσθεσε δικό σου test

Στο `exercise_test.c` της άσκησης 02, πριν από τη `main`, πρόσθεσε:

C

```
1 void test_my_number(void) {  
2     TEST_ASSERT_EQUAL_INT(23, solve(23));  
3 }
```

Μέσα στη `main`, πριν από το `return UNITY_END()`, πρόσθεσε:

C

```
1 RUN_TEST(test_my_number);
```

Μεταγλώττισε ξανά και τρέξε το νέο εκτελέσιμο. Η καταχώριση με `RUN_TEST` είναι απαραίτητη για να εκτελεστεί η καινούργια δοκιμή.

ΠΡΩΤΑ ΚΑΤΑΛΑΒΑΙΝΟΥΜΕ, ΜΕΤΑ ΠΡΟΧΩΡΑΜΕ

Η διαδρομή

Ένα μικρό βήμα κάθε φορά. Οι ασκήσεις εμπέδωσης δεν απαιτούν νέα έννοια. Οι είσοδοι είναι μικρές και έγκυρες. Δεν χρησιμοποιούμε ακόμη επαναλήψεις, πίνακες, αναδρομή, αρχεία ή δικτυακές υπηρεσίες.

Βήμα	Η νέα ιδέα
01	Επιστροφή μιας σταθερής τιμής
02	Μία παράμετρος
03	Πρόσθεση με +
04	Δύο παράμετροι
05	Πολλαπλασιασμός με *
06	Δίνουμε ονόματα σε τιμές και αποτελέσματα
07	Αφαίρεση με -
08	Παρενθέσεις σε μια έκφραση
09	Υπόλοιπο διαίρεσης
10	Σύγκριση ισότητας και λογική τιμή
11	Σύγκριση με >
12	Σύγκριση που περιλαμβάνει την ισότητα
13	Εμπέδωση: υπόλοιπο και ισότητα
14	Πρώτη επιλογή με if / else
15	Εμπέδωση: επιλέγουμε ανάμεσα σε δύο παραμέτρους
16	Περισσότερα από δύο ενδεχόμενα
17	Συνδυάζουμε δύο ελέγχους με και
18	Εμπέδωση: ο διαιρέτης γίνεται παράμετρος
19	Εμπέδωση: μια συνθήκη επιλέγει κείμενο
20	Σειρά ελέγχων όταν οι περιπτώσεις συμπίπτουν

Μετά την εικοστή μπορούν να ακολουθήσουν η επανάληψη, το μέτρημα από το 1 έως το n και η πλήρης ακολουθία FizzBuzz.

Οι ασκήσεις

Διάβασε το μάθημα, γράψε και τρέξε τα tests.

Κάθε άσκηση δίνει τα αρχεία και την ακριβή εντολή.

01	02	03	04	05	06	07	08	09	10
									
11	12	13	14	15	16	17	18	19	20
									

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

01 Ο ορισμός και η πρώτη τιμή

Πριν ζητήσουμε από τον υπολογιστή να λύσει κάτι, του δίνουμε έναν κανόνα με όνομα. Το παράδειγμα επιστρέφει πάντα το κείμενο "Ready!". Διάβασε πρώτα πού αρχίζει και πού τελειώνει η συνάρτηση.

C

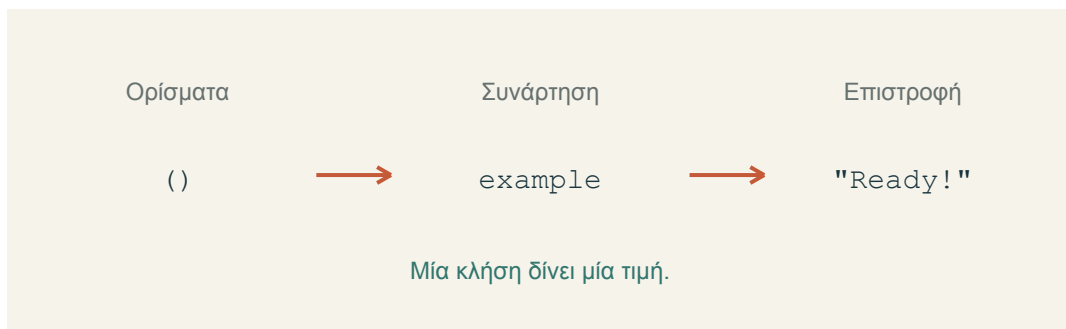
```
1  const char *example(void) {  
2      return "Ready!";  
3  }
```

Διάβασε τις γραμμές

- 01 Το `example` είναι το όνομα της συνάρτησης. Το `(void)` λέει ρητά ότι δεν δέχεται παραμέτρους. Ο τύπος αποτελέσματος γράφεται πριν από το όνομα.
- 02 Το `const char *` δηλώνει δείκτη σε χαρακτήρες που δεν αλλάζουμε μέσω αυτού. Εδώ η συνάρτηση δίνει τη διεύθυνση ενός σταθερού κειμένου. Κράτησε τον έτοιμο τύπο χωρίς να χρειάζεται ακόμη να χειριστείς δείκτες.
- 03 Τα `{` και `}` ανοίγουν και κλείνουν το σώμα. Η εσοχή δείχνει ποια γραμμή ανήκει στη συνάρτηση.
- 04 Το `return "Ready!";` δίνει πίσω την τιμή. Το `;` είναι απαραίτητο στο τέλος της εντολής. Τα διπλά εισαγωγικά ορίζουν το κείμενο.
- 05 Η `example()` είναι κλήση χωρίς ορίσματα. Στην κλήση δεν γράφουμε `void`. Το αποτέλεσμα επιστρέφεται στον κώδικα που την κάλεσε.
- 06 Στο άλτο αρχείο θα δεις και `#include "exercise.h"`. Διαβάζει την κοινή δήλωση της συνάρτησης από το έτοιμο αρχείο `exercise.h`. Το κρατάς.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

01 Από τον κώδικα στην τιμή



Ακολουθήσε μια κλήση

Η `example()` επιστρέφει `"Ready!"`. Η κλήση δεν δίνει είσοδο. Το σώμα δίνει τη σταθερή τιμή.

Στάσου λίγο

Τα διπλά εισαγωγικά είναι μέρος του αποτελέσματος; Αρκεί να γράψεις τον ορισμό για να εμφανιστεί το μήνυμα;

Έλεγε τη σκέψη σου. Όχι και στις δύο ερωτήσεις. Τα εισαγωγικά ορίζουν το κείμενο στον κώδικα. Χρειάζεται κλήση για να πάρουμε την τιμή και χωριστή εντολή εμφάνισης για να τη δούμε. Στις ασκήσεις, το `test` καλεί τη συνάρτηση της άσκησης και ελέγχει την τιμή της.

Το αρχείο της προσπάθειάς σου

Στην άσκηση κράτησε τις έτοιμες γραμμές `#include` και την επικεφαλίδα της `solve`. Αντικατάστησε το προσωρινό σώμα με το `TODO`, το `fputs` και το `exit` με τη δική σου επιστροφή.

Συνέχισε στην εκφώνηση της άσκησης 01, στη σελίδα 16.

Η ΑΣΚΗΣΗ

01 Μια σταθερή απάντηση

Τι θα φτιάξεις

Συμπλήρωσε τη συνάρτηση ώστε να επιστρέφει ακριβώς το κείμενο `"Hello!"`. Δεν δέχεται καμία παράμετρο.

Όρια: Δεν υπάρχει είσοδος. Η απάντηση έχει κεφαλαίο H και ένα θαυμαστικό, χωρίς κενά ή αλλαγή γραμμής.



```
solve() → "Hello!"
```

Μία κλήση. Η ίδια απάντηση κάθε φορά.

Η ιδέα

Μια συνάρτηση μπορεί να δίνει πάντα την ίδια απάντηση. Το κείμενο γράφεται μέσα σε διπλά εισαγωγικά. Τα εισαγωγικά δείχνουν πού αρχίζει και πού τελειώνει το κείμενο. Δεν αποτελούν μέρος του αποτελέσματος.

ΜΙΑ ΣΤΑΘΕΡΗ ΑΠΑΝΤΗΣΗ

01 Γράψε και έλεγξε

Η συνάρτηση καλείται χωρίς ορίσματα.

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve()</code>	<code>"Hello!"</code>

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο `c`.

Προσπάθεια: `exercises/01-hello/exercise.c`.

Tests: `exercises/01-hello/exercise_test.c`.

```
SH
cd exercises/01-hello
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο `testing`.

Μικρή βοήθεια

Το αρχείο έχει ήδη το περίβλημα της συνάρτησης. Χρειάζεσαι μόνο τη γραμμή που δίνει την απάντηση.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Η λύση βρίσκεται στη σελίδα **81**.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

02 Μια τιμή αποκτά όνομα

Η ίδια συνάρτηση μπορεί να δεχτεί διαφορετικούς αριθμούς σε διαφορετικές κλήσεις. Θα ονομάσουμε τον αριθμό `value` και θα τον επιστρέψουμε όπως τον παραλάβαμε.

C

```
1 int example(int value) {  
2     return value;  
3 }
```

Διάβασε τις γραμμές

- 01 Το `int value` μέσα στις παρενθέσεις δηλώνει ακέραια παράμετρο με όνομα `value`. Αντικαθιστά το `void` που σήμαινε καμία παράμετρο. Το `int` πριν από το όνομα της συνάρτησης δηλώνει ακέραιο αποτέλεσμα.
- 02 Στον ορισμό, το `value` είναι παράμετρος. Στην κλήση με `6`, το `6` είναι όρισμα. Δεν γράφουμε ούτε τύπο ούτε όνομα παραμέτρου στην κλήση.
- 03 Το `value` στο σώμα διαβάζει την τιμή της παραμέτρου. Το `"value"` με εισαγωγικά θα ήταν κείμενο, όχι η τιμή της.

Ακολούθησε μια κλήση

Η `example(6)` επιστρέφει `6`. Το `value` παίρνει την τιμή `6`.

Η `example(-3)` επιστρέφει `-3`. Σε νέα κλήση, το `value` παίρνει την τιμή `-3`.

Στάσου λίγο

Τι θα επιστρέψει η `example(0)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

Έλεγξε τη σκέψη σου. Η τιμή είναι `0`. Επιστρέφουμε την τιμή της συγκεκριμένης κλήσης, όχι τον αριθμό που χρησιμοποιήσαμε στην προηγούμενη.

Συνέχισε στην εκφώνηση της άσκησης 02, στη σελίδα 19.

Η ΑΣΚΗΣΗ

02 Ο ίδιος αριθμός

Τι θα φτιάξεις

Δέξου έναν ακέραιο n και επέστρεψε τον ίδιο αριθμό.

Όρια: $-1000 \leq n \leq 1000$.



Η ιδέα

Η παράμετρος n είναι το όνομα της τιμής που δίνεται στη συνάρτηση. Σε κάθε κλήση μπορεί να έχει άλλη τιμή. Το όνομα γράφεται χωρίς εισαγωγικά, γιατί θέλουμε την τιμή του.

Ο ΙΔΙΟΣ ΑΡΙΘΜΟΣ

02 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά n .

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(8)</code>	8
<code>solve(-4)</code>	-4
<code>solve(0)</code>	0
<code>solve(-1000)</code>	-1000
<code>solve(1000)</code>	1000

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **c**.

Προσπάθεια: `exercises/02-echo/exercise.c`.

Tests: `exercises/02-echo/exercise_test.c`.

SH

```
cd exercises/02-echo
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο **testing**.

Μικρή βοήθεια

Αν σου δώσουν 8, επιστρέφεις 8. Αν σου δώσουν -4, επιστρέφεις -4. Ποιο όνομα κρατά αυτή την τιμή;

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Η λύση βρίσκεται στη σελίδα 82.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

03 Η πρόσθεση γράφεται με +

Μια έκφραση είναι ένα κομμάτι κώδικα που δίνει μια τιμή. Το `value + 3` είναι έκφραση. Υπολογίζουμε το άθροισμα πριν το επιστρέψουμε.

C

```
1 int example(int value) {  
2     return value + 3;  
3 }
```

Διάβασε τις γραμμές

- 01 Το + ανάμεσα σε δύο ακεραίους υπολογίζει το άθροισμα. Πρώτα διαβάζεται η τιμή του `value` και μετά προστίθεται το 3.
- 02 Το αποτέλεσμα της έκφρασης επιστρέφεται. Δεν αλλάζουμε την παράμετρο `value` και δεν χρειάζεται νέο τοπικό όνομα.
- 03 Το 3 χωρίς εισαγωγικά είναι αριθμός. Στο "3" τα εισαγωγικά θα το έκαναν κείμενο, ακατάλληλο για αυτή την ακέρεια πρόσθεση.

