OS Project 1

Βησσαρίων Μουτάφης sdi1800119@di.uoa.gr
1115201800119

Abstract

Η εφαρμογή βασίζεται σε 6 κύρια Modules, το **Hash Table**, το **List**, το **Inverted Index**, το **Student Manager**, το **CMD Argument Parser** και το **TTY API**. Το αυτοσχέδιο tty που παρέχουμε διαθέτει κάποια error messages, τις allowed mngstd εντολες αλλά και το help του tty, αλλα και τα allowed arguments τα οποία θα χρειαστούν για την δημιουργία του User Interface. Περαιτέρω έχουμε το Student Manager που φροντίζει να κρατάει record από τους μαθητές και να απαντάει στις ερωτήσεις. Συνδέεται με το tty με ένα κλασσικό enumeration based τρόπο και δύνανται να εκτυπώσει τιμές αλλά και μηνύματα λάθους στο stdout. Το Argument Parser αποτελείται από κάποιες συναρτήσεις που διευκολύνουν τον caller στο να κάνει parse και να χρησιμοποιήσει τις τιμές των arguments, παρέχοντας ταυτόχρονα ένα basic error handling για null argument values, καθώς και για εισαγωγή άγνωστων στο πρόγραμμα arguments.

Compilation and Run

Για το compilation και running του προγράμματος παρέχουμε ένα Makefile.

- **just compile; no run** : *make*
- **compile and run** : *make run*
- **re-compile and run**: *make re-run*
- **run with valgrind** : *make valgrind*
- **delete all objective and executable files** : *make clean*

Note for Running: Στο Makefile υπάρχει μία παράμετρος ***ARGS***, στην οποία **ο user μπορεί να βάλει τα δικά του command-line arguments**, ώστε να μην χρειάζεται να τρέχει όλη την εντολή εκτέλεσης και απλά να χρησιμοποιέι εξ ολοκλήρου το *make utility*.

NOTES για τους διορθωτές

- Για το input των 300,000 εισόδων δουλεύει απλά θα χρειαστεί να περιμένετε κοντά στο 1 λεπτό για να κάνει load το αρχείο.
- Παρατήρησα ένα θέμα στην έκδοση valgrind των PC της σχολής; όταν τερματίζει το εκάστοτε πρόγραμμα που τρέχει το valgrind, το τελευταίο αφήνει ένα-δύο memory blocks

που είναι unreachable. Στο δικό μου μηχάνημα δεν τα δείχνει (valgrind –version : *valgrind-3.15.0*).

- Για την generic υλοποίηση των δομών δεδομένων που ζητήθηκαν, ΟΛΕΣ οι δομές παίρνουν ως item ένα αντικείμενο $void^*$. Με αυτό τον τρόπο βελτιώσαμε και το *code repetition* πρόβλημα που υπάρχει στην χρήση ίδιων δομών με items διαφορετικών τύπων.
- Οι συναρτήσεις διαγραφής έχουν όλες υλοποιήθει με ένα destroy flag και ένα old pointer τα οποία δίνουν την δυνατότητα να ανακτήσουμε κάτι που θέλουμε να διαγράψουμε χωρίς επιπλοκές.
- Οι συναρτήσεις δημιουργίας έχουν όλες υλοποιήθει με ένα *deep_copy* flag για την καλύτερη γνώση του Pointer που μας επιστρέφεται, ώστε να αποφεύγονται memory corruptions από κάποια λάθος $free$.
- 'Ολες οι ορισμένες από τον χρήστη "βοηθητικές" δομές αλλά και συναρτήσεις για το print, hash, create, destroy έχουν υλοποιηθεί στο *struct_manipulation.c*
- Όλες οι δομές δεδομένων έχουν function pointers σε αντίστοιχες συναρτήσεις για το item destruction/hashing/comparison/printing.
- Όλες οι δομές δεδομένων που χρησιμοποιεί ο χρήστης (όπως List, Hash, etc.) είναι απλά ένα header για την καλύτερη επίτευξη του data abstraction.
- Το configuration file πρέπει να περιέχει στην πρώτη γραμμή το cuurent έτος και στην δέυτερη το bucket size που θέλουμε να έχουμε.
- Το generic parser ($parse\_line(line, *cols, separator)$) είναι υλοποιημένο βάσει της συνάρτησης *str.split* της python.
- Η $make\_srt(FILE^{**}\ stream)$ υλοποιήθηκε για μία ανάνγωση μίας ολόκληρης γραμμής του stream και για να το προχωράