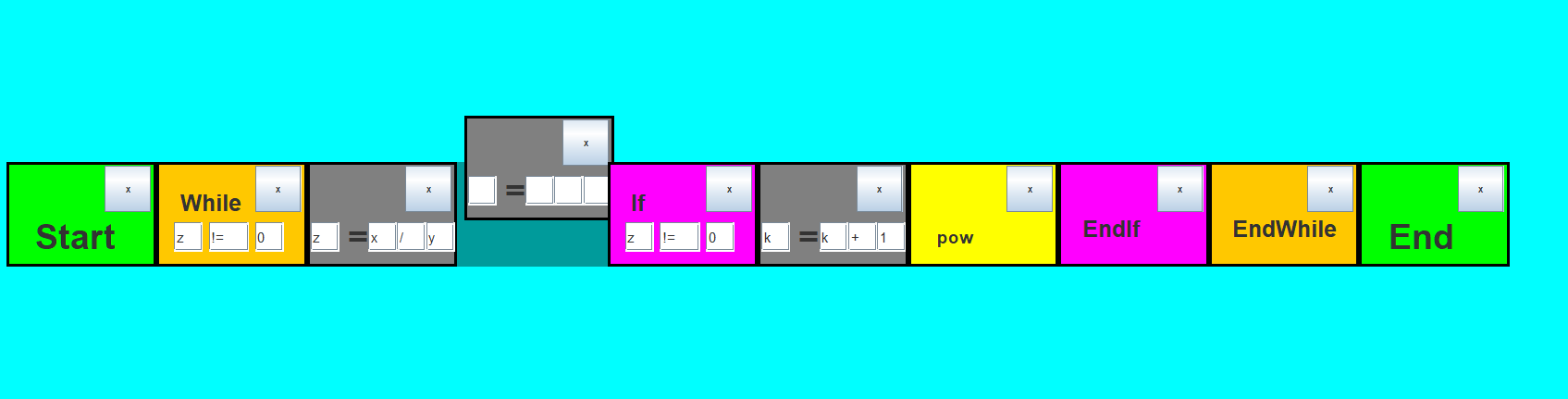
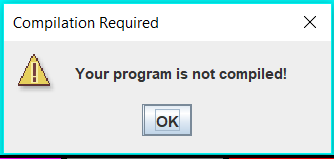
Test Cases

Παρακάτω φαίνεται η διαδικασία δημιουργίας ενός προγράμματος το οποίο μετράει τα ψηφία ενός αριθμού. Παρατίθενται κάποια λάθη τα οποία μπορεί να κάνει ο χρήστης και το πως αυτά φαίνονται στην εφαρμογή μας.

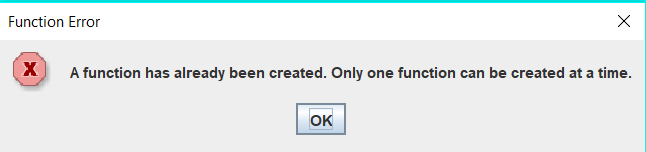
Αρχικά, φαίνεται η λειτουργία του snapping ενός μπλοκ το οποίο ο χρήστης θέλει να τοποθετήσει στο ενδιάμεσο μίας ήδη υπάρχουσας λίστας προγράμματος και το πως τα μπλοκ μετακινούνται προκειμένου να μπει στην σωστή θέση. Αντίστοιχα λειτουργεί και η περίπτωση που ο χρήστης βγάζει ένα ενδιάμεσο μπλοκ από την λίστα.



Έπειτα, δύο γενικότερα λάθη που μπορεί να κάνει ο χρήστης.



Ο χρήστης προσπαθεί να τρέξει το πρόγραμμα, ενώ έχει πραγματοποιήσει μία αλλαγή στη λίστα του προγράμματος, χωρίς να έχει πατήσει compile και εμφανίζεται μήνυμα λάθους το οποίο τον ενημερώνει πως δεν έχει κάνει compile.