Ακολούθησε μια κλήση

Η `example(4)` επιστρέφει 7. Το `value + 3` γίνεται `4 + 3` και μετά 7.

Η `example(8)` επιστρέφει 11. Το `value + 3` γίνεται `8 + 3` και μετά 11.

Στάσου λίγο

Τι θα επιστρέψει η `example(-3)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

Έλεγξε τη σκέψη σου. Η τιμή είναι 0, επειδή το `-3 + 3` κάνει μηδέν.

Συνέχισε στην εκφώνηση της άσκησης 03, στη σελίδα 22.

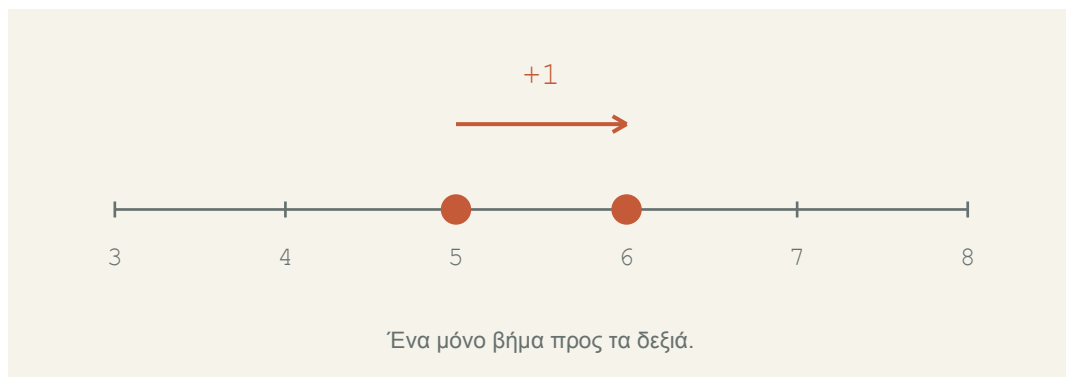
Η ΑΣΚΗΣΗ

03 Ο επόμενος αριθμός

Τι θα φτιάξεις

Δέξου έναν ακέραιο n και επέστρεψε τον αριθμό που είναι κατά 1 μεγαλύτερος.

Όρια: $-1000 \leq n \leq 1000$.



Η ιδέα

Μια έκφραση είναι ένα μικρό κομμάτι κώδικα που δίνει μια τιμή. Το σύμβολο $+$ προσθέτει δύο αριθμούς. Μπορείς να επιστρέψεις κατευθείαν την τιμή μιας έκφρασης.

Ο ΕΠΟΜΕΝΟΣ ΑΡΙΘΜΟΣ

03 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά n .

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(5)</code>	6
<code>solve(0)</code>	1
<code>solve(-1)</code>	0
<code>solve(-1000)</code>	-999
<code>solve(1000)</code>	1001

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **c**.

Προσπάθεια: `exercises/03-next-number/exercise.c`.

Tests: `exercises/03-next-number/exercise_test.c`.

SH

```
cd exercises/03-next-number
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο **testing**.

Μικρή βοήθεια

Ξεκίνα από την τιμή που σου δόθηκε και πρόσθεσε μία μονάδα.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Η λύση βρίσκεται στη σελίδα **83**.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

04 Δύο παράμετροι, δύο θέσεις

Μετράμε μήλα και αχλάδια. Χρειαζόμαστε δύο τιμές στην ίδια κλήση. Κάθε τιμή παίρνει τη δική της θέση μέσα στις παρενθέσεις.

C

```
1  int example(int apples, int pears) {  
2      return apples + pears;  
3  }
```

Διάβασε τις γραμμές

- 01 Στο `int apples, int pears`, το κόμμα χωρίζει τις δύο παραμέτρους. Κάθε παράμετρος έχει δικό της όνομα.
- 02 Στην κλήση, το πρώτο όρισμα αντιστοιχεί στο `apples` και το δεύτερο στο `pears`. Το κόμμα χωρίζει και τα ορίσματα.
- 03 Το `apples + pears` διαβάζει και τις δύο τιμές και επιστρέφει το άθροισμα. Τα ονόματα βοηθούν να καταλάβουμε τι μετράμε.

Ακολούθησε μια κλήση

Η `example(2, 4)` επιστρέφει 6. Το `apples` γίνεται 2 και το `pears` γίνεται 4.

Η `example(3, 5)` επιστρέφει 8. Το `3 + 5` δίνει 8.

Στάσου λίγο

Στην κλήση με ορίσματα 0, 7, ποια τιμή παίρνει κάθε παράμετρος και τι επιστρέφεται;

Έλεγε τη σκέψη σου. Το `apples` παίρνει 0 και το `pears` παίρνει 7. Το άθροισμα είναι 7.

Συνέχισε στην εκφώνηση της άσκησης 04, στη σελίδα 25.

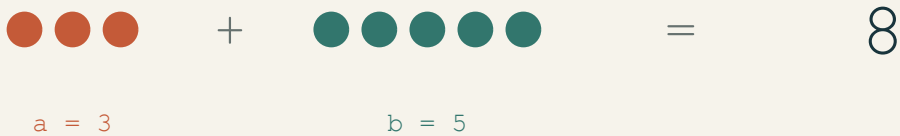
Η ΑΣΚΗΣΗ

04 Δύο αριθμοί μαζί

Τι θα φτιάξεις

Δέξου δύο ακέραιους a και b και επέστρεψε το άθροισμά τους.

Όρια: $-1000 \leq a, b \leq 1000$.



The diagram illustrates the addition of two quantities. On the left, there are three red dots, followed by a plus sign, then five green dots, followed by an equals sign, and finally the number 8. Below the red dots is the text $a = 3$ and below the green dots is the text $b = 5$.

Δύο ποσότητες γίνονται ένα άθροισμα.

Η ιδέα

Μια συνάρτηση μπορεί να δέχεται περισσότερες από μία τιμές. Τα ονόματα χωρίζονται με κόμμα. Στην κλήση `solve(3, 5)`, το 3 αντιστοιχεί στο a και το 5 στο b .

ΔΥΟ ΑΡΙΘΜΟΙ ΜΑΖΙ

04 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά a, b .

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(3, 5)</code>	8
<code>solve(0, 0)</code>	0
<code>solve(-3, 5)</code>	2
<code>solve(-4, -6)</code>	-10
<code>solve(1000, 1000)</code>	2000

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **c**.

Προσπάθεια: `exercises/04-add/exercise.c`.

Tests: `exercises/04-add/exercise_test.c`.

SH

```
cd exercises/04-add
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο **testing**.

Μικρή βοήθεια

Η πρόσθεση είναι η ίδια που χρησιμοποίησες πριν. Τώρα και οι δύο αριθμοί δίνονται από την κλήση.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Η λύση βρίσκεται στη σελίδα **84**.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

05 Ο πολλαπλασιασμός γράφεται με *

Για να πάρουμε τρεις ίδιες ποσότητες, πολλαπλασιάζουμε την αρχική τιμή με το 3. Στον κώδικα ο πολλαπλασιασμός γράφεται με αστερίσκο.

```
c
1  int example(int value) {
2      return value * 3;
3  }
```

Διάβασε τις γραμμές

- 01 Το * είναι ο τελεστής του πολλαπλασιασμού. Χρειάζεται και ανάμεσα σε μεταβλητή και αριθμό. Δεν γράφουμε 3value.
- 02 Στο value * 3 το value διαβάζεται από την κλήση, ενώ το 3 είναι σταθερό. Το αποτέλεσμα παραμένει ακέραιος.
- 03 Τα κενά γύρω από το * βοηθούν την ανάγνωση. Το μαθηματικό σύμβολο × και το γράμμα x δεν αντικαθιστούν τον αστερίσκο στον κώδικα.

Ακολούθησε μια κλήση

Η example(4) επιστρέφει 12. Το 4 * 3 είναι 4 + 4 + 4.

Η example(0) επιστρέφει 0. Το 0 * 3 παραμένει 0.

Στάσου λίγο

Τι θα επιστρέψει η example(-2); Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

Έλεγξε τη σκέψη σου. Η τιμή είναι -6. Ο αστερίσκος λειτουργεί και με αρνητικό ακέραιο.

Συνέχισε στην εκφώνηση της άσκησης 05, στη σελίδα 28.

Η ΑΣΚΗΣΗ

05 Το διπλάσιο

Τι θα φτιάξεις

Δέξου έναν ακέραιο n και επέστρεψε το διπλάσιό του.

Όρια: $-1000 \leq n \leq 1000$.



8

Τέσσερα, δύο φορές.

Η ιδέα

Ο πολλαπλασιασμός γράφεται με $*$. Η ιδέα είναι η ίδια με το να πάρεις μια ποσότητα δύο φορές. Δεν χρησιμοποιούμε το γράμμα x για πολλαπλασιασμό.

ΤΟ ΔΙΠΛΑΣΙΟ

05 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά n .

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(4)</code>	8
<code>solve(1)</code>	2
<code>solve(0)</code>	0
<code>solve(-3)</code>	-6
<code>solve(1000)</code>	2000

Αποθήκευσε την προσπάθειά. Ξεκίνα από τον φάκελο **c**.

Προσπάθεια: `exercises/05-double/exercise.c`.

Tests: `exercises/05-double/exercise_test.c`.

SH

```
cd exercises/05-double
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο **testing**.

Μικρή βοήθεια

Μπορείς πρώτα να το σκεφτείς ως πρόσθεση του αριθμού με τον εαυτό του.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Η λύση βρίσκεται στη σελίδα **85**.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

06 Κρατάμε ένα ενδιάμεσο αποτέλεσμα

Ένα κουτί έχει δέκα αντικείμενα. Δίνουμε όνομα σε αυτή την ποσότητα και μετά όνομα στο συνολικό πλήθος. Έτσι διαβάζουμε τον υπολογισμό σε μικρά βήματα.

```
c
1  int example(int boxes) {
2      int items_per_box = 10;
3      int total = boxes * items_per_box;
4      return total;
5  }
```

Διάβασε τις γραμμές

- 01 Το `int items_per_box = 10;` δηλώνει τοπική ακέραια μεταβλητή. Το `int` είναι ο τύπος, το `items_per_box` το όνομα και το `=` της δίνει την αρχική τιμή.
- 02 Το `int total = boxes * items_per_box;` αποθηκεύει το γινόμενο. Το `return total;` επιστρέφει αυτή την τιμή. Η κάτω παύλα επιτρέπεται μέσα σε ονόματα.
- 03 Και οι δύο πλευρές του `*` είναι τώρα ονόματα μεταβλητών. Χρησιμοποιούνται οι τιμές τους. Τα τοπικά ονόματα ανήκουν στη συνάρτηση και ορίζονται ξανά σε κάθε κλήση.

Ακολούθησε μια κλήση

Η `example(4)` επιστρέφει 40. Τέσσερα κουτιά επί δέκα αντικείμενα δίνουν 40.

Η `example(0)` επιστρέφει 0. Μηδέν κουτιά δίνουν 0.

Στάσου λίγο

Τι θα επιστρέψει η `example(3)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

Έλεγξε τη σκέψη σου. Η τιμή είναι 30. Το όνομα της ενδιάμεσης τιμής δεν αλλάζει την πράξη που υπολογίζουμε.

Συνέχισε στην εκφώνηση της άσκησης 06, στη σελίδα 31.

Η ΑΣΚΗΣΗ

06 Από λεπτά σε δευτερόλεπτα

Τι θα φτιάξεις

Δέξου έναν ακέραιο αριθμό λεπτών `minutes` και επέστρεψε πόσα δευτερόλεπτα είναι. Κάθε λεπτό έχει 60 δευτερόλεπτα. Δώσε πρώτα ένα όνομα στην τιμή 60. Πολλαπλασίασε το `minutes` με αυτό το όνομα, αποθήκευσε το αποτέλεσμα στο `seconds` και μετά επέστρεψέ το.

Όρια: $0 \leq \text{minutes} \leq 1000$. Δουλεύουμε μόνο με ολόκληρα λεπτά.



3 λεπτά, 60 δευτερόλεπτα στο καθένα.

Η ιδέα

Μπορούμε να δώσουμε ένα όνομα σε μια τιμή και να το χρησιμοποιήσουμε στην επόμενη γραμμή. Ο πολλαπλασιασμός λειτουργεί και όταν γράφουμε ονόματα και στις δύο πλευρές του `*`. Ο υπολογιστής χρησιμοποιεί τις τιμές τους. Το `seconds` λέει ποια ποσότητα έχουμε βρει.

ΑΠΟ ΛΕΠΤΑ ΣΕ ΔΕΥΤΕΡΟΛΕΠΤΑ

06 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `minutes`.

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(3)</code>	180
<code>solve(1)</code>	60
<code>solve(0)</code>	0
<code>solve(60)</code>	3600
<code>solve(1000)</code>	60000

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **c**.

Προσπάθεια: `exercises/06-minutes-to-seconds/exercise.c`.

Tests: `exercises/06-minutes-to-seconds/exercise_test.c`.

SH

```
cd exercises/06-minutes-to-seconds
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο **testing**.

