Assignment 4: "Sudoku checker, solver, and generator"

Σκοπός

Ο στόχος της άσκησης αυτής είναι να βοηθήσει στην εκμάθηση/κατανόηση (1) της αναδρομικής κλήσης και διαχείριση της στοίβας, (2) του περάσματος παραμέτρων by value και by reference, και (3) της χρήσης αναδρομής για ψάξιμο ενός συνόλου επιλογών.

Η άσκηση

Μέσα στο repository που κάνατε fork θα βρείτε κάτω από τον φαελο src το interface sudoku.h. Το interface χρησιμοποιεί έναν ADT grid.h (και αυτό βρίσκεται κάτω από τον φάκελο src). Η άσκηση σας ζητάει να:

1. Ορίσετε το grid.h. Μπορείτε είτε να ορίσετε το δικό σας grid.h είτε είτε να χρησιμοποιήσετε το grid.h που βρίσκεται μέσα στο repository.
2. Υλοποιήσετε το grid.h σε ένα αρχείο grid.c.
3. Υλοποιήσετε το sudoku.h σε ένα αρχείο sudoku.c.
4. Υλοποιήσετε το εκτελέσιμο sudoku που εκτελείται με τις εξής παραμέτρους:
    a. sudoku: διαβάζει από το stdin ένα puzzle και
        - τυπώνει το input puzzle στο stderr,
        - λύνει το puzzle,
        - τυπώνει στο stderr ένα μήνυμα που λέει αν το puzzle έχει λύση μοναδικής, περισσότερων επιλογών, ή καμία λύση, και
        - τυπώνει στο stdout μια λύση αν υπάρχει ή κάποιο από τα puzzles που εξέτασε ως πιθανή λύση με τα λάθη που αυτό παρουσιάζει.
    b. sudoku –c: διαβάζει από το stdin ένα puzzle και
        - τυπώνει το input puzzle στο stderr,
        - ελέγχει αν το puzzle είναι σωστό, και
        - τυπώνει στο stderr ένα μήνυμα που λέει αν το puzzle είναι σωστό ή όχι και αν όχι ποια είναι τα λάθη του.
    c. sudoku –g nelts –u: παράγει ένα νέο puzzle που έχει «περίπου» nelts μη κενά (μηδενικά) στοιχεία και
        - τυπώνει το puzzle στο stdout
        - Χωρίς την παράμετρο -u το puzzle που παράγεται *μπορεί να μην έχει* λύση με βήματα μοναδικής επιλογής
        - Με την παράμετρο -u το puzzle που παράγεται *έχει* λύση με βήματα μοναδικής επιλογής
5. Η συνάρτηση main θα βρίσκεται στο αρχείο sudoku.c
6. Ένα ενδεικτικό εκτελέσιμο sudoku βρίσκεται στο directory ~hy255/as4_sudoku/run. Π.χ.

```
run> sudoku -g 50 -u | sudoku | sudoku -c
New puzzle:
1 2 0 4 0 0 8 0 0
0 7 0 2 8 0 1 3 0
3 4 8 1 7 9 5 0 6
0 5 1 9 0 0 0 0 2
6 3 2 0 1 5 4 9 8
9 0 4 6 3 2 0 5 0
0 6 0 3 2 0 0 8 0
2 1 3 0 0 0 6 4 5
0 9 7 5 0 4 2 0 0
Puzzle has a (unique choice) solution:
```

```
1 2 6 4 5 3 8 7 9
5 7 9 2 8 6 1 3 4
3 4 8 1 7 9 5 2 6
7 5 1 9 4 8 3 6 2
6 3 2 7 1 5 4 9 8
9 8 4 6 3 2 7 5 1
4 6 5 3 2 1 9 8 7
2 1 3 8 9 7 6 4 5
8 9 7 5 6 4 2 1 3
Puzzle solution is correct.
run>
```

Μέσα στο repo υπάρχει ο φάκελος puzzle όπου περιέχει διάφορα test για να τρέξετε με το δικό σας εκτελέσιμο.

Sudoku

Το sudoku είναι ένα παιχνίδι με puzzles αριθμών. Σε κάθε puzzle δίνεται ένα grid 9x9 που αποτελείται από 3x3 υπο-grids το κάθε ένα με μέγεθος 3x3. Αρχικά το puzzle έχει ορισμένες από τις θέσεις του grid συμπληρωμένες με αριθμούς. Ο στόχος είναι να συμπληρώσει κανείς το υπόλοιπο puzzle με αριθμούς έτσι ώστε κάθε γραμμή, κάθε στήλη, και κάθε 3x3 grid να περιέχει όλα τα ψηφία από 1 ως 9. Περισσότερες πληροφορίες για το sudoku μπορείτε να βρείτε σε διάφορα sites του internet.

Interface grid.h

Το grid.h περιλαμβάνει τη δομή που χρησιμοποιείται για την αποθήκευση του grid και των επιπλέον δεδομένων που χρησιμοποιούνται κατά την επίλυση του grid. Το grid.h που δίνεται ορίζει ένα συγκεκριμένο τύπο για το Grid_T και ένα σύνολο από εξαιρετικά απλές συναρτήσεις που χρησιμοποιούνται για να προσπελαύνουν (διαβάζουν ή γράφουν) τα στοιχεία ενός αντικειμένου Grid_T.

Παρατηρήστε ότι συναρτήσεις που μόνο διαβάζουν δεδομένα από ένα αντικείμενο Grid_T παίρνουν ως παράμετρο ένα αντίγραφο του αντικειμένου, ενώ συναρτήσεις που μεταβάλουν ένα αντικείμενο Grid_T, επιστρέφουν ένα νέο αντίγραφο. Παρατηρείστε επίσης, ότι καμιά από τις συναρτήσεις δεν χρησιμοποιεί μεταβλητές τύπου pointer (εκτός από το απαραίτητο FILE * που μας επιβάλει η standard βιβλιοθήκη της C).

Interface sudoku.h

Το interface sudoku.h περιλαμβάνει τις συναρτήσεις που χρειάζεται κάποιο πρόγραμμα για να διαχειριστεί και να λύσει sudoku puzzles.

Το interface περιλαμβάνει τις βοηθητικές συναρτήσεις:

```
Grid_T sudoku_read(void);
void sudoku_print(FILE *s, Grid_T g);
int sudoku_is_correct(Grid_T g);
void sudoku_print_errors(Grid_T g);
```

Η συνάρτηση

```
Grid_T sudoku_solve(Grid_T g, Choice_T c);
```

είναι η κύρια συνάρτηση, παίρνει ως παράμετρο ένα puzzle (grid) το οποίο και δεν αλλάζει καθόλου και επιστρέφει τη λύση του puzzle. Αν το puzzle δεν έχει μοναδική λύση, τότε επιστρέφει μια από τις δυνατές λύσεις, ενώ αν δεν υπάρχει καμία λύση, επιστρέφει ένα συμπληρωμένο puzzle που εξέτασε ως πιθανή λύση. Η συνάρτηση sudoku_solve πρέπει να καλύπτει δύο περιπτώσεις, αν το puzzle έχει μοναδική λύση ή όχι. Η δεύτερη παράμετρος της sudoku_solve() έχει σκοπό να σας βοηθήσει κατά την αναδρομή να κάνετε iterate πάνω από όλα τα cells του

puzzle.

Τέλος η συνάρτηση

```
Grid_T sudoku_generate(int nelts, int unique);
```

έχει σαν στόχο να μπορεί ένα πρόγραμμα να δημιουργεί το ίδιο νέα puzzles.

Παρατηρείστε ότι καμιά από τις παραπάνω συναρτήσεις δεν χρησιμοποιεί μεταβλητές τύπου pointer (εκτός από το απαραίτητο FILE * που μας επιβάλει η standard βιβλιοθήκη της C).

Υλοποίηση

Για κάθε συνάρτηση που υλοποιείτε κάνετε ένα ξεχωριστό commit στο git repository σας.

