Assignment 3: ADTs: Πίνακας Συμβόλων (Symbol Table)

Σκοπός
Ο σκοπός αυτής της άσκησης είναι να σας βοηθήσει να κάνετε επανάληψη/μάθετε (1) τη χρήση arrays και pointers στη γλώσσα προγραμματισμού C, (2) πως να δημιουργείτε modules χωρίς state στη C, και (3) τη χρήσης των εργαλείων προγραμματισμού GNU/UNIX, ειδικά gcc, shell, και editing (emacs).

Background
Ένας symbol table είναι μια συλλογή από ζεύγη δεσμεύσεων (bindings). Ένα ζεύγος δέσμευσης αποτελείται από ένα κλειδί (key) και μία τιμή (value). Το κλειδί είναι ένα string που χαρακτηρίζει με μοναδικό τρόπο το αντίστοιχο ζεύγος δέσμευσης (binding). Η τιμή είναι δεδομένα που κατά κάποιο τρόπο αντιστοιχούν στο συγκεκριμένο κλειδί. Ο symbol table επιτρέπει στον χρήστη να εισάγει (put) ένα νέα bindings, να διαβάσει (get) τις τιμές των bindings με βάση τα κλειδιά, και να σβήσει (remove) bindings με βάση τα κλειδιά. Οι symbol tables χρησιμοποιούνται συχνά σε συστήματα προγραμματισμού, όπως compilers, assemblers, και λειτουργικά συστήματα.
Υπάρχουν διάφοροι τρόποι να υλοποιηθεί ένας symbol table. Ένας απλός τρόπος είναι να αποθηκεύσουμε τα bindings σε μία διασυνδεδεμένη λίστα (linked list). Διασυνδεδεμένες λίστες περιγράφονται σε διάφορες αναφορές, π.χ. στο Section 17.5 του C Programming: A Modern Approach (King). Μια πιο αποτελεσματική υλοποίηση ενός symbol table, μπορεί να χρησιμοποιήσει έναν hash table. Οι hash tables περιγράφονται επίσης σε πολλές αναφορές, όπως στο Chapter 3 του "C Interfaces and Implementations (David R. Hanson)".

Η άσκηση
Η άσκηση σας ζητάει να δημιουργήσετε έναν abstract data type (ADT) που ονομάζεται SymTable. Κάθε αντίγραφο (instance) του SymTable ADT θα είναι ένας symbol table. Πρέπει να σχεδιάσετε τον SymTable ADT ώστε να είναι «γενικός», δηλαδή οι τιμές (values) να είναι void pointers, και επομένως να μπορούν να δείξουν σε δεδομένα οποιουδήποτε τύπου. Θα δημιουργήσετε δύο υλοποιήσεις του SymTable ADT: μία υλοποίηση που χρησιμοποιεί μια διασυνδεδεμένη λίστα και μια που χρησιμοποιεί ένα hash table. Ο SymTable ADT που θα δημιουργήσετε θα σας χρησιμεύσει και αργότερα.

Το Interface του SymTable ADT

- Το interface toy SymTable ADT θα πρέπει να βρίσκεται στο αρχείο symtable.h. Πρέπει να περιέχει τις δηλώσεις των παρακάτω συναρτήσεων:

```
SymTable_T SymTable_new(void);
void SymTable_free(SymTable_T oSymTable);
unsigned int SymTable_getLength(SymTable_T oSymTable);
   int SymTable_put(SymTable_T oSymTable, const char *pcKey, const void
*pvValue);
int SymTable_remove(SymTable_T oSymTable, const char *pcKey);
int SymTable_contains(SymTable_T oSymTable, const char *pcKey);
void *SymTable_get(SymTable_T oSymTable, const char *pcKey);
void SymTable_map(SymTable_T oSymTable, void (*pfApply)(const char *pcKey,
void *pvValue, void *pvExtra),const void *pvExtra);
```