Μικρή βοήθεια

Για 3 λεπτά έχεις τρεις ομάδες των 60 δευτερολέπτων.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος αποτέλεσμα

Η λύση βρίσκεται στη σελίδα 86.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

07 Η αφαίρεση και η σειρά των ορισμάτων

Αφαιρούμε ένα κουπόνι από μια τιμή. Το $-$ ανάμεσα σε δύο τιμές είναι αφαίρεση. Εδώ φαίνεται γιατί έχει σημασία η σειρά των ορισμάτων.

```
c
1  int example(int price, int coupon) {
2      return price - coupon;
3  }
```

Διάβασε τις γραμμές

- 01 Το `price - coupon` αφαιρεί τη δεξιά τιμή από την αριστερή. Το πρώτο όρισμα πηγαίνει στο `price` και το δεύτερο στο `coupon`.
- 02 Αν ανταλλάξουμε τα ορίσματα, συνήθως αλλάζει το αποτέλεσμα. Αυτό δεν φαινόταν στην πρόσθεση, αλλά εδώ είναι ουσιαστικό.
- 03 Στο $5 - 20$ το $-$ κάνει αφαίρεση δύο τιμών. Στο αποτέλεσμα -15 δηλώνει ότι ο αριθμός είναι αρνητικός.

Ακολούθησε μια κλήση

Η `example(20, 5)` επιστρέφει 15. Το `price - coupon` γίνεται $20 - 5$.

Η `example(5, 20)` επιστρέφει -15 . Με αντίστροφα ορίσματα γίνεται $5 - 20$.

Στάσου λίγο

Τι θα επιστρέψει η `example(12, 12)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

Έλεγξε τη σκέψη σου. Η τιμή είναι 0. Αφαιρούμε ίσες ποσότητες.

Συνέχισε στην εκφώνηση της άσκησης 07, στη σελίδα 34.

Η ΑΣΚΗΣΗ

07 Η διαφορά δύο αριθμών

Τι θα φτιάξεις

Δέξου δύο ακέραιους a και b και επέστρεψε τη διαφορά τους, αφαιρώντας τον δεύτερο από τον πρώτο.

Όρια: $-1000 \leq a, b \leq 1000$.



$$8 - 3 = 5$$

Αφαιρούμε τρία. Απομένουν πέντε.

Η ιδέα

Το $-$ ανάμεσα σε δύο αριθμούς κάνει αφαίρεση. Η σειρά έχει σημασία. Το 8 μείον 3 δίνει 5, ενώ το 3 μείον 8 δίνει -5. Η συνάρτηση δέχεται δύο παραμέτρους όπως η άσκηση 4. Αλλάζει μόνο η πράξη.

Η ΔΙΑΦΟΡΑ ΔΥΟ ΑΡΙΘΜΩΝ

07 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά *a*, *b*.

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(8, 3)</code>	5
<code>solve(3, 8)</code>	-5
<code>solve(0, 0)</code>	0
<code>solve(-4, -6)</code>	2
<code>solve(1000, -1000)</code>	2000

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **c**.

Προσπάθεια: `exercises/07-difference/exercise.c`.

Tests: `exercises/07-difference/exercise_test.c`.

SH

```
cd exercises/07-difference
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο **testing**.

Μικρή βοήθεια

Γράψε πρώτα τον αριθμό από τον οποίο αφαιρείς και μετά αυτόν που αφαιρείται.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Η λύση βρίσκεται στη σελίδα **87**.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

08 Οι παρενθέσεις αλλάζουν τη σειρά

Θέλουμε τρεις φορές το άθροισμα δύο αριθμών. Οι παρενθέσεις μας επιτρέπουν να πούμε ποια πράξη πρέπει να γίνει πρώτη.

```
C
1  int example(int first, int second) {
2      return (first + second) * 3;
3  }
```

Διάβασε τις γραμμές

- 01 Οι εσωτερικές παρενθέσεις ομαδοποιούν το άθροισμα. Πρώτα υπολογίζεται το `first + second` και μετά το γινόμενο.
- 02 Χωρίς αυτές, το `first + second * 3` θα έκανε πρώτα τον πολλαπλασιασμό. Με 2 και 4 θα έδινε 14 αντί για 18.
- 03 Οι παρενθέσεις εδώ ομαδοποιούν πράξεις. Οι παρενθέσεις δίπλα στο όνομα της συνάρτησης έχουν άλλο ρόλο, τις παραμέτρους ή τα ορίσματα.

Ακολούθησε μια κλήση

Η `example(2, 4)` επιστρέφει 18. Πρώτα $2 + 4 = 6$. Μετά $6 * 3 = 18$.

Η `example(1, 5)` επιστρέφει 18. Πρώτα $1 + 5 = 6$. Μετά $6 * 3 = 18$.

Στάσου λίγο

Τι θα επιστρέψει η `example(0, 3)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

Έλεγξε τη σκέψη σου. Η τιμή είναι 9. Πρώτα βρίσκουμε το $0 + 3$.

Συνέχισε στην εκφώνηση της άσκησης 08, στη σελίδα 37.

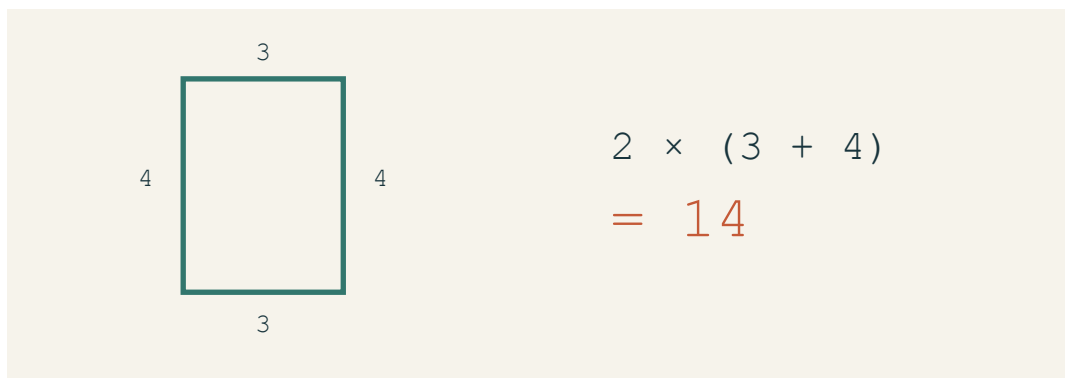
Η ΑΣΚΗΣΗ

08 Γύρω από το ορθογώνιο

Τι θα φτιάξεις

Δέξου το πλάτος `width` και το ύψος `height` και επέστρεψε την περίμετρο: δύο φορές το άθροισμα πλάτους και ύψους.

Όρια: $0 \leq \text{width}, \text{height} \leq 1000$. Εφαρμόζουμε τον ίδιο τύπο και για μηδενικές διαστάσεις.



Η ιδέα

Οι παρενθέσεις ομαδοποιούν μια πράξη. Το $2 * (3 + 4)$ υπολογίζει πρώτα το άθροισμα. Χωρίς τις παρενθέσεις, ο πολλαπλασιασμός εκτελείται πριν από την πρόσθεση.

ΓΥΡΩ ΑΠΟ ΤΟ ΟΡΘΟΓΩΝΙΟ

08 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `width, height`.

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(3, 4)</code>	14
<code>solve(1, 1)</code>	4
<code>solve(0, 5)</code>	10
<code>solve(5, 0)</code>	10
<code>solve(0, 0)</code>	0

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **c**.

Προσπάθεια: `exercises/08-rectangle-perimeter/exercise.c`.

Tests: `exercises/08-rectangle-perimeter/exercise_test.c`.

SH

```
cd exercises/08-rectangle-perimeter
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο **testing**.

Μικρή βοήθεια

Βάλε μέσα σε παρενθέσεις την ποσότητα που θέλεις να πάρεις δύο φορές.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Η λύση βρίσκεται στη σελίδα 88.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

09 Τι μένει από μια μοιρασιά

Μοιράζουμε έντεκα μάρκες σε τέσσερα κουτιά. Κάθε κουτί παίρνει δύο και μένουν τρεις. Το $11 = 4 * 2 + 3$ δείχνει το πηλίκο 2 και το υπόλοιπο 3. Εδώ ζητάμε μόνο το υπόλοιπο.

C

```
1 int example(int tokens, int boxes) {  
2     return tokens % boxes;  
3 }
```

Διάβασε τις γραμμές

- 01 Το `tokens % boxes` δίνει το υπόλοιπο. Το `%` εδώ δεν είναι ποσοστό. Στο παράδειγμα επιστρέφει 3, όχι το πηλίκο 2.
- 02 Με τιμές `int`, το $11 / 4$ δίνει πηλίκο 2. Το $11 \% 4$ δίνει το υπόλοιπο 3.
- 03 Στα παραδείγματα οι μάρκες είναι μη αρνητικές και τα κουτιά περισσότερα από μηδέν. Δεν διαιρούμε ποτέ με το μηδέν.

Ακολούθησε μια κλήση

Η `example(11, 4)` επιστρέφει 3. Τέσσερα κουτιά επί δύο μάρκες χρησιμοποιούν οκτώ. Μένουν 3.

Η `example(12, 4)` επιστρέφει 0. Τέσσερα κουτιά επί τρεις μάρκες χρησιμοποιούν δώδεκα. Μένουν 0.

Στάσου λίγο

Τι θα επιστρέψει η `example(14, 4)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

Έλεγξε τη σκέψη σου. Μένουν 2, επειδή $14 = 4 * 3 + 2$.

Συνέχισε στην εκφώνηση της άσκησης 09, στη σελίδα 40.

Η ΑΣΚΗΣΗ

09 Τι περισσεύει

Τι θα φτιάξεις

Έχεις `candies` καραμέλες και `children` παιδιά. Δίνεις σε κάθε παιδί τον ίδιο ακέραιο αριθμό καραμελών, όσο περισσότερες γίνεται. Επέστρεψε πόσες περισσεύουν.

Όρια: $0 \leq \text{candies} \leq 1000$ και $1 \leq \text{children} \leq 1000$. Υπάρχει πάντα τουλάχιστον ένα παιδί.



Η ιδέα

Ας ξεκινήσουμε από τη μοιρασιά. Με 8 καραμέλες και 4 παιδιά, η διαίρεση $8 \div 4$ δίνει 2 καραμέλες σε κάθε παιδί. Με 3 παιδιά δίνεις πάλι 2 στο καθένα, άρα μοιράζεις $3 \times 2 = 6$ και μένουν $8 - 6 = 2$. Αυτό που μένει ονομάζεται υπόλοιπο. Το υπόλοιπο διαίρεσης δίνει ακριβώς αυτό που περισσεύει. Στις περισσότερες γλώσσες γράφεται με `%`. Στην Elixir γράφεται `rem(a, b)`.

ΤΙ ΠΕΡΙΣΣΕΥΕΙ

09 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `candies`, `children`.

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(8, 3)</code>	2
<code>solve(12, 4)</code>	0
<code>solve(2, 5)</code>	2
<code>solve(0, 3)</code>	0
<code>solve(7, 1)</code>	0

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο `c`.

Προσπάθεια: `exercises/09-leftover/exercise.c`.

Tests: `exercises/09-leftover/exercise_test.c`.

SH

```
cd exercises/09-leftover
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο `testing`.

Μικρή βοήθεια

Δεν ζητάμε πόσες παίρνει το κάθε παιδί. Ζητάμε ό,τι απομένει μετά τη μοιρασιά.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Η λύση βρίσκεται στη σελίδα 89.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

10 Μια σύγκριση δίνει λογική τιμή

Μέχρι εδώ επιστρέφαμε αριθμό ή κείμενο. Τώρα ρωτάμε αν ένας αριθμός είναι ίσος με το 5. Η απάντηση είναι μια λογική τιμή, αληθές ή ψευδές.