Για να λύσουμε ένα puzzle μπορούμε να χρησιμοποιήσουμε τον εξής αλγόριθμο:

1. Για κάθε κενή θέση του grid σχηματίζουμε τις δυνατές επιλογές, που περιλαμβάνουν όλους τους αριθμούς οι οποίοι δεν επαναλαμβάνονται στην ίδια γραμμή, στήλη, ή υπο-grid.
2. Διαλέγουμε μια κενή θέση του grid και μια από τις δυνατές επιλογές για να τη συμπληρώσουμε.
3. Αφαιρούμε την επιλογή μας από το στοιχείο που διαλέξαμε και από το υπόλοιπο puzzle.
4. Υπάρχουν δύο περιπτώσεις:
    a. Αν το puzzle έχει σε κάθε βήμα ένα τουλάχιστον στοιχείο με μοναδική επιλογή, τότε η επιλογή μας ήταν «σίγουρη» και δε θα χρειαστεί να «δοκιμάσουμε» άλλες επιλογές για το συγκεκριμένο στοιχείο/cell. Σε αυτή την περίπτωση αρκεί απλά να κάνουμε iterate όλα τα στοιχεία του puzzle, χωρίς κάποια «αβεβαιότητα».
    b. (Αναδρομή) Αν το puzzle δεν έχει λύση με μοναδική επιλογή σε κάθε βήμα, ενδέχεται η επιλογή που κάνουμε σε κάποιο βήμα να μην οδηγεί σε λύση. Οπότε, πρέπει να την εφαρμόσουμε προσωρινά μόνο, ώστε αν τυχόν δεν οδηγεί σε λύση να δοκιμάσουμε την επόμενη δυνατή επιλογή. Αυτή η τεχνική ονομάζεται backtracking και μας εγγυάται ότι θα βρούμε τη λύση, αν υπάρχει. Για να υλοποιήσουμε το backtracking χρησιμοποιούμε ένα αντίγραφο (βοηθητικό) του grid, το οποίο "λύνουμε" αναδρομικά χρησιμοποιώντας την sudoku_solve(). Αν το βοηθητικό αυτό grid έχει λύση τότε το επιστρέφουμε ως αποτέλεσμα της τρέχουσας κλήσης της solve_grid() διαφορετικά το «πετάμε» και δοκιμάζουμε μια άλλη επιλογή.
5. Επαναλαμβάνουμε έως ότου το puzzle συμπληρωθεί σωστά.

Για το βήμα (1) μπορείτε να υλοποιήσετε μια συνάρτηση:

```
Grid_T grid_init(Grid_T g, int v[9][9]);
```

η οποία παίρνει σαν input ένα puzzle και το αλλάζει/αρχικοποιεί με βάση τις τιμές του array v. Η grid_init() πιθανώς θα είναι βολικό να χρησιμοποιεί την grid_update_value().

Για το βήμα (2) μπορείτε να υλοποιήσετε μια συνάρτηση:

```
Choice_T grid_iterate(Grid_T g, Choice_T t);
```

η οποία παίρνει ως input ένα puzzle και την βοηθητική παράμετρο t που περιγράφει την κατάσταση του iterator, εξετάζει τα cells του puzzle και τις επιλογές που απομένουν και επιστρέφει μια επιλογή ώστε να την δοκιμάσει η solve(). Είναι βολικό (και γρήγορο), αν υπάρχουν μια ή περισσότερες μοναδικές επιλογές, να επιστρέφει κάποια από αυτές (θα δείτε ότι ίσως σας βολεύει να μη διαλέγετε κάθε φορά την ίδια μοναδική επιλογή αλλά να χρησιμοποιήσετε τη συνάρτηση rand() για να εισάγετε κάποιο βαθμό τυχαιότητας). Διαφορετικά επιστρέφει κάποια

άλλη επιλογή (ποια είναι μια καλή επιλογή να επιστρέψει στην περίπτωση που δεν υπάρχει μοναδική επιλογή για κάποιο cell;).