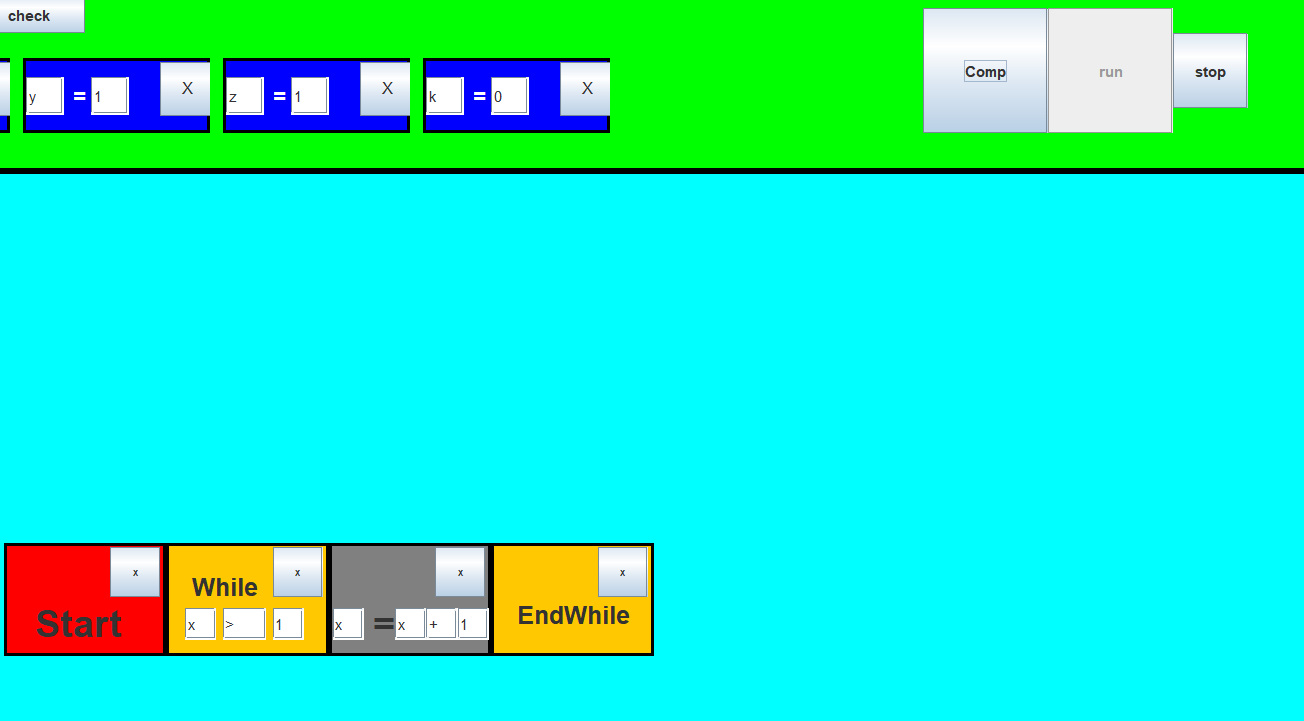


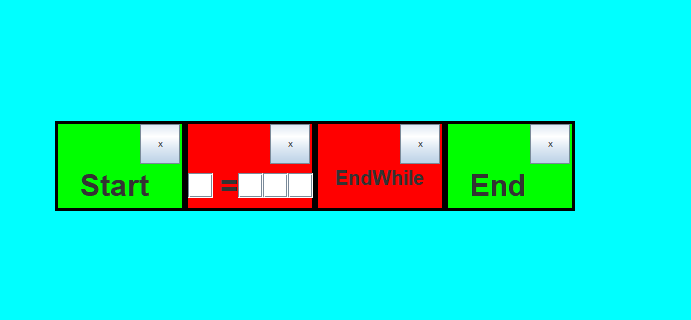
Στην περίπτωση που προσπαθήσει να φτιάξει δεύτερη ρουτίνα(function), εμφανίζεται μήνυμα λάθους το οποίο λέει πως υπάρχει ήδη μία ρουτίνα(function) και πως δεν μπορεί να υπάρχει άλλη.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, ορθογώνιο παραλληλόγραμμο, πολυχρωμία