```
C
1  #include <stdbool.h>
2
3  bool example(int value) {
4      return value == 5;
5  }
```

Διάβασε τις γραμμές

- 01 Το `==` συγκρίνει την αριστερή τιμή με τη δεξιά. Δεν αλλάζει το `value`. Το μονό `=` δεν είναι ο έλεγχος ισότητας.
- 02 Το `#include <stdbool.h>` διαθέτει τον τύπο `bool` και τα ονόματα `true` και `false` στη C17. Η σύγκριση δίνει 0 ή 1 και μετατρέπεται σε `bool` στην επιστροφή. Στην άσκηση η εισαγωγή βρίσκεται ήδη μέσα στο `exercise.h`.
- 03 Δεν χρειάζεται να γράψουμε ξεχωριστή απόφαση. Η ίδια η σύγκριση παράγει την απάντηση που θα επιστρέψουμε.

Ακολούθησε μια κλήση

Η `example(5)` επιστρέφει `true`. Το `value` είναι ίσο με 5.

Η `example(6)` επιστρέφει `false`. Το 6 δεν είναι ίσο με 5.

Στάσου λίγο

Τι θα επιστρέψει η `example(0)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

Έλεγξε τη σκέψη σου. Η σύγκριση είναι ψευδής. Το 0 δεν ισούται με το 5.

Συνέχισε στην εκφώνηση της άσκησης 10, στη σελίδα 43.

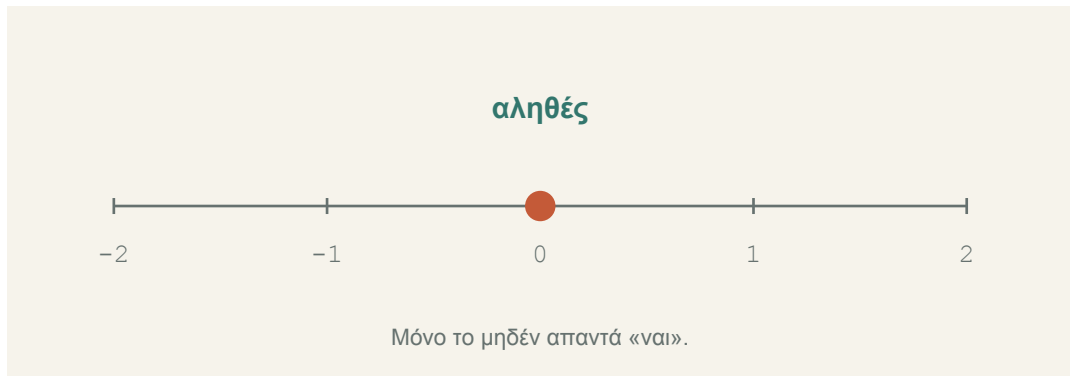
Η ΑΣΚΗΣΗ

10 Είναι μηδέν;

Τι θα φτιάξεις

Δέξου έναν ακέραιο `n`. Επέστρεψε αληθές αν είναι μηδέν και ψευδές σε κάθε άλλη περίπτωση.

Όρια: $-1000 \leq n \leq 1000$.



Η ιδέα

Μια σύγκριση απαντά σε μια ερώτηση. Το `n == 0` ελέγχει αν δύο τιμές είναι ίσες. Στην TypeScript χρησιμοποιούμε `===`. Το αποτέλεσμα είναι λογική τιμή, δηλαδή αληθές ή ψευδές.

ΕΙΝΑΙ ΜΗΔΕΝ;

10 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`. Η τιμή `true` σημαίνει αληθές και η `false` ψευδές.

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(0)</code>	<code>true</code>
<code>solve(1)</code>	<code>false</code>
<code>solve(-1)</code>	<code>false</code>
<code>solve(1000)</code>	<code>false</code>
<code>solve(-1000)</code>	<code>false</code>

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο `c`.

Προσπάθεια: `exercises/10-is-zero/exercise.c`.

Tests: `exercises/10-is-zero/exercise_test.c`.

SH

```
cd exercises/10-is-zero
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο [κεφάλαιο testing](#).

Μικρή βοήθεια

Η ίδια η σύγκριση παράγει την απάντηση. Μπορείς να την επιστρέψεις απευθείας.

Η λύση βρίσκεται στη σελίδα [90](#).

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

11 Μεγαλύτερο σημαίνει αυστηρά μεγαλύτερο

Το σύμβολο `>` ρωτά αν η αριστερή τιμή είναι μεγαλύτερη από τη δεξιά. Η ισότητα δεν αρκεί.

```
C
1  #include <stdbool.h>
2
3  bool example(int value) {
4      return value > 10;
5  }
```

Διάβασε τις γραμμές

- 01 Το `value > 10` είναι μια σύγκριση, όπως η ισότητα. Δίνει λογική τιμή, όχι τη διαφορά των δύο αριθμών.
- 02 Όταν οι δύο τιμές είναι ίσες, το `>` δίνει ψευδές. Δοκίμαζε πάντα και τον ίδιο τον αριθμό του ορίου.
- 03 Το αντίστροφο σύμβολο `<` σημαίνει «μικρότερο». Για παράδειγμα, το `9 < 10` είναι αληθές. Η ανοιχτή πλευρά του συμβόλου κοιτά προς τη μεγαλύτερη τιμή.

Ακολουθήσε μια κλήση

Η `example(11)` επιστρέφει `true`. Το `11 > 10` είναι αληθές.

Η `example(10)` επιστρέφει `false`. Το `10 > 10` είναι ψευδές.

Στάσου λίγο

Τι θα επιστρέψει η `example(9)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

Έλεγχε τη σκέψη σου. Η σύγκριση είναι ψευδής, αφού το `9` είναι μικρότερο από το `10`.

Συνέχισε στην εκφώνηση της άσκησης 11, στη σελίδα 46.

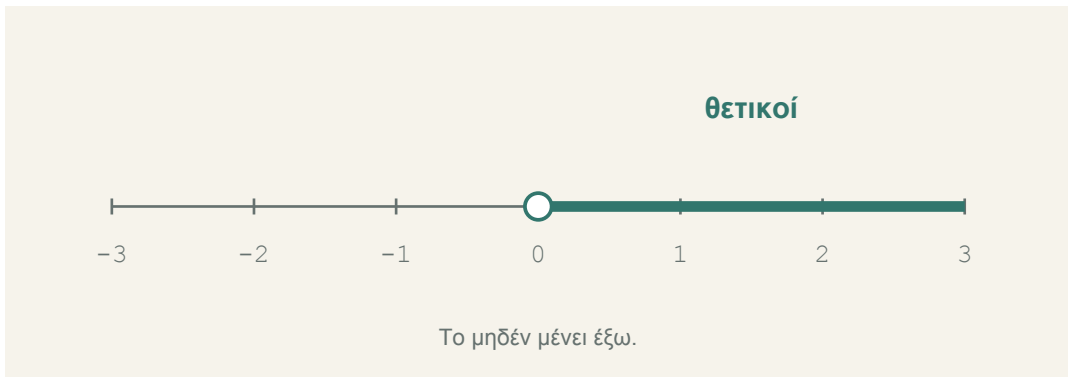
Η ΑΣΚΗΣΗ

11 Είναι θετικός;

Τι θα φτιάξεις

Δέξου έναν ακέραιο n και επέστρεψε αν είναι αυστηρά μεγαλύτερος από το μηδέν.

Όρια: $-1000 \leq n \leq 1000$.



Η ιδέα

Το $>$ διαβάζεται «μεγαλύτερο από». Η λέξη αυστηρά σημαίνει ότι το ίδιο το όριο δεν περιλαμβάνεται. Το μηδέν δεν είναι θετικός αριθμός.

ΕΙΝΑΙ ΘΕΤΙΚΟΣ;

11 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`. Η τιμή `true` σημαίνει αληθές και η `false` ψευδές.

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(3)</code>	<code>true</code>
<code>solve(1)</code>	<code>true</code>
<code>solve(0)</code>	<code>false</code>
<code>solve(-1)</code>	<code>false</code>
<code>solve(1000)</code>	<code>true</code>

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο `c`.

Προσπάθεια: `exercises/11-is-positive/exercise.c`.

Tests: `exercises/11-is-positive/exercise_test.c`.

SH

```
cd exercises/11-is-positive
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο [κεφάλαιο testing](#).

Μικρή βοήθεια

Δοκίμασε στο χαρτί τις τιμές -1, 0 και 1 πριν γράψεις τον έλεγχο.

Η λύση βρίσκεται στη σελίδα 91.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

12 Όταν το όριο συμπεριλαμβάνεται

Έχουμε μια διαθέσιμη ποσότητα και μια απαιτούμενη. Ρωτάμε αν αρκεί. Ακριβώς όση χρειάζεται είναι αρκετή.

C

```
1  #include <stdbool.h>
2
3  bool example(int available, int needed) {
4      return available >= needed;
5  }
```

Διάβασε τις γραμμές

- 01 Το `>=` σημαίνει «μεγαλύτερο ή ίσο». Οι δύο χαρακτήρες γράφονται ενωμένοι και με αυτή τη σειρά. Δεν γράφουμε `=>`.
- 02 Η σύγκριση χρησιμοποιεί δύο παραμέτρους. Το όριο είναι η τιμή του `needed` σε αυτή την κλήση.
- 03 Το αντίστοιχο `<=` σημαίνει «μικρότερο ή ίσο». Και οι δύο συγκρίσεις περιλαμβάνουν την ισότητα.

Ακολούθησε μια κλήση

Η `example(4, 4)` επιστρέφει `true`. Η διαθέσιμη ποσότητα είναι ακριβώς η απαιτούμενη.

Η `example(3, 4)` επιστρέφει `false`. Λείπει μία μονάδα από την απαιτούμενη ποσότητα.

Στάσου λίγο

Τι θα επιστρέψει η `example(5, 4)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

Έλεγξε τη σκέψη σου. Η απάντηση είναι αληθής. Το 5 είναι μεγαλύτερο από το 4.

Συνέχισε στην εκφώνηση της άσκησης 12, στη σελίδα 49.

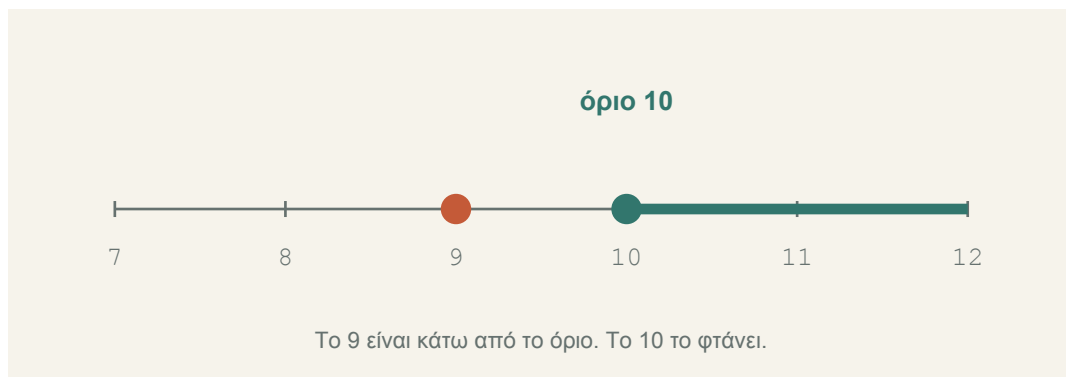
Η ΑΣΚΗΣΗ

12 Έφτασε το όριο;

Τι θα φτιάξεις

Δέξου μια βαθμολογία `score` και ένα απαιτούμενο όριο `threshold`. Επέστρεψε αν η βαθμολογία έφτασε ή ξεπέρασε το όριο.

Όρια: $0 \leq \text{score}, \text{threshold} \leq 1000$.



Η ιδέα

Το $\geq$ διαβάζεται «μεγαλύτερο ή ίσο». Όταν η εκφώνηση λέει «τουλάχιστον» ή «έφτασε το όριο», περιλαμβάνει και την ισότητα.

ΕΦΤΑΣΕ ΤΟ ΟΡΙΟ;

12 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `score`, `threshold`. Η τιμή `true` σημαίνει αληθές και η `false` ψευδές.

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(9, 10)</code>	<code>false</code>
<code>solve(10, 10)</code>	<code>true</code>
<code>solve(11, 10)</code>	<code>true</code>
<code>solve(0, 0)</code>	<code>true</code>
<code>solve(0, 1)</code>	<code>false</code>

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο `c`.

Προσπάθεια: `exercises/12-at-least/exercise.c`.

Tests: `exercises/12-at-least/exercise_test.c`.

SH

```
cd exercises/12-at-least
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο [κεφάλαιο testing](#).

Μικρή βοήθεια

Αν το όριο είναι 10, σκέψου τι πρέπει να γίνει με βαθμολογίες 9, 10 και 11.

Η λύση βρίσκεται στη σελίδα [92](#).

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

13 Μια πράξη μέσα σε σύγκριση

Ενώνουμε δύο πράγματα που ήδη γνωρίζουμε. Για έναν μη αρνητικό ακέραιο, υπόλοιπο 1 στη διαίρεση με το 2 σημαίνει ότι περισεύει μία μονάδα μετά τον σχηματισμό ζευγαριών.