Για το βήμα 3 μπορείτε να υλοποιήσετε τη συνάρτηση:

```
Grid_T grid_update(Grid_T g, Choice_T c);
```

η οποία παίρνει ως input ένα Grid_T αντικείμενο και το αλλάζει ώστε να εφαρμόζει στο cell c.i, c.j την επιλογή c.n και να την αφαιρεί από το υπόλοιπο g, επιστρέφοντας το νέο g.

Για την περίπτωση/βήμα (4a) αρκεί το πρόγραμμα να κάνει μια απλή επανάληψη πάνω από όλα τα στοιχεία του puzzle.

Για την περίπτωση/βήμα (4b) το backtracking υλοποιείται «αυτόματα» με αναδρομική κλήση της sudoku_solve(). Αν η sudoku_solve() μας επιστρέψει μια σωστή λύση, τότε την επιστρέφουμε, διαφορετικά «πετάμε» το αποτέλεσμα της sudoku_solve() και την καλούμε πάλι με μια νέα επιλογή από τον iterator.

Για την υλοποίηση της sudoku_generate(), μια προσέγγιση είναι να παράγετε αρχικά ένα σωστά συμπληρωμένο puzzle. Αυτό μπορείτε να το κάνετε με μια συνάρτηση:

```
Grid_T sudoku_generate(int nelts, int unique);
```

που κάνει δύο βήματα:

(a) ξεκινάει από ένα άδειο puzzle, παράγει όλες τις δυνατές επιλογές για κάθε θέση (sudoku_init()) και χρησιμοποιεί την sudoku_try_next() για να αντικαταστήσει ένα-ένα τα μηδενικά του αρχικού puzzle με σωστές τιμές (σε αυτό το σημείο αν η grid_iterate() επιστρέφει τυχαίες θέσεις/επιλογές σε κάθε εκτέλεση θα έχετε ένα διαφορετικό puzzle). Επειδή ωστόσο δεν υπάρχει κάποια εγγύηση ότι το puzzle που θα παραχθεί θα είναι σωστό, η sudoku_generate() μπορεί μετά από μερικές προσπάθειες (π.χ. 10 προσπάθειες) να χρησιμοποιεί κάποιο προ-συμπληρωμένο puzzle.

Μετά από την υλοποίηση αυτού του βήματος κάνετε ένα ξεχωριστό commit στο git repository σας.

(b) Στη συνέχεια, η sudoku_generate() προσπαθεί να αφαιρέσει από το συμπληρωμένο puzzle όλα εκτός από nelts στοιχεία. Κάθε φορά διαλέγει τυχαία μια θέση του puzzle και θέτει το στοιχείο στην τιμή 0. Αν πρέπει να παράγει puzzle με μοναδική επιλογή σε κάθε βήμα (-u) τότε καλεί την sudoku_solve() για να δει αν το puzzle έχει μοναδική επιλογή. Ανάλογα με την περίπτωση, κρατάει την τρέχουσα αλλαγή και συνεχίζει να αφαιρέσει το επόμενο στοιχείο ή την «ξεχνάει» και επιστρέφει στο προηγούμενο puzzle για να διαλέξει κάποιο άλλο στοιχείο προς αφαίρεση. Η διαδικασία επαναλαμβάνεται έως ότου έχουν μείνει στο puzzle όσα στοιχεία ορίζει η παράμετρος nelts (τι γίνεται αν η τυχαία επιλογή διαλέγει συνεχώς τις ίδιες θέσεις και επομένως κάθε βήμα δεν καταφέρνει να αφαιρέσει και ένα στοιχείο).

Σημείωση: Παρ’ότι εμείς χρησιμοποιούμαι ως ένδειξη δυσκολίας τον αριθμό των συμπληρωμένων αρχικών στοιχείων και την ύπαρξη μοναδικής επιλογής σε κάθε βήμα, αυτό δεν ισχύει απαραίτητα στην πράξη, μια και η δυσκολία έχει σχέση με το πως σκέφτεται ο κάθε παίκτης. Από την άλλη, η ύπαρξη backtracking είναι κάτι που φαίνεται να δυσκολεύει πάντα ανθρώπινους (σε αντίθεση με μηχανικούς) παίκτες.