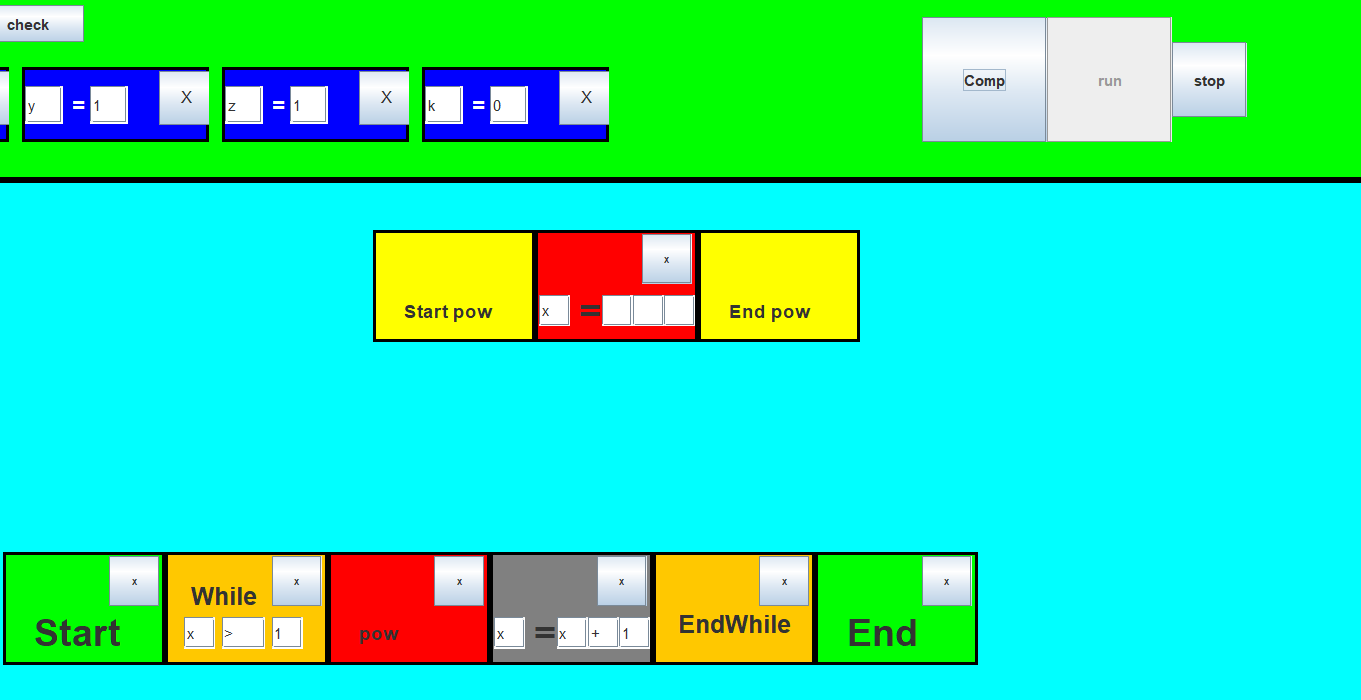
Το περιεχόμενο που δημιουργείται από τεχνολογία AI ενδέχεται να είναι εσφαλμένο.

Τα μπλοκ γίνονται κόκκινα εννοώντας πως υπάρχει κάποιο συντακτικό σφάλμα στον ορισμό της εκάστοτε μεταβλητής. Πιο συγκεκριμένα, δίνεται μη αποδεκτός αριθμός ή μη αποδεκτό όνομα μεταβλητής.