C

```
1  #include <stdbool.h>
2
3  bool example(int value) {
4      return value % 2 == 1;
5  }
```

Διάβασε τις γραμμές

- 01 Πρώτα υπολογίζεται το `value % 2`. Έπειτα το `== 1` συγκρίνει το υπόλοιπο με το 1.
- 02 Η έκφραση έχει αριθμητικό ενδιαμέσο αποτέλεσμα, αλλά η τελική τιμή της είναι λογική. Ο τύπος αποτελέσματος το περιγράφει.
- 03 Εδώ χρησιμοποιούμε μη αρνητικούς ακεραίους. Το υπόλοιπο στη διαίρεση με το 2 είναι τότε είτε 0 είτε 1.

Ακολούθησε μια κλήση

Η `example(7)` επιστρέφει `true`. Από το 7 μένει 1. Συγκρίνουμε αυτό το 1 με το 1.

Η `example(8)` επιστρέφει `false`. Από το 8 μένει 0. Η σύγκριση με το 1 είναι ψευδής.

Στάσου λίγο

Τι θα επιστρέψει η `example(9)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

Έλεγξε τη σκέψη σου. Η απάντηση είναι αληθής. Το 9 αφήνει υπόλοιπο 1.

Συνέχισε στην εκφώνηση της άσκησης 13, στη σελίδα 52.

Η ΑΣΚΗΣΗ

13 Ζυγός αριθμός

Τι θα φτιάξεις

Δέξου έναν μη αρνητικό ακέραιο n . Επέστρεψε αν είναι ζυγός, δηλαδή αν διαιρείται ακριβώς με το 2.

Όρια: $0 \leq n \leq 1000$. Το 0 θεωρείται ζυγός αριθμός.

The diagram illustrates the concept of even and odd numbers using pairs of dots. For the number 4, there are two pairs of dots, each pair enclosed in a white box, labeled 'ζυγός' (even). For the number 5, there are two pairs of dots in white boxes and one single red dot, labeled 'περιττός' (odd). Below the diagram, the text reads: 'Χωρίζουμε σε ζευγάρια και κοιτάμε αν περισσεύει ένα.' (We divide into pairs and see if there is one left over.)

Η ιδέα

Στην άσκηση με τις καραμέλες έβρισκες ένα υπόλοιπο. Εδώ ελέγχεις αν το υπόλοιπο από τη διαίρεση με το 2 είναι μηδέν. Συνδυάζεις δύο πράγματα που έχεις ήδη χρησιμοποιήσει.

ΖΥΓΟΣ ΑΡΙΘΜΟΣ

13 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`. Η τιμή `true` σημαίνει αληθές και η `false` ψευδές.

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(4)</code>	<code>true</code>
<code>solve(3)</code>	<code>false</code>
<code>solve(0)</code>	<code>true</code>
<code>solve(1)</code>	<code>false</code>
<code>solve(2)</code>	<code>true</code>

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο `c`.

Προσπάθεια: `exercises/13-is-even/exercise.c`.

Tests: `exercises/13-is-even/exercise_test.c`.

SH

```
cd exercises/13-is-even
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο [κεφάλαιο testing](#).

Μικρή βοήθεια

Πρώτα βρες τι μένει όταν χωρίσεις τον αριθμό σε ζευγάρια. Μετά έλεγξε αν έμεινε κάτι.

Η λύση βρίσκεται στη σελίδα [93](#).

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

14 Δύο δρόμοι με if και else

Μέχρι εδώ επιστρέψαμε τη σύγκριση. Τώρα τη χρησιμοποιούμε για να επιλέξουμε αριθμό. Πάνω από το 10 δίνουμε 1. Σε κάθε άλλη περίπτωση δίνουμε 0.

C

```
1  int example(int value) {  
2      if (value > 10) {  
3          return 1;  
4      } else {  
5          return 0;  
6      }  
7  }
```

Διάβασε τις γραμμές

- 01 Το `if (value > 10)` ελέγχει τη συνθήκη στις παρενθέσεις. Το `else` καλύπτει την άλλη περίπτωση. Τα άγκιστρα περικλείουν τους κλάδους. Μετά το `if (...)` δεν προσθέτουμε `;`. Το `;` μπαίνει μετά τις εντολές `return`.
- 02 Κλάδος είναι ο κώδικας που αντιστοιχεί σε μία επιλογή. Σε αυτή την κλήση εκτελείται μόνο ένας από τους δύο κλάδους.
- 03 Η συνθήκη είναι λογική, αλλά η απάντηση της συνάρτησης είναι ακέραια. Άλλο ο έλεγχος και άλλο η τιμή που επιλέγει.

Ακολούθησε μια κλήση

Η `example(12)` επιστρέφει 1. Η συνθήκη είναι αληθής. Επιλέγεται ο πρώτος κλάδος.

Η `example(10)` επιστρέφει 0. Η ισότητα δεν περνά το `>`. Επιλέγεται ο κλάδος `else`.

Στάσου λίγο

Τι θα επιστρέψει η `example(3)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

Έλεγξε τη σκέψη σου. Η τιμή είναι 0. Εκτελείται ο κλάδος `else`.

Συνέχισε στην εκφώνηση της άσκησης 14, στη σελίδα 55.

Η ΑΣΚΗΣΗ

14 Απόσταση από το μηδέν

Τι θα φτιάξεις

Δέξου έναν ακέραιο `n` και επέστρεψε την απόστασή του από το μηδέν. Για παράδειγμα, τόσο το `-4` όσο και το `4` απέχουν `4`. Χρησιμοποίησε μια συνθήκη, χωρίς έτοιμη συνάρτηση απόλυτης τιμής.

Όρια: $-1000 \leq n \leq 1000$.



Η ιδέα

Το `if` επιλέγει τι θα γίνει όταν μια συνθήκη είναι αληθής. Το `else` καλύπτει την άλλη περίπτωση. Για έναν αρνητικό αριθμό, το `-n` αλλάζει το πρόσημο. Για μη αρνητικό αριθμό κρατάμε την τιμή του.

ΑΠΟΣΤΑΣΗ ΑΠΟ ΤΟ ΜΗΔΕΝ

14 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά n .

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(-4)</code>	4
<code>solve(4)</code>	4
<code>solve(0)</code>	0
<code>solve(-1)</code>	1
<code>solve(1)</code>	1

Αποθήκευσε την προσπάθειά. Ξεκίνα από τον φάκελο **c**.

Προσπάθεια: `exercises/14-absolute/exercise.c`.

Tests: `exercises/14-absolute/exercise_test.c`.

SH

```
cd exercises/14-absolute
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο **testing**.

Μικρή βοήθεια

Χώρισε τις εισόδους σε δύο περιπτώσεις: κάτω από το μηδέν και όλες τις υπόλοιπες.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Η λύση βρίσκεται στη σελίδα **94**.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

15 Ένας κλάδος μπορεί να επιστρέφει παράμετρο

Θα κρατήσουμε τον μικρότερο από δύο αριθμούς. Χρησιμοποιούμε την ίδια επιλογή με πριν, αλλά κάθε κλάδος επιστρέφει μια από τις τιμές που πήρε η συνάρτηση.

C

```
1  int example(int a, int b) {  
2      if (a > b) {  
3          return b;  
4      } else {  
5          return a;  
6      }  
7  }
```

Διάβασε τις γραμμές

- 01 Το `if` ελέγχει αν το `a` είναι μεγαλύτερο από το `b`. Αν ναι, μικρότερο είναι το `b`. Διαφορετικά κρατάμε το `a`.
- 02 Μια παράμετρος μπορεί να μπει στη θέση μιας σταθερής απάντησης. Ο κλάδος επιστρέφει την τρέχουσα τιμή της.
- 03 Η ισότητα οδηγεί στον δεύτερο κλάδο, επειδή χρησιμοποιούμε `>`. Εδώ αυτό είναι σωστό, αφού οι τιμές είναι ίδιες.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

15 Από τον κώδικα στην τιμή



Ακολουθήσε μια κλήση

Η `example(3, 8)` επιστρέφει 3. Το $3 > 8$ είναι ψευδές. Επιστρέφεται το `a`.

Η `example(9, 4)` επιστρέφει 4. Το $9 > 4$ είναι αληθές. Επιστρέφεται το `b`.

Στάσου λίγο

Τι θα επιστρέψει η `example(5, 5)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

Έλεγξε τη σκέψη σου. Η τιμή είναι 5. Η συνθήκη είναι ψευδής και ο άλλος κλάδος επιστρέφει το `a`. Με ίσες τιμές και οι δύο επιλογές δίνουν το ίδιο.

Συνέχισε στην εκφώνηση της άσκησης 15, στη σελίδα 59.

Η ΑΣΚΗΣΗ

15 Ο μεγαλύτερος από τους δύο

Τι θα φτιάξεις

Δέξου δύο ακέραιους a και b και επέστρεψε τον μεγαλύτερο. Αν είναι ίσοι, επέστρεψε αυτή την κοινή τιμή. Χρησιμοποίησε `if`, χωρίς έτοιμη συνάρτηση μέγιστου.

Όρια: $-1000 \leq a, b \leq 1000$.



Η ιδέα

Η συνθήκη δεν χρειάζεται να συγκρίνει μια παράμετρο με έναν σταθερό αριθμό. Μπορεί να συγκρίνει τις δύο παραμέτρους μεταξύ τους και να επιλέξει ποια τιμή θα επιστρέψει.

Ο ΜΕΓΑΛΥΤΕΡΟΣ ΑΠΟ ΤΟΥΣ ΔΥΟ

15 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά *a*, *b*.

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(7, 2)</code>	7
<code>solve(2, 7)</code>	7
<code>solve(5, 5)</code>	5
<code>solve(-7, -2)</code>	-2
<code>solve(0, -1)</code>	0

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **c**.

Προσπάθεια: `exercises/15-larger/exercise.c`.

Tests: `exercises/15-larger/exercise_test.c`.

SH

```
cd exercises/15-larger
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο **testing**.

Μικρή βοήθεια

Αν το *a* δεν είναι μεγαλύτερο από το *b*, ποια από τις δύο τιμές μπορείς να επιστρέψεις;

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Η λύση βρίσκεται στη σελίδα 95.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

16 Μια τρίτη περίπτωση

Ξεχωρίζουμε τρεις περιπτώσεις γύρω από το 10. Πάνω από δέκα δίνουμε 2, ακριβώς δέκα δίνουμε 1 και κάτω από δέκα δίνουμε 0. Ελέγχουμε με αυτή τη σειρά.

C

```
1  int example(int value) {  
2      if (value > 10) {  
3          return 2;  
4      } else if (value == 10) {  
5          return 1;  
6      } else {  
7          return 0;  
8      }  
9  }
```

Διάβασε τις γραμμές

- 01 Το `else if` σημαίνει «αλλιώς, αν». Η δεύτερη συνθήκη ελέγχεται μόνο όταν η πρώτη είναι ψευδής. Το τελικό `else` καλύπτει όσα απέμειναν.
- 02 Διαβάζουμε από πάνω προς τα κάτω. Η πρώτη συνθήκη που πετυχαίνει καθορίζει το αποτέλεσμα. Δεν αθροίζουμε τις απαντήσεις.
- 03 Και οι τρεις κλάδοι δίνουν ακέραιο. Χρειάζεται απάντηση και για τιμή μικρότερη, ίση και μεγαλύτερη από το όριο.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

16 Από τον κώδικα στην τιμή



Ακολουθήσε μια κλήση

Η `example(12)` επιστρέφει 2. Περνά η πρώτη συνθήκη. Δεν χρειάζεται ο δεύτερος έλεγχος.

Η `example(10)` επιστρέφει 1. Αποτυγχάνει ο πρώτος έλεγχος. Περνά η ισότητα.

Στάσου λίγο

Τι θα επιστρέψει η `example(6)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

Έλεγε τη σκέψη σου. Η τιμή είναι 0. Αποτυγχάνουν και οι δύο έλεγχοι, οπότε επιλέγεται η τελευταία περίπτωση.

Συνέχισε στην εκφώνηση της άσκησης 16, στη σελίδα 63.

Η ΑΣΚΗΣΗ

16 Τρεις περιπτώσεις

Τι θα φτιάξεις

Δέξου έναν ακέραιο n . Επέστρεψε -1 αν είναι αρνητικός, 0 αν είναι μηδέν και 1 αν είναι θετικός.

Όρια: $-1000 \leq n \leq 1000$.

αρνητικός

 -1

μηδέν

0

θετικός

1

Τρεις περιπτώσεις. Μία απάντηση για καθεμία.

Η ιδέα

Μερικές αποφάσεις έχουν τρεις περιπτώσεις. Ελέγχουμε την πρώτη και, αν δεν ισχύει, συνεχίζουμε στη δεύτερη. Ό,τι απομένει καλύπτεται από την τελευταία περίπτωση.

ΤΡΕΙΣ ΠΕΡΙΠΤΩΣΕΙΣ

16 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά n .

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(-7)</code>	-1
<code>solve(0)</code>	0
<code>solve(42)</code>	1
<code>solve(-1)</code>	-1
<code>solve(1)</code>	1

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **c**.

Προσπάθεια: `exercises/16-sign/exercise.c`.

Tests: `exercises/16-sign/exercise_test.c`.

SH

```
cd exercises/16-sign
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο **testing**.