Οι παρακάτω συναρτήσεις του grid.h θα σας βοηθήσουν να διακρίνεται και να διαχειριστείτε στην υλοποίηση του sudoku.h τις περιπτώσεις που υπάρχει λύση μοναδικής επιλογής ή όχι.

```
int grid_unique(Grid_T g);
Choice_T grid_exist_unique(Grid_T g);
Grid_T grid_clear_unique(Grid_T g);
```

Η συνάρτηση:

```
Choice_T grid_read_value(Grid_T g, Choice_T c);
```

σας επιτρέπει να διαβάζεται individual cells ενός puzzle (π.χ. θα σας χρειαστεί στην υλοποίηση της sudoku_print()).

Η υλοποίησή σας μπορεί να χρειαστεί επιπλέον, βοηθητικές συναρτήσεις είτε στην υλοποίηση του interface grid.h είτε στην υλοποίηση του sudoku.c, τις οποίες και μπορείτε να προσθέσετε στις αντίστοιχες υλοποιήσεις και να κρύψετε κατάλληλα.

Extra credit

- Σιγουρευτείτε ότι η υλοποίησή σας δεν χρησιμοποιεί καθόλου μεταβλητές τύπου pointer.
- Σιγουρευτείτε ότι η υλοποίησή σας δεν χρησιμοποιεί καθόλου global μεταβλητές.
- Ποιες συναρτήσεις της υλοποίησής σας εισάγουν κόστος χωρίς αυτό να είναι απαραίτητο; Εξηγήστε στο README.md αρχείο σας. Πως μπορείτε να μειώσετε το κόστος αυτών των συναρτήσεων;

Τυχαιότητα

Για να εισάγετε τυχαιότητα σε σημεία που πιθανόν χρειάζεται η υλοποίηση σας μπορείτε να χρησιμοποιήσετε τις συναρτήσεις rand() και srand() της stdlib (man rand, man srand). Για να αρχικοποιήσετε τις συναρτήσεις αυτές θα πρέπει να καλέσετε την srand με κάποια παράμετρο που αλλάζει σε κάθε κλήση του προγράμματος σας. Αυτό μπορεί να γίνει με την κλήση:

```
srand(getpid());
```

Η getpid() είναι μια συνάρτηση που επιστρέφει το process id του προγράμματος (και το οποίο είναι γενικά διαφορετικό από εκτέλεση σε εκτέλεση). Επειδή η getpid() ορίζεται στο <unistd.h> θα χρειστεί να χρησιμοποιήσετε το συγκεκριμένο interface.

Τέλος, για να αναθέσετε σε μια μεταβλητή r τυχαία τις τιμές π.χ. 0 έως N μπορείτε να χρησιμοποιήσετε:

```
r = rand()% N;
```

Test puzzles

Δίνονται διάφορα puzzles "s.*" μέσα στο repo μέσα στον φάκελο puzzles.

Makefile

Συμπληρώστε το Makefile που υπάρχει στο repo το οποίο θα πρέπει να περιέχει τουλάχιστον τα εξής targets:

- make clean: πρέπει να σβήνει όλα τα αρχεία που παράγονται κατά την μετάφραση του προγράμματος και να αφήνει μόνο τα header και source αρχεία του προγράμματος σας.
- make: πρέπει να παράγει το εκτελέσιμο sudoku.

Logistics

Βήμα 1:

- Κάντε fork το repository assignment4 από την ομάδα του μαθήματος στο csd gitlab. Στη συνέχεια αλλάξτε τα permissions σε private όπως αναγράφει εδώ. Προσθέστε ως members στο repo σας τους TAs του μαθήματος.