- Η συνάρτηση SymTable_new θα επιστρέφει ένα καινούργιο SymTable_T που είναι άδειο (δεν περιέχει καθόλου bindings).
- Η συνάρτηση SymTable_free πρέπει να ελευθερώνει όλη τη μνήμη που χρησιμοποιείται από το oSymTable. Αν το oSymTable είναι NULL, τότε η συνάρτηση δεν πρέπει να κάνει τίποτε.
- Η συνάρτηση SymTable_getLength πρέπει να επιστρέφει τον αριθμό των bindings στο oSymTable. Είναι ελεγχόμενο λάθος χρόνου εκτέλεσης του προγράμματος (checked runtime error) το oSymTable να είναι NULL.
- Η συνάρτηση SymTable_put πρέπει, αν δεν υπάρχει binding με κλειδί pcKey στο oSymTable, να προσθέτει ένα καινούργιο binding στο oSymTable που θα αποτελείται από το κλειδί pcKey και την τιμή pvValue, και θα επιστρέφει 1 (TRUE). Διαφορετικά, η συνάρτηση δεν πρέπει να αλλάζει το oSymTable, και πρέπει να επιστρέφει 0 (FALSE). Είναι ελεγχόμενο λάθος χρόνου εκτέλεσης (checked runtime error) αν το oSymTable ή το pcKey είναι NULL.
- Η συνάρτηση SymTable_remove πρέπει, αν υπάρχει ένα binding με κλειδί pcKey στο oSymTable, να αφαιρεί το binding από το oSymTable και να επιστρέφει 1 (TRUE). Διαφορετικά, η συνάρτηση δεν πρέπει να αλλάζει το oSymTable, και πρέπει να επιστρέφει 0 (FALSE). Είναι ελεγχόμενο λάθος χρόνου εκτέλεσης (checked runtime error) αν το oSymTable ή το pcKey είναι NULL.
- Η συνάρτηση SymTable_contains πρέπει να επιστρέφει 1 (TRUE) αν το oSymTable περιέχει ένα binding του οποίου το κλειδί είναι pcKey, και 0 (FALSE) διαφορετικά. Είναι ελεγχόμενο λάθος χρόνου εκτέλεσης (checked runtime error) αν το oSymTable ή το pcKey είναι NULL.

- Η συνάρτηση SymTable_get πρέπει να επιστρέφει την τιμή του binding στο oSymTable του οποίου το κλειδί είναι pcKey, ή NULL, αν δεν υπάρχει τέτοιο binding. Είναι ελεγχόμενο λάθος χρόνου εκτέλεσης (checked runtime error) αν το oSymTable ή το pcKey είναι NULL.
- Η συνάρτηση SymTable_map πρέπει να εφαρμόζει τη συνάρτηση *pfApply σε κάθε binding στο oSymTable, περνώντας το pvExtra ως επιπλέον παράμετρο. Είναι ελεγχόμενο λάθος χρόνου εκτέλεσης (checked runtime error) αν το oSymTable ή το pfApply είναι NULL.
- Ο SymTable πρέπει να "κατέχει" τα κλειδιά του (keys). Δηλαδή, η συνάρτηση SymTable_put δεν πρέπει απλά να αποθηκεύει την τιμή του pcKey στο binding που δημιουργεί. Πρέπει να δημιουργεί ένα αντίγραφο (copy) του string pcKey, και να αποθηκεύει τη διεύθυνση του αντιγράφου αυτού στο νέο binding. Στη δημιουργία του αντιγράφου, θα σας βοηθήσουν οι standard C συναρτήσεις strlen και malloc. Αντίθετα ο SymTable δεν πρέπει να “κατέχει” τις τιμές του (values). Μάλιστα, δεν μπορεί να έχει τις δικές του τιμές, μια και δεν είναι δυνατό να καθορίσει το μέγεθος αυτών των τιμών (values) και επομένως δεν μπορεί να χρησιμοποιήσει αντίγραφα αυτών.
- Για κάθε λειτουργία function/λειτουργία που υλοποιείτε είναι καλή πρακτική να έχετε ένα ξεχωριστό commit στο git repository σας.

Η υλοποίηση του SymTable με διασυνδεδεμένη λίστα
Η υλοποίηση του SymTable με διασυνδεδεμένη λίστα (linked list) πρέπει:

- Να είναι αποθηκευμένη στο αρχείο με όνομα symtablelist.c.
- Να ελέγχει τα checked runtime errors με την χρήση του standard assert macro.
- Να μην περιέχει "memory leaks" (διαφυγή μνήμης). Ο SymTable ADT θα χρησιμοποιεί δυναμική εκχώρηση μνήμης (dynamic memory allocation).
- Επίσης, θα πρέπει να ελευθερώνει ρητά όλη την μνήμη που εκχωρήθηκε δυναμικά όταν αυτή η μνήμη δεν χρειάζεται πλέον. Για κάθε χρήση της συνάρτησης malloc (ή calloc) function στον ADT σας, τελικά θα πρέπει να υπάρχει ακριβώς μια κλήση της συνάρτησης free.