Μικρή βοήθεια

Έλεγξε πρώτα αν ο αριθμός είναι κάτω από το μηδέν. Αν δεν είναι, τι χρειάζεται να ξεχωρίσεις ακόμη;

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Η λύση βρίσκεται στη σελίδα 96.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

17 Δύο συνθήκες πρέπει να ισχύουν μαζί

Μια επιτρεπτή τιμή πρέπει να είναι τουλάχιστον 2 και το πολύ ίση με το `limit`. Χρειαζόμαστε τη λογική πράξη «και». Το κάτω όριο μένει σταθερό, άρα συνεχίζουμε με δύο παραμέτρους.

C

```
1  #include <stdbool.h>
2
3  bool example(int value, int limit) {
4      return value >= 2 && value <= limit;
5  }
```

Διάβασε τις γραμμές

- 01 Το `&&` ενώνει δύο λογικές συγκρίσεις. Δίνει αληθές μόνο όταν και οι δύο δίνουν αληθές.
- 02 Η πρώτη σύγκριση περιλαμβάνει το κάτω όριο 2. Η δεύτερη περιλαμβάνει το πάνω όριο `limit`. Αν η πρώτη είναι ψευδής, η δεύτερη δεν χρειάζεται να υπολογιστεί.
- 03 Κάθε σύγκριση έχει και το δικό της `value`. Γράφουμε τους δύο ελέγχους ολόκληρους για να φαίνονται καθαρά.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

17 Από τον κώδικα στην τιμή



Ακολουθήσε μια κλήση

Η `example(5, 8)` επιστρέφει `true`. Ισχύουν και το κάτω και το πάνω όριο.

Η `example(1, 8)` επιστρέφει `false`. Η πρώτη συνθήκη είναι ψευδής. Αυτό αρκεί για ψευδές αποτέλεσμα.

Στάσου λίγο

Τι θα επιστρέψει η `example(9, 8)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

Έλεγξε τη σκέψη σου. Η απάντηση είναι ψευδής. Το 9 περνά το κάτω όριο, αλλά ξεπερνά το πάνω όριο 8.

Συνέχισε στην εκφώνηση της άσκησης 17, στη σελίδα 67.

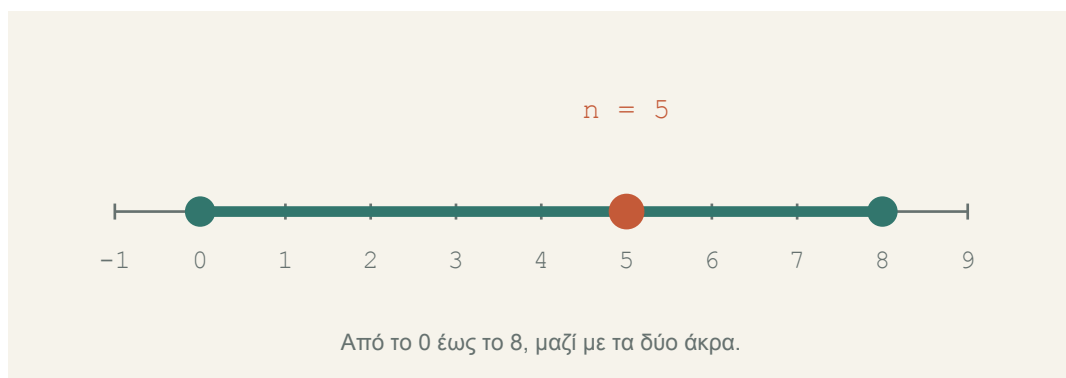
Η ΑΣΚΗΣΗ

17 Μέσα στα όρια

Τι θα φτιάξεις

Δέξου έναν ακέραιο `n` και ένα μη αρνητικό πάνω όριο `high`. Επέστρεψε αν ο αριθμός βρίσκεται από το 0 έως το `high`, συμπεριλαμβανομένων και των δύο άκρων. Το κάτω όριο είναι πάντα 0.

Όρια: $-1000 \leq n \leq 1000$ και $0 \leq \text{high} \leq 1000$.



Η ιδέα

Για να βρίσκεται ένας αριθμός μέσα σε ένα διάστημα, πρέπει να ικανοποιούνται δύο έλεγχοι μαζί. Το λογικό «και» γράφεται `and` στην Python και στην Elixir και `&&` στις υπόλοιπες γλώσσες.

ΜΕΣΑ ΣΤΑ ΟΡΙΑ

17 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`, `high`. Η τιμή `true` σημαίνει αληθές και η `false` ψευδές.

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(5, 8)</code>	<code>true</code>
<code>solve(0, 8)</code>	<code>true</code>
<code>solve(8, 8)</code>	<code>true</code>
<code>solve(-1, 8)</code>	<code>false</code>
<code>solve(9, 8)</code>	<code>false</code>

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο `c`.

Προσπάθεια: `exercises/17-in-range/exercise.c`.

Tests: `exercises/17-in-range/exercise_test.c`.

SH

```
cd exercises/17-in-range
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο [κεφάλαιο testing](#).

Μικρή βοήθεια

Γράψε δύο ξεχωριστές προτάσεις: «ο αριθμός δεν είναι κάτω από το μηδέν» και «δεν είναι πάνω από το πάνω όριο».

Η λύση βρίσκεται στη σελίδα 97.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

18 Ο διαιρέτης μπορεί να αλλάξει

Θέλουμε να χωρίσουμε ένα πλήθος σε ίσες ομάδες, χωρίς περίσσευμα. Το μέγεθος της ομάδας δίνεται στη δεύτερη παράμετρο. Αυτό συνδυάζει γνωστή σύνταξη, δεν προσθέτει νέα εντολή.

C

```
1  #include <stdbool.h>
2
3  bool example(int total, int size) {
4      return total % size == 0;
5  }
```

Διάβασε τις γραμμές

- 01 Στο `total % size`, το `size` παίρνει τον ρόλο του διαιρέτη. Χρησιμοποιούμε την τιμή της παραμέτρου, όχι σταθερό αριθμό.
- 02 Το `== 0` ελέγχει αν δεν περίσσεψε τίποτα. Δεν επιστρέφουμε το ίδιο το υπόλοιπο αλλά το αποτέλεσμα της σύγκρισης.
- 03 Το `size` πρέπει να είναι θετικό. Μηδενικό `total` με θετικό `size` αφήνει μηδενικό υπόλοιπο και περνά τον έλεγχο.

Ακολούθησε μια κλήση

Η `example(12, 4)` επιστρέφει `true`. Το 12 χωρίζεται σε ομάδες των 4 χωρίς υπόλοιπο.

Η `example(14, 4)` επιστρέφει `false`. Το 14 αφήνει υπόλοιπο 2.

Στάσου λίγο

Τι θα επιστρέψει η `example(15, 5)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

Έλεγξε τη σκέψη σου. Η απάντηση είναι αληθής. Το 15 χωρίζεται σε τρεις ομάδες των 5.

Συνέχισε στην εκφώνηση της άσκησης 18, στη σελίδα 70.

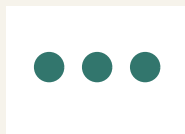
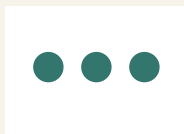
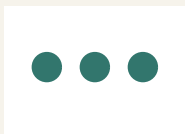
Η ΑΣΚΗΣΗ

18 Χωρίς υπόλοιπο

Τι θα φτιάξεις

Δέξου έναν μη αρνητικό ακέραιο `n` και έναν θετικό ακέραιο `divisor`. Επέστρεψε αν ο πρώτος είναι πολλαπλάσιο του δεύτερου, δηλαδή αν η διαίρεσή τους δεν αφήνει υπόλοιπο.

Όρια: $0 \leq n \leq 1000$ και $1 \leq \text{divisor} \leq 1000$. Το 0 είναι πολλαπλάσιο κάθε επιτρεπτού διαιρέτη.



Το 12 χωρίζεται σε ομάδες των 3 χωρίς υπόλοιπο.

Η ιδέα

Ο έλεγχος για ζυγό αριθμό ήταν μια ειδική περίπτωση με διαιρέτη το 2. Εδώ ο διαιρέτης δίνεται στη συνάρτηση. Η ιδέα παραμένει ίδια.

ΧΩΡΙΣ ΥΠΟΛΟΙΠΟ

18 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`, `divisor`. Η τιμή `true` σημαίνει αληθές και η `false` ψευδές.

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(12, 3)</code>	<code>true</code>
<code>solve(13, 3)</code>	<code>false</code>
<code>solve(0, 7)</code>	<code>true</code>
<code>solve(7, 1)</code>	<code>true</code>
<code>solve(2, 5)</code>	<code>false</code>

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο `c`.

Προσπάθεια: `exercises/18-is-multiple/exercise.c`.

Tests: `exercises/18-is-multiple/exercise_test.c`.

SH

```
cd exercises/18-is-multiple
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο [κεφάλαιο testing](#).

Μικρή βοήθεια

Ξαναδές την άσκηση 13 και σκέψου ποια σταθερά πρέπει να γίνει παράμετρος.

Η λύση βρίσκεται στη σελίδα 98.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

19 Η απόφαση μπορεί να δίνει κείμενο

Για έναν αριθμό που σχηματίζει πλήρη ζευγάρια επιστρέφουμε το κείμενο "Pair". Διαφορετικά επιστρέφουμε κενό κείμενο. Οι κλάδοι παραμένουν δύο, αλλά ο τύπος της απάντησης αλλάζει.

C

```
1  const char *example(int value) {  
2      if (value % 2 == 0) {  
3          return "Pair";  
4      } else {  
5          return "";  
6      }  
7  }
```

Διάβασε τις γραμμές

- 01 Ο τύπος πριν από το όνομα γίνεται `const char *`, όπως στο πρώτο μάθημα. Και οι δύο κλάδοι επιστρέφουν σταθερό κείμενο.
- 02 Το "" έχει μηδέν χαρακτήρες. Το " " θα είχε ένα κενό και θα ήταν διαφορετική απάντηση.
- 03 Η σύγκριση παραμένει λογική. Χρησιμοποιείται για την επιλογή κειμένου και δεν είναι η τελική απάντηση.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

19 Από τον κώδικα στην τιμή



Ακολουθήσε μια κλήση

Η `example(8)` επιστρέφει `"Pair"`. Δεν μένει υπόλοιπο. Επιλέγεται το πρώτο κείμενο.

Η `example(7)` επιστρέφει `" "`. Μένει υπόλοιπο 1. Επιλέγεται το κενό κείμενο.

Στάσου λίγο

Τι θα επιστρέψει η `example(0)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

Έλεγξε τη σκέψη σου. Η τιμή είναι `"Pair"`. Το μηδέν αφήνει μηδενικό υπόλοιπο στη διαίρεση με το 2.

Συνέχισε στην εκφώνηση της άσκησης 19, στη σελίδα 74.

Η ΑΣΚΗΣΗ

19 Η πρώτη λέξη του FizzBuzz

Τι θα φτιάξεις

Δέξου έναν θετικό ακέραιο n . Αν είναι πολλαπλάσιο του 3, επέστρεψε `"Fizz"`. Σε κάθε άλλη περίπτωση επέστρεψε το κενό κείμενο `" "`.

Όρια: $1 \leq n \leq 1000$. Επιστρέφουμε μόνο μια ετικέτα. Δεν μετατρέπουμε τον αριθμό σε κείμενο.



Μερικές φορές η απάντηση είναι κενό κείμενο.

Η ιδέα

Έχεις ήδη επιστρέψει κείμενο στην πρώτη άσκηση και έχεις ελέγξει πολλαπλάσια. Τώρα τα συνδυάζεις. Το `if` μπορεί να επιλέξει κείμενο όπως επέλεγε αριθμό στην άσκηση 15.

Η ΠΡΩΤΗ ΛΕΞΗ ΤΟΥ FIZZBUZZ

19 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά n .