Βήμα 2:

- Γράψτε το πρόγραμμα σας στα συστήματα x86 του CSD χρησιμοποιώντας τα εργαλεία gcc, vim, emacs, gdb. Επεξεργαστείτε τα αρχεία (grid.c, grid.h, soduku.c, sudoku.h, Makefile) που βρίσκονται κάτω από το φάκελο src συμπληρώνοντας τον κώδικά σας μέσα.
- Περιορίστε το μέγεθος των γραμμών (πλάτος) στο αρχείο σας σε 78 ή 80 χαρακτήρες. Αυτό σας επιτρέπει να τυπώνετε σε δύο στήλες σε χαρτί και να έχετε ταυτόχρονα ανοιχτά παράθυρα για editing και compilation και execution.
- Για κάθε ζητούμενη λειτουργία όπως περιγράφεται πιο πάνω που υλοποιείτε κάνετε ένα ξεχωριστό commit στο git repository σας.

Βήμα 3: Preprocess, Compile, Assemble, and Link
Χρησιμοποιήστε τον gcc με τις command line παραμέτρους "-Wall, -ansi, -pedantic" για να κάνετε preprocess, compile, assemble, και link το πρόγραμμά σας.

Βήμα 4: README.md
Χρησιμοποιήστε τον αγαπημένο σας κειμενογράφο (emacs/vim/nano) για να επεξεργαστείτε το README.md text file που υπάρχει συμπληρώνοντας το:

- Το όνομά σας
- Πράγματα που χειρίζεστε με διαφορετικό τρόπο από ότι ορίζει η άσκηση.
- Μια περιγραφή της βοήθειας που είχατε από άλλους στη δημιουργία του προγράμματος σας, και σε συμφωνία με το “Policies” section του web page του μαθήματος.
- (Προαιρετικά) Μία ένδειξη του πόσο χρόνο αφιερώσατε για την άσκηση.
- (Προαιρετικά) Οτιδήποτε άλλο θέλετε να αναφέρετε.

Σχόλια που περιγράφουν τον κώδικά σας δεν πρέπει να υπάρχουν στο readme file. Πρέπει να τα ενσωματώσετε στο κατάλληλο σημείο του προγράμματος σας.

Βήμα 5: Υποβολή
Η παράδοση της άσκησής σας θα γίνει μέσω git, σύμφωνα με τις οδηγίες που περιγράφονται εδώ. Συγκεκριμένα το repository σας στο gitlab θα πρέπει να είναι fork του repository assignment4 και θα πρέπει να προσθέσετε ως members τους TAs του μαθήματος. Σιγουρευτείτε ότι ο κώδικάς σας έχει γίνει σωστά commit και ότι φαίνονται στο online repository στο account σας στο csd-gitlab.

Προσοχή! Μην κάνετε commit object και executables αρχεία.

Επειδή η εξέταση θα γίνει στα μηχανήματα του Τμήματος θα πρέπει να κάνετε clone το repository στα μηχανήματα του Τμήματος και να κάνετε compile και run τις ασκήσεις σας σε αυτά τα συστήματα. Αυτή είναι η ίδια διαδικασία που θα ακολουθηθεί την ημέρα της εξέτασης.

Στην προθεσμία της παράδοσης ένα script θα τρέξει και θα κατεβάσει όλα τα repositories που έχουν γίνει fork. Αυτά είναι τα repositories που θα βαθμολογηθούν.

Βαθμολογία
Η βαθμολογία θα βασιστεί και στην ορθότητα αλλά και στο σχεδιασμό, όπως αναφέρεται στη σελίδα Policies του μαθήματος. Για να ενθαρρύνεται η χρήση σωστών πρακτικών προγραμματισμού, προσπαθείτε να τηρείτε όσο το δυνατόν πιο κοντά τις οδηγίες της σχετικής σελίδας και επίσης τις αρχές που εξηγούνται στο μάθημα. Όπως πάντα, η κατανόηση της άσκησης αλλά και η αναγνωσιμότητα ενός προγράμματος είναι σημαντικό μέρος του σχεδιασμού.

Last Modified: 21-03-2023 09:39