Η υλοποίηση του SymTable με Hash Table
Η υλοποίηση του SymTable με hash table πρέπει:

- Να είναι αποθηκευμένη στο αρχείο με όνομα symtablehash.c.
- Να περιέχει αρχικά 509 buckets.
- Να χρησιμοποιεί κάποια λογική hash function, π.χ.

```
#define HASH_MULTIPLIER 65599
...
/* Return a hash code for pcKey. */
static unsigned int SymTable_hash(const char *pcKey)
{
       size_t ui;
       unsigned int uiHash = 0U;
       for (ui = 0U; pcKey[ui] != '\0'; ui++)
              uiHash = uiHash * HASH_MULTIPLIER + pcKey[ui];
       return uiHash;
}
```

Φυσικά μπορείτε να χρησιμοποιήσετε και άλλες εναλλακτικές συναρτήσεις.

- Να ελέγχει τα checked runtime errors με την χρήση του standard assert macro.
- Να μην περιέχει "memory leaks" (διαφυγή μνήμης).

Extra Credits
Ο SymTable μπορεί να κάνει expand. Δηλαδή, για λόγους αποτελεσματικότητας, η υλοποίησή σας μπορεί να αυξάνει τον αριθμό των buckets (και, αναγκαστικά να επανατοποθετεί όλα τα bindings) όποτε ο αριθμός των bindings γίνεται πολύ μεγάλος. Μπορείτε να χρησιμοποιείτε τα εξής μεγέθη για τον αριθμό των buckets σε κάθε βήμα: 509, 1021, 2053, 4093, 8191, 16381, 32771, και 65521. Όταν η συνάρτηση SymTable_put ανιχνεύει ότι με το νέο binding ξεπερνιούνται τα 509 bindings, τότε πρέπει να αυξάνει τον αριθμό των buckets σε 1021. Όταν η συνάρτηση ανιχνεύει ότι με το νέο binding ξεπερνιούνται τα 1021 bindings, τότε θα αυξάνει τον αριθμό των buckets σε 2053, κ.ο.κ. Όταν ο SymTable_put ανιχνεύει ότι με το νέο binding ξεπερνιούνται τα 65521 bindings, τότε δεν θα αυξάνει τον αριθμό των buckets. Οπότε το 65521 είναι ο μέγιστος αριθμός buckets που μπορεί να περιέχει ο SymTable. Θα έχετε καλύτερο βαθμό αν παραδώσετε μια έκδοση του hash table που δεν υποστηρίζει επέκταση αλλά δουλεύει σωστά, από το να παραδώσετε μια έκδοση του hash table που υποστηρίζει επέκταση αλλά δεν δουλεύει σωστά. Αν προσπαθήσετε να υλοποιήσετε την επέκταση και δεν την τελειώσετε, τότε αφαιρέστε τον κώδικα σας για την επέκταση και περιγράψτε στο readme αρχείο την προσπάθειά σας.

Test Program

Δημιουργήστε ένα test πρόγραμμα testsymtab (testsymtab.c) που θα εκτελεί διάφορες πράξεις σε έναν ή περισσότερους symbol tables. Προσπαθήστε να καλύψετε όσο το δυνατόν περισσότερες περιπτώσεις στις δοκιμές σας.

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 list: πρέπει να κάνει build το executable testsymtab χρησιμοποιώντας την υλοποίηση του SymTable με την χρήση της linked list.
make hash: πρέπει να κάνει build το executable testsymtab χρησιμοποιώντας την υλοποίηση του SymTable με την χρήση του hash table.

Logistics

Βήμα 1:
Κάντε fork το repository assignment3 από την ομάδα του μαθήματος στο csd gitlab. Στη συνέχεια αλλάξτε τα permissions σε private όπως αναγράφει στα 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 (symtable.h, symtablelist.c, symtablehash.c, testsymtab.c, 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 συνάρτηση όπως περιγράφεται πιο πάνω είναι καλή πρακτική να 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 file
Προσθέστε στο README.md text file :

- 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: Submit
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 assignment3 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. Η κατανόηση της άσκησης αλλά και η αναγνωσιμότητα ενός προγράμματος είναι σημαντικό μέρος του σχεδιασμού.

Last Modified: 12-Feb-2021 08:23