

4^ο Εργαστήριο Τεχνικών Προγραμματισμού

Στόχοι

1. Εντοπισμός Λάθους (*error detection*) και Αναφορά (*Reporting*)
2. Οι συναρτήσεις ως παράμετροι συναρτήσεων
3. Συμβόλαια (*Contracts*) Συναρτήσεων

Ως βάση του εργαστηρίου σας δίδεται πρόγραμμα υλοποίησης στοίβας με Ολική Απόκρυψη, το οποίο θα επεκτείνετε με ελέγχους (στον φάκελο StackOA TS Struct + SetValue Parameter). Η δομή και η εν γένει υλοποίηση βασίζεται στα εργαστήρια 2+3. Η μόνη ουσιαστική διαφορά είναι η χρήση συναρτήσεων ως παράμετροι και η χρήση `assert`, που είναι το αντικείμενο του παρόντος 4^{ου} εργαστηρίου. Ο τύπος στοιχείου που διαχειρίζεται είναι `struct` με μέλη `int`, `char[cplithos]`.

Ακολουθήστε βήμα-βήμα τις οδηγίες

1. Τρέξτε το πρόγραμμα και ξεκινήστε με την επιλογή 2 (εισαγωγή στοιχείου). Θα πάρετε απροσδιόριστο `run-time error`. Ο λόγος βέβαια είναι ότι δεν έχετε δημιουργήσει την στοίβα, αλλά δεν είναι προφανής.
2. Σε όσες συναρτήσεις στο `stack.c` έχουν ως παράμετρο `StoivaPtr` προσθέστε την εντολή `assert(StoivaPtr!=NULL)`, ώστε να ελέγχουν την εγκυρότητα της παραμέτρου. Μερικές συναρτήσεις την περιέχουν ήδη.
3. Τρέξτε πάλι όπως το 1. Θα πάρετε `run-time error`, αλλά υπό έλεγχο. Το `assert` έδωσε `run-time error` αλλά **εντοπίστηκε** το λάθος άμεσα και δίνει **αναφορά**.
4. Το `assert` είναι η τελευταία γραμμή άμυνας. Το πρόγραμμα κανονικά δεν πρέπει να σταματήσει λόγω `assert`. Για αυτό προστατεύουμε τις κλήσεις με ελέγχους. Προσθέστε ελέγχους δημιουργίας της στοίβας στην επιλογή ώθηση (επιλογή 2), κατ' αναλογία με την επιλογή 3. Στην επιλογή 2 υπάρχουν οδηγίες σε σχόλια.
5. Τρέξτε πάλι το όπως το 1. Τώρα θα πρέπει να πάρετε το δικό σας μήνυμα λάθους και το πρόγραμμα δεν πρέπει να προχωράει στην είσοδο στοιχείων και πολύ περισσότερο να καλέσει την πράξη ώθηση.
6. Τρέξτε τώρα το πρόγραμμα κανονικά αρχίζοντας με την δημιουργία (επιλογή 1) και μετά ώθηση (επιλογή 2) όταν σας ζητήσει `int` δώστε είσοδο μη ακεραίο, π.χ «q». Κατόπιν με την επιλογή 3 εξωθήστε το στοιχείο. Τι παίρνετε; Παρατηρήστε ότι η συνάρτηση `TSstoiva_readValue` επιστρέφει συνθήκη λάθους. Δείτε τον ορισμό της και βάλτε κατάλληλους ελέγχους ώστε να μην κληθεί η ώθηση, αν το στοιχείο δεν είναι σωστό. Η συνάρτηση **εντοπίζει** το λάθος και το `main` το **αναφέρει**.
7. Τρέξτε πάλι το πρόγραμμα κανονικά αρχίζοντας με την δημιουργία (επιλογή 1) και ωθήστε 2-3 στοιχεία. Καλέστε τώρα πάλι την δημιουργία. Η νέα στοίβα είναι κενή. Τι έγινε η προηγούμενη στοίβα και τα προηγούμενα στοιχεία;
8. Προσθέστε έλεγχο στην επιλογή 1, ώστε να καλείται η συνάρτηση μόνο αν η `StoivaPtr` είναι κενή. Το μήνυμα λάθους πρέπει να δίνει κατάλληλες οδηγίες στον χρήστη, ώστε να μην επιτρέπεται σπατάλη μνήμης. Τι οδηγία δίνετε στον χρήστη;

9. Ελέγξτε ότι όλες οι επιλογές έχουν κατάλληλους ελέγχους.
10. Οι συναρτήσεις `othisi` & `exagogi` είναι τύπου `void` και χρησιμοποιούν μια σημαία (`flag`) παράμετρο ως ένδειξη λάθους, υπερχείλισης και υποχείλισης αντίστοιχα. Αλλάξτε τους ορισμούς (τουλάχιστον σε μια από αυτές) ώστε να επιστρέφουν την συνθήκη λάθους, αντί για παράμετρο. Να κάνετε και τις απαραίτητες αλλαγές στο `main`.
11. Τέλος ελέγξτε με `assert` αν η `othisi` κάνει όντως αυτό που υπόσχεται (`contract`). Προσθέστε στο τέλος της συνάρτησης μια εντολή `assert` που να ελέγχει αν το `korifi(πριν)+1==korifi(meta)`.
12. Για να δείτε αν δουλεύει κάνετε επίτηδες λάθος βάζοντας σε σχόλια την εντολή `korifi++`. Κατόπιν επαναφέρετε την εντολή.
13. Επίσης προσθέστε μια `assert` που να ελέγχει αν το στοιχείο εισόδου ισούται με το στοιχείο στις στοίβας στην θέση `korifi`. Χρησιμοποιήστε μια συνάρτηση `Equal` ως παράμετρο, κατά την `set_value`. Δείτε το πρότυπό της στον τύπο στοιχείου.
14. Για να δείτε αν δουλεύει κάνετε επίτηδες κάποιο λάθος στην `TSstoiva_setValue`. Κατόπιν επαναφέρετε την εντολή.
15. Τώρα η `othisi` ελέγχει τον εαυτό της βάσει του συμβολαίου της. Αν έχετε χρόνο διαμορφώστε αντίστοιχα την `exagogi`.
16. Επίσης δίνονται παραδείγματα και οδηγίες χρήσης `assert` και `errno` για δική σας μελέτη και αναφορά.