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(3)</code>	<code>"Fizz"</code>
<code>solve(1)</code>	<code>""</code>
<code>solve(5)</code>	<code>""</code>
<code>solve(6)</code>	<code>"Fizz"</code>
<code>solve(15)</code>	<code>"Fizz"</code>

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο `c`.

Προσπάθεια: `exercises/19-fizz/exercise.c`.

Tests: `exercises/19-fizz/exercise_test.c`.

SH

```
cd exercises/19-fizz
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο `testing`.

Μικρή βοήθεια

Πρώτα απάντησε αν ο αριθμός διαιρείται ακριβώς με το 3. Μετά διάλεξε ένα από τα δύο κείμενα.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Η λύση βρίσκεται στη σελίδα 99.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

20 Η σειρά των κανόνων έχει σημασία

Δίνουμε τις ετικέτες "Pair" για πολλαπλάσια του 2 και "Trio" για πολλαπλάσια του 3. Όταν ισχύουν και οι δύο κανόνες δίνουμε "PairTrio". Συνδυάζουμε όσα ήδη μάθαμε.

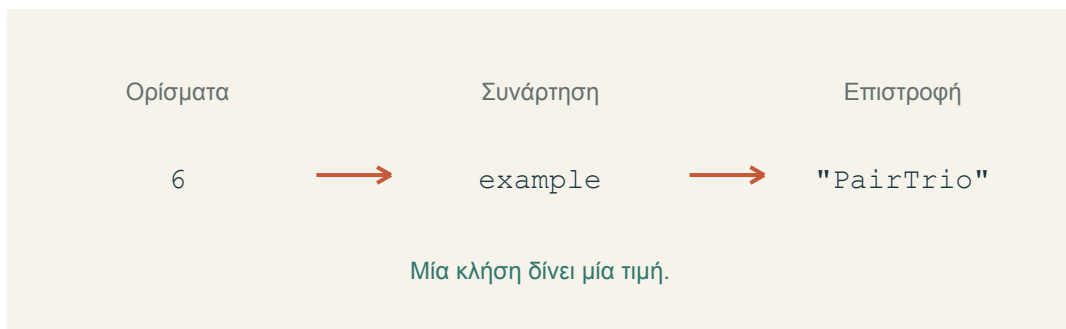
```
c
1  const char *example(int value) {
2      if (value % 2 == 0 && value % 3 == 0) {
3          return "PairTrio";
4      } else if (value % 2 == 0) {
5          return "Pair";
6      } else if (value % 3 == 0) {
7          return "Trio";
8      } else {
9          return "";
10     }
11 }
```

Διάβασε τις γραμμές

- 01 Ο πρώτος έλεγχος ενώνει με && τους δύο κανόνες. Πρέπει να προηγείται, επειδή κάθε κοινό πολλαπλάσιο θα περνούσε και κάποιον από τους απλούς ελέγχους.
- 02 Η αλυσίδα ελέγχων επιλέγει τον πρώτο κλάδο που πετυχαίνει. Το τελικό else επιστρέφει κενό κείμενο.
- 03 Δεν εμφανίζουμε πολλά αποτελέσματα και δεν επαναλαμβάνουμε την εκτέλεση μέσα στη συνάρτηση. Μία κλήση δίνει μία ετικέτα.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

20 Από τον κώδικα στην τιμή



Ακολουθήσε μια κλήση

Η `example(6)` επιστρέφει `"PairTrio"`. Ισχύουν και οι δύο κανόνες. Επιλέγεται ο πρώτος κλάδος.

Η `example(4)` επιστρέφει `"Pair"`. Ισχύει μόνο ο κανόνας του 2.

Η `example(9)` επιστρέφει `"Trio"`. Ισχύει μόνο ο κανόνας του 3.

Η `example(5)` επιστρέφει `" "`. Δεν ισχύει κανένας από τους δύο κανόνες.

Στάσου λίγο

Τι θα επιστρέψει η `example(12)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

Έλεγε τη σκέψη σου. Η τιμή είναι `"PairTrio"`. Ο κοινός έλεγχος πετυχαίνει πρώτος. Αν βάzaμε πρώτο τον απλό έλεγχο του 2, θα παίρναμε λάθος `"Pair"`.

Συνέχισε στην εκφώνηση της άσκησης 20, στη σελίδα 78.

Η ΑΣΚΗΣΗ

20 Οι κανόνες του FizzBuzz

Τι θα φτιάξεις

Δέξου έναν θετικό ακέραιο n και επέστρεψε τη λέξη που του αντιστοιχεί: "FizzBuzz" αν είναι πολλαπλάσιο και του 3 και του 5, "Fizz" αν είναι μόνο του 3, "Buzz" αν είναι μόνο του 5 και "" αν δεν είναι κανενός από τα δύο.

Όρια: $1 \leq n \leq 1000$. Η συνάρτηση επιστρέφει μόνο την ετικέτα για έναν αριθμό. Για παράδειγμα, στο 7 επιστρέφει "", όχι "7".

n	με το 3;	με το 5;	απάντηση
15	ναι	ναι	FizzBuzz
9	ναι	όχι	Fizz
10	όχι	ναι	Buzz
7	όχι	όχι	" "

Η κοινή περίπτωση προηγείται.

Η ιδέα

Ένας αριθμός μπορεί να περνά δύο ελέγχους ταυτόχρονα. Το 15 είναι πολλαπλάσιο και του 3 και του 5. Σε μια αλυσίδα αποφάσεων, η πρώτη περίπτωση που ταιριάζει καθορίζει την απάντηση.

ΟΙ ΚΑΝΟΝΕΣ ΤΟΥ FIZZBUZZ

20 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά n .

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(15)</code>	<code>"FizzBuzz"</code>
<code>solve(9)</code>	<code>"Fizz"</code>
<code>solve(10)</code>	<code>"Buzz"</code>
<code>solve(7)</code>	<code>""</code>
<code>solve(1)</code>	<code>""</code>

Αποθήκευσε την προσπάθειά. Ξεκίνα από τον φάκελο `c`.

Προσπάθεια: `exercises/20-fizzbuzz-label/exercise.c`.

Tests: `exercises/20-fizzbuzz-label/exercise_test.c`.

SH

```
cd exercises/20-fizzbuzz-label
cc -std=c17 -Wall -Wextra -Werror -pedantic \
  -I../../lib/unity exercise.c exercise_test.c \
  ../../lib/unity/unity.c -o exercise_tests
./exercise_tests
cd ../../
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο `testing`.

Μικρή βοήθεια

Πριν ελέγξεις ξεχωριστά το 3 και το 5, ξεχώρισε την περίπτωση όπου ισχύουν και τα δύο.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Η λύση βρίσκεται στη σελίδα 100.

Οι λύσεις

Σύγκρινε τη σκέψη σου με μια πιθανή λύση.

Μετά κλείσε το βιβλίο και ξαναδοκίμασε μόνος σου.

01	02	03	04	05	06	07	08	09	10
11	12	13	14	15	16	17	18	19	20

Η ΛΥΣΗ

01 Μια σταθερή απάντηση

Η σκέψη

Δεν υπάρχει υπολογισμός. Η συνάρτηση επιστρέφει το συγκεκριμένο κείμενο κάθε φορά που καλείται.

C

```
1  #include "exercise.h"
2
3  const char *solve(void) {
4      return "Hello!";
5  }
```

Διάβασε τον κώδικα

Το `const char *` είναι δείκτης σε χαρακτήρες που δεν αλλάζουμε μέσω αυτού. Εδώ επιστρέφει τη διεύθυνση ενός σταθερού κειμένου. Το `(void)` δηλώνει ότι η συνάρτηση δεν δέχεται παραμέτρους.

Πρόσεξε αυτό

Αν εμφανίσεις το μήνυμα στην οθόνη χωρίς να το επιστρέψεις, το test δεν θα πάρει την τιμή που περιμένει.

Η ΛΥΣΗ

02 Ο ίδιος αριθμός

Η σκέψη

Επιστρέφουμε την παράμετρο χωρίς να την αλλάξουμε. Η ίδια γραμμή δουλεύει για θετικούς αριθμούς, αρνητικούς και μηδέν.

```
C
1  #include "exercise.h"
2
3  int solve(int n) {
4      return n;
5  }
```

Διάβασε τον κώδικα

Η δήλωση στο `exercise.h` περιγράφει την ίδια συνάρτηση που συμπληρώνεις στο αρχείο `.c`. Το `return` δίνει το αποτέλεσμα.

Πρόσεξε αυτό

Το `"n"` είναι το γράμμα `n` ως κείμενο. Το `n` είναι η τιμή της παραμέτρου.

Η ΛΥΣΗ

03 Ο επόμενος αριθμός

Η σκέψη

Υπολογίζουμε $n + 1$ και επιστρέφουμε το αποτέλεσμα. Για -1 , η πρόσθεση δίνει 0 .

c

```
1  #include "exercise.h"
2
3  int solve(int n) {
4      return n + 1;
5  }
```

Διάβασε τον κώδικα

Η δήλωση στο `exercise.h` περιγράφει την ίδια συνάρτηση που συμπληρώνεις στο αρχείο `.c`. Το `return` δίνει το αποτέλεσμα.

Πρόσεξε αυτό

Δεν χρειάζεται να αλλάξεις την παράμετρο ή να γράψεις ξεχωριστό κανόνα για τους αρνητικούς.

Η ΛΥΣΗ

04 Δύο αριθμοί μαζί

Η σκέψη

Προσθέτουμε τις δύο παραμέτρους. Δεν χρησιμοποιούμε σταθερούς αριθμούς από τα παραδείγματα.

```
C
1  #include "exercise.h"
2
3  int solve(int a, int b) {
4      return a + b;
5  }
```

Διάβασε τον κώδικα

Η δήλωση στο `exercise.h` περιγράφει την ίδια συνάρτηση που συμπληρώνεις στο αρχείο `.c`. Το `return` δίνει το αποτέλεσμα.

Πρόσεξε αυτό

Η λύση πρέπει να δουλεύει με οποιοδήποτε ζευγάρι μέσα στα όρια, όχι μόνο με το πρώτο παράδειγμα.

Η ΛΥΣΗ

05 Το διπλάσιο

Η σκέψη

Πολλαπλασιάζουμε το n με το 2. Το μηδέν παραμένει μηδέν και οι αρνητικοί αριθμοί παραμένουν αρνητικοί.

C

```
1  #include "exercise.h"
2
3  int solve(int n) {
4      return n * 2;
5  }
```

Διάβασε τον κώδικα

Η δήλωση στο `exercise.h` περιγράφει την ίδια συνάρτηση που συμπληρώνεις στο αρχείο `.c`. Το `return` δίνει το αποτέλεσμα.

Πρόσεξε αυτό

Το $n + 2$ προσθέτει δύο μονάδες. Δεν δίνει γενικά το διπλάσιο.

Η ΛΥΣΗ

06 Από λεπτά σε δευτερόλεπτα

Η σκέψη

Δίνουμε όνομα στην τιμή 60, τα δευτερόλεπτα ενός λεπτού. Πολλαπλασιάζουμε το `minutes` με αυτό το όνομα, αποθηκεύουμε το αποτέλεσμα στο `seconds` και το επιστρέφουμε. Ο υπολογιστής πολλαπλασιάζει τις τιμές των δύο ονομάτων. Ο τελεστής `*` είναι ο ίδιος με εκείνον της προηγούμενης άσκησης.

C

```
1  #include "exercise.h"
2
3  int solve(int minutes) {
4      int seconds_per_minute = 60;
5      int seconds = minutes * seconds_per_minute;
6      return seconds;
7  }
```

Διάβασε τον κώδικα

Το `int seconds_per_minute` δηλώνει μια τοπική ακέραια μεταβλητή. Το `=` της δίνει την τιμή 60. Με τον ίδιο τρόπο το `seconds` παίρνει το αποτέλεσμα του πολλαπλασιασμού.

Πρόσεξε αυτό

Τα tests ελέγχουν το αριθμητικό αποτέλεσμα. Δεν μπορούν να βεβαιώσουν ότι έδωσες όνομα στην τιμή 60 ή ότι χρησιμοποίησες το `seconds`. Έλεγξέ τα και διαβάζοντας τον κώδικά σου.

Η ΛΥΣΗ

07 Η διαφορά δύο αριθμών

Η σκέψη

Επιστρέφουμε $a - b$. Η πρώτη παράμετρος είναι η αρχική τιμή και η δεύτερη αυτή που αφαιρούμε. Αν αφαιρέσουμε έναν αρνητικό αριθμό, το αποτέλεσμα αυξάνεται. Για -4 και -6 έχουμε $-4 - (-6) = 2$.