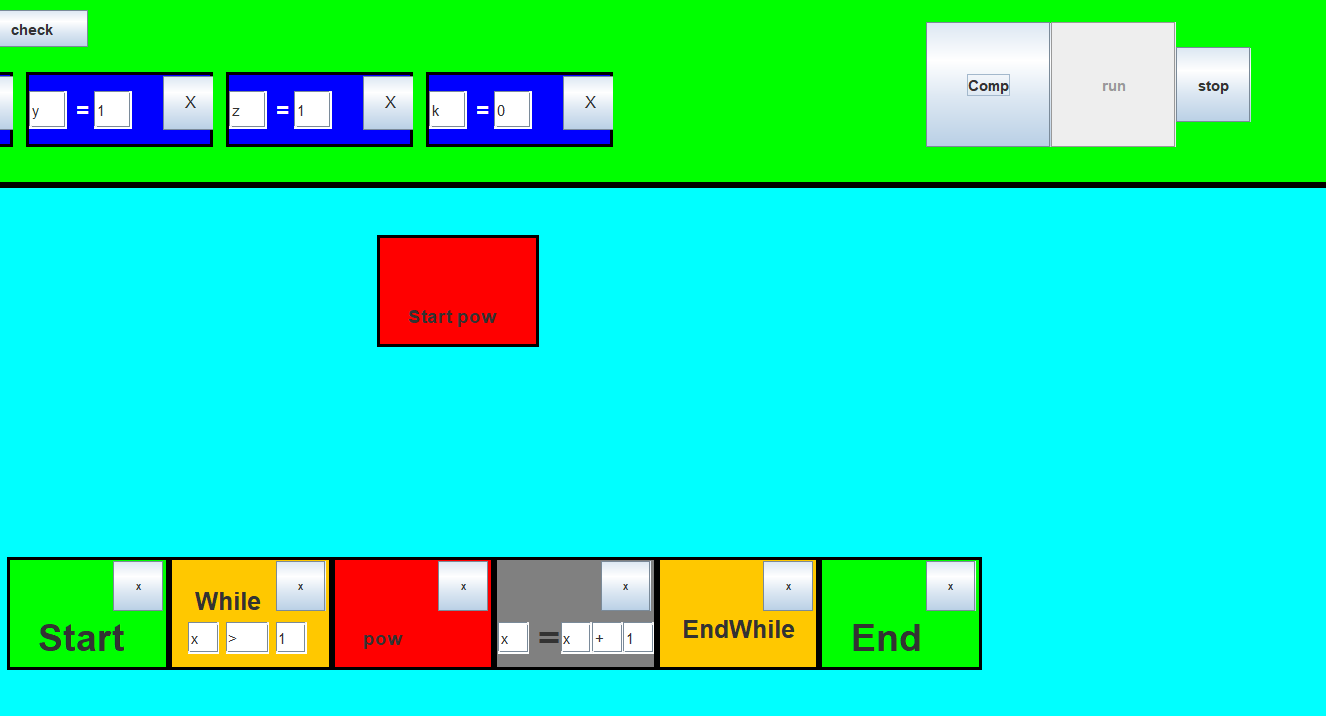




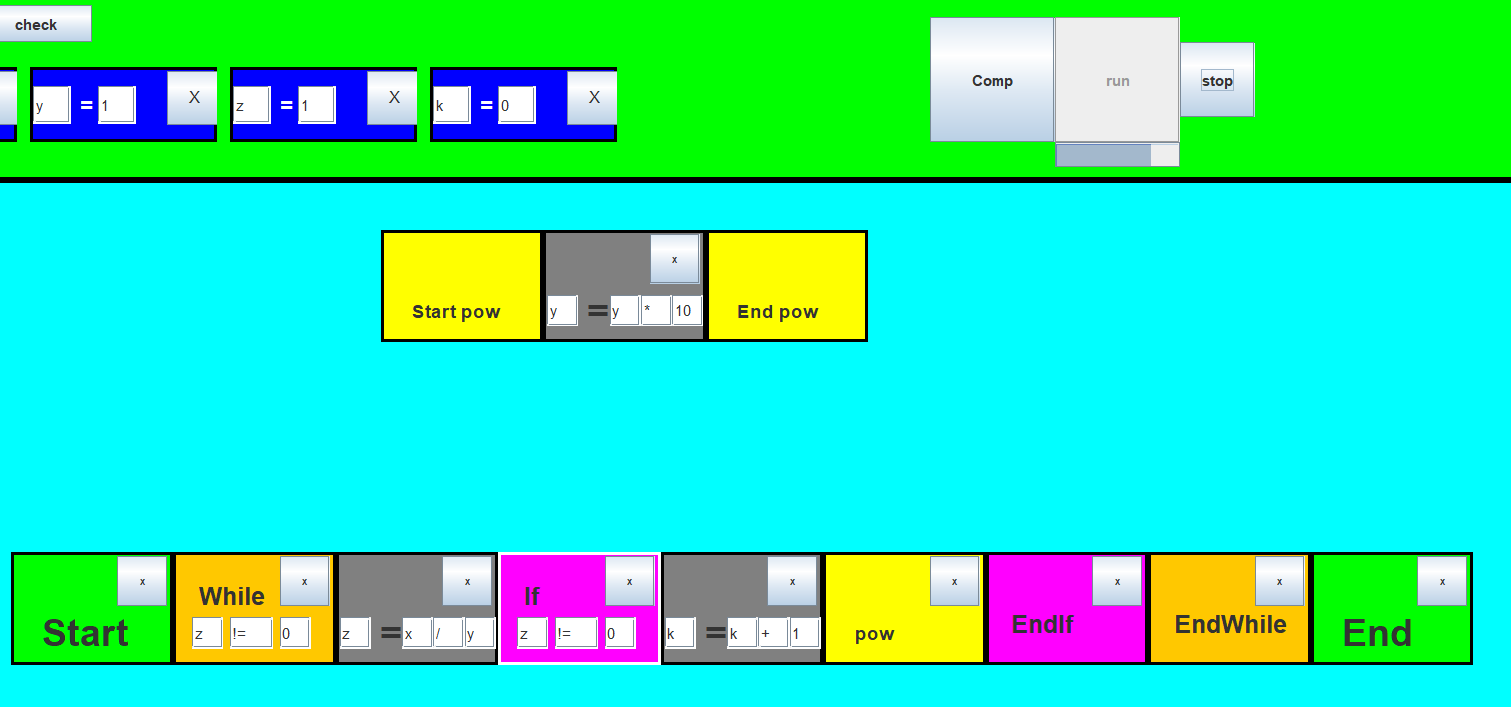
Όταν χρησιμοποιηθεί μία εντολή η οποία αποτελείται από δύο μέρη, το που αρχίζει και τελειώνει, χωρίς να έχει εισαχθεί στην λίστα προγράμματος το ένα από τα δύο μέρη της εντολής, το μέρος το οποίο υπάρχει στο πρόγραμμα γίνεται κόκκινο και απενεργοποιείται το run. Αυτό ισχύει για όλες της εντολές αυτού του τύπου.



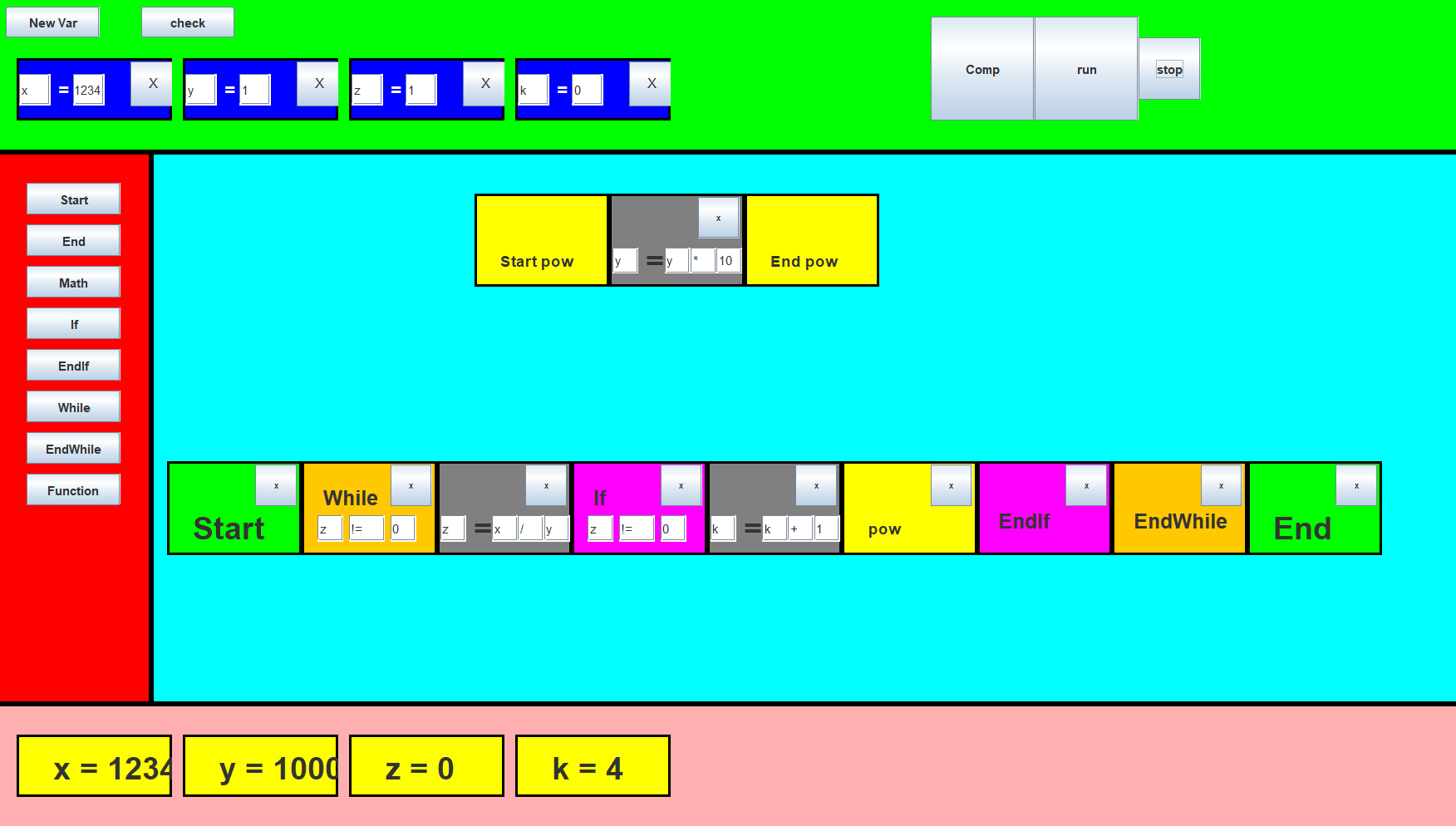
Αν υπάρχει λάθος μέσα στην λίστα του function τότε γίνεται κόκκινο και το μπλοκ που καλεί την function.



Συνδυαστικά τα δύο παραπάνω λάθη, η έλλειψη δηλαδή, του end function μπλοκ και το λάθος στην λίστα.



Παραπάνω φαίνεται η επιτυχημένη ολοκλήρωση του προγράμματος καθώς αυτό τρέχει. Η μπάρα κάτω από το run αντικατοπτρίζει την καθυστέρηση μέχρι την εκτέλεση της επόμενης εντολής και η εντολή που υλοποιείται την συγκεκριμένη χρονική στιγμή εμφανίζεται με το φωτεινό περίγραμμα.



Τέλος φαίνεται το ολοκληρωμένο πρόγραμμα μαζί με τα αποτελέσματά του.

Παρακάτω φαίνεται ο κώδικας στην κάθε επανάληψή του.

x = 1234; // Αρχικά x = 1234

y = 1; // Αρχικά y = 1

z = 1; // Αρχικά z = 1

k = 0; // Αρχικά k = 0

pow() { // Function: πολλαπλασιασμός του y επί το 10

y = y \* 10;

}

while( z != 0 ) { // Πρώτος έλεγχος : z = 1 ≠ 0 → μπαίνει στη λούπα

z = x / y; // Επανάληψη 1: z = 1234 / 1 = 1234

if( z != 0 ) {

k = k + 1; // k = 0 + 1 = 1

pow(); // y = 1 \* 10 = 10

}

z = x / y; // Επανάληψη 2: z = 1234 / 10 = 123

if( z != 0 ) {

k = k + 1; // k = 1 + 1 = 2

pow(); // y = 10 \* 10 = 100

}

z = x / y; // Επανάληψη 3: z = 1234 / 100 = 12

if( z != 0 ) {

k = k + 1; // k = 2 + 1 = 3

pow(); // y = 100 \* 10 = 1000

}

z = x / y; // Επανάληψη 4: z = 1234 / 1000 = 1

if( z != 0 ) {

k = k + 1; // k = 3 + 1 = 4

pow(); // y = 1000 \* 10 = 10000

}

z = x / y; // Επανάληψη 5: z = 1234 / 10000 = 0

if( z != 0 ) {

// δεν μπαίνει στο if —> z == 0

}

} // Τέλος λούπας — z = 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Iteration** | **y** | **z = x / y** | **z != 0?** | **k (if z ≠ 0)** | **pow() → y** |
| 1 | 1 | 1234 / 1 = 1234 | Yes | 1 | 10 |
| 2 | 10 | 1234 / 10 = 123 | Yes | 2 | 100 |
| 3 | 100 | 1234 / 100 = 12 | Yes | 3 | 1000 |
| 4 | 1000 | 1234 / 1000 = 1 | Yes | 4 | 10000 |
| 5 | 10000 | 1234 / 10000 = 0 | **No** | - (loop ends) | - |