C

```
1  #include "exercise.h"
2
3  int solve(int a, int b) {
4      return a - b;
5  }
```

Διάβασε τον κώδικα

Η δήλωση στο `exercise.h` περιγράφει την ίδια συνάρτηση που συμπληρώνεις στο αρχείο `.c`. Το `return` δίνει το αποτέλεσμα.

Πρόσεξε αυτό

Το $b - a$ αντιστρέφει τη σειρά και συνήθως αλλάζει το αποτέλεσμα. Δοκίμασε δύο διαφορετικούς αριθμούς για να ελέγξεις τη σειρά.

Η ΛΥΣΗ

08 Γύρω από το ορθογώνιο

Η σκέψη

Προσθέτουμε πλάτος και ύψος και πολλαπλασιάζουμε ολόκληρο το άθροισμα με το 2. Ισοδύναμα θα μπορούσαμε να προσθέσουμε και τις τέσσερις πλευρές.

C

```
1  #include "exercise.h"
2
3  int solve(int width, int height) {
4      return 2 * (width + height);
5  }
```

Διάβασε τον κώδικα

Η δήλωση στο `exercise.h` περιγράφει την ίδια συνάρτηση που συμπληρώνεις στο αρχείο `.c`. Το `return` δίνει το αποτέλεσμα.

Πρόσεξε αυτό

Το `2 * width + height` διπλασιάζει μόνο το πλάτος.

Η ΛΥΣΗ

09 Τι περισσεύει

Η σκέψη

Παίρνουμε το υπόλοιπο της διαίρεσης `candies` με `children`. Αν οι καραμέλες μοιράζονται ακριβώς, το αποτέλεσμα είναι 0.

C

```
1  #include "exercise.h"
2
3  int solve(int candies, int children) {
4      return candies % children;
5  }
```

Διάβασε τον κώδικα

Η δήλωση στο `exercise.h` περιγράφει την ίδια συνάρτηση που συμπληρώνεις στο αρχείο `.c`. Το `return` δίνει το αποτέλεσμα.

Πρόσεξε αυτό

Η σειρά έχει σημασία. Το υπόλοιπο 8 διά 3 δεν είναι το ίδιο με το υπόλοιπο 3 διά 8. Δεν διαιρούμε ποτέ με μηδέν εδώ.

Η ΛΥΣΗ

10 Είναι μηδέν;

Η σκέψη

Συγκρίνουμε το `n` με το 0. Δεν χρειάζεται ακόμη να γράψουμε `if`, γιατί η σύγκριση δίνει ήδη τη σωστή λογική τιμή.

C

```
1  #include "exercise.h"
2
3  bool solve(int n) {
4      return n == 0;
5  }
```

Διάβασε τον κώδικα

Η δήλωση στο `exercise.h` περιγράφει την ίδια συνάρτηση που συμπληρώνεις στο αρχείο `.c`. Το `return` δίνει το αποτέλεσμα.

Πρόσεξε αυτό

Η απάντηση είναι λογική τιμή. Τα κείμενα `"true"` και `"false"` είναι διαφορετικός τύπος. Το `=` δεν είναι ο τελεστής ισότητας.

Η ΛΥΣΗ

11 Είναι θετικός;

Η σκέψη

Επιστρέφουμε την τιμή της σύγκρισης $n > 0$.

C

```
1  #include "exercise.h"
2
3  bool solve(int n) {
4      return n > 0;
5  }
```

Διάβασε τον κώδικα

Η δήλωση στο `exercise.h` περιγράφει την ίδια συνάρτηση που συμπληρώνεις στο αρχείο `.c`. Το `return` δίνει το αποτέλεσμα.

Πρόσεξε αυτό

Αν γράψεις `>=`, το μηδέν θα δώσει λάθος απάντηση.

Η ΛΥΣΗ

12 Έφτασε το όριο;

Η σκέψη

Συγκρίνουμε τη βαθμολογία με το όριο χρησιμοποιώντας `>=`. Το όριο δίνεται ως παράμετρος και μπορεί να αλλάζει σε κάθε κλήση.

```
C
1  #include "exercise.h"
2
3  bool solve(int score, int threshold) {
4      return score >= threshold;
5  }
```

Διάβασε τον κώδικα

Η δήλωση στο `exercise.h` περιγράφει την ίδια συνάρτηση που συμπληρώνεις στο αρχείο `.c`. Το `return` δίνει το αποτέλεσμα.

Πρόσεξε αυτό

Μην γράφεις ένα σταθερό όριο από κάποιο παράδειγμα. Χρησιμοποίησε το `threshold`.

Η ΛΥΣΗ

13 Ζυγός αριθμός

Η σκέψη

Υπολογίζουμε το υπόλοιπο με το 2 και το συγκρίνουμε με το 0. Η σύγκριση δίνει τη λογική τιμή που ζητά η εκφώνηση.

```
C  
  
1  #include "exercise.h"  
2  
3  bool solve(int n) {  
4      return n % 2 == 0;  
5  }
```

Διάβασε τον κώδικα

Η δήλωση στο `exercise.h` περιγράφει την ίδια συνάρτηση που συμπληρώνεις στο αρχείο `.c`. Το `return` δίνει το αποτέλεσμα.

Πρόσεξε αυτό

Μην επιστρέφεις το ίδιο το υπόλοιπο. Το αποτέλεσμα πρέπει να είναι λογική τιμή.

Η ΛΥΣΗ

14 Απόσταση από το μηδέν

Η σκέψη

Αν $n < 0$, επιστρέφουμε $-n$. Αλλιώς επιστρέφουμε n . Το 0 ανήκει στη δεύτερη περίπτωση και μένει 0.

C

```
1  #include "exercise.h"
2
3  int solve(int n) {
4      if (n < 0) {
5          return -n;
6      } else {
7          return n;
8      }
9  }
```

Διάβασε τον κώδικα

Το `if` ελέγχει τη σύγκριση και επιλέγει ποιο από τα δύο `return` θα εκτελεστεί.

Πρόσεξε αυτό

Αν αλλάξεις το πρόσημο σε κάθε είσοδο, οι θετικοί αριθμοί θα γίνουν αρνητικοί. Τα tests ελέγχουν τη συμπεριφορά, ενώ εσύ ελέγχεις ότι χρησιμοποίησες τη ζητούμενη συνθήκη.

Η ΛΥΣΗ

15 Ο μεγαλύτερος από τους δύο

Η σκέψη

Αν $a > b$, επιστρέφουμε a . Αλλιώς επιστρέφουμε b . Στην ισότητα η δεύτερη επιλογή είναι σωστή, επειδή οι δύο τιμές είναι ίδιες.

C

```
1  #include "exercise.h"
2
3  int solve(int a, int b) {
4      if (a > b) {
5          return a;
6      } else {
7          return b;
8      }
9  }
```

Διάβασε τον κώδικα

Η δήλωση στο `exercise.h` περιγράφει την ίδια συνάρτηση που συμπληρώνεις στο αρχείο `.c`. Το `return` δίνει το αποτέλεσμα.

Πρόσεξε αυτό

Με αρνητικούς αριθμούς, το -2 είναι μεγαλύτερο από το -7 . Δεν συγκρίνουμε τις αποστάσεις τους από το μηδέν.

Η ΛΥΣΗ

16 Τρεις περιπτώσεις

Η σκέψη

Πρώτα ελέγχουμε $n < 0$, μετά $n == 0$ και τέλος επιστρέφουμε 1. Αν οι δύο πρώτοι έλεγχοι απέτυχαν, ο αριθμός είναι θετικός.

```
C
1  #include "exercise.h"
2
3  int solve(int n) {
4      if (n < 0) {
5          return -1;
6      } else if (n == 0) {
7          return 0;
8      } else {
9          return 1;
10     }
11 }
```

Διάβασε τον κώδικα

Η αλυσίδα `if / else if / else` ξεχωρίζει τις τρεις περιπτώσεις χωρίς επιπλέον υπολογισμό.

Πρόσεξε αυτό

Η απάντηση δεν είναι ο ίδιος ο αριθμός. Για είσοδο 42, ζητάμε 1.

Η ΛΥΣΗ

17 Μέσα στα όρια

Η σκέψη

Συνδυάζουμε $n \geq 0$ με $n \leq \text{high}$. Το αποτέλεσμα είναι αληθές μόνο όταν και οι δύο συγκρίσεις είναι αληθείς.

C

```
1  #include "exercise.h"
2
3  bool solve(int n, int high) {
4      return (n >= 0) && (n <= high);
5  }
```

Διάβασε τον κώδικα

Το λογικό «και» γράφεται `&&`. Κάθε πλευρά του είναι μια σύγκριση που δίνει λογική τιμή.

Πρόσεξε αυτό

Το λογικό «ή» θα δεχόταν αριθμούς έξω από τα όρια. Η ισότητα πρέπει να γίνεται δεκτή και στα δύο άκρα.

Η ΛΥΣΗ

18 Χωρίς υπόλοιπο

Η σκέψη

Υπολογίζουμε το υπόλοιπο του `n` με το `divisor` και ελέγχουμε αν είναι 0. Ο διαιρέτης δεν είναι ποτέ μηδέν, σύμφωνα με την εκφώνηση.

C

```
1  #include "exercise.h"
2
3  bool solve(int n, int divisor) {
4      return n % divisor == 0;
5  }
```

Διάβασε τον κώδικα

Η δήλωση στο `exercise.h` περιγράφει την ίδια συνάρτηση που συμπληρώνεις στο αρχείο `.c`. Το `return` δίνει το αποτέλεσμα.

Πρόσεξε αυτό

Το 2 δεν είναι πολλαπλάσιο του 5, αλλά το 0 είναι. Η συνάρτηση δεν πρέπει να έχει σταθερό διαιρέτη.

Η ΛΥΣΗ

19 Η πρώτη λέξη του FizzBuzz

Η σκέψη

Αν το υπόλοιπο με το 3 είναι 0, επιστρέφουμε "Fizz". Αλλιώς επιστρέφουμε "".
Ακόμη και το 15 δίνει μόνο Fizz σε αυτή την άσκηση, επειδή υπάρχει μόνο ο κανόνας του 3.

C

```
1  #include "exercise.h"
2
3  const char *solve(int n) {
4      if (n % 3 == 0) {
5          return "Fizz";
6      } else {
7          return "";
8      }
9  }
```

Διάβασε τον κώδικα

Η δήλωση στο `exercise.h` περιγράφει την ίδια συνάρτηση που συμπληρώνεις στο αρχείο `.c`. Το `return` δίνει το αποτέλεσμα.

Πρόσεξε αυτό

Το "" δεν έχει κενό χαρακτήρα. Το " " έχει έναν και θα αποτύχει στον έλεγχο.

Η ΛΥΣΗ

20 Οι κανόνες του FizzBuzz

Η σκέψη

Ελέγχουμε πρώτα τα δύο υπόλοιπα μαζί και επιστρέφουμε FizzBuzz. Έπειτα ελέγχουμε μόνο το 3, μετά μόνο το 5 και τέλος δίνουμε το κενό κείμενο. Το 30 ακολουθεί τον πρώτο κλάδο, το 9 τον δεύτερο, το 10 τον τρίτο και το 7 τον τελευταίο.

C

```
1  #include "exercise.h"
2
3  const char *solve(int n) {
4      if ((n % 3 == 0) && (n % 5 == 0)) {
5          return "FizzBuzz";
6      } else if (n % 3 == 0) {
7          return "Fizz";
8      } else if (n % 5 == 0) {
9          return "Buzz";
10     } else {
11         return "";
12     }
13 }
```

Διάβασε τον κώδικα

Κάθε κλάδος επιστρέφει ένα σταθερό κείμενο. Δεν χρειαζόμαστε μετατροπή αριθμού σε κείμενο ή δικό μας χώρο μνήμης.

Πρόσεξε αυτό

Αν ελέγξεις πρώτα μόνο το 3 και επιστρέψεις αμέσως Fizz, το 15 δεν θα φτάσει ποτέ στον κοινό έλεγχο. Το πλήρες FizzBuzz από το 1 έως το 100 θα προστεθεί αφού μάθουμε επανάληψη.

ΜΕΤΑ ΤΙΣ ΕΙΚΟΣΙ ΑΣΚΗΣΕΙΣ

Δοκίμασε τη σκέψη σου

Διάλεξε μια άσκηση που σε δυσκόλεψε. Πρόσθεσε ένα test με διαφορετική είσοδο και εξήγησε πρώτα την αναμενόμενη απάντηση. Έπειτα τρέξε την εντολή του εργαλείου και διάβασε το αποτέλεσμα.

Αν το test αποτύχει, έλεγξε αν το λάθος βρίσκεται στην προσπάθειά σου ή στην αναμενόμενη τιμή που έγραψες. Όταν περάσει, εξήγησε ποια περίπτωση έλεγξες που έλειπε πριν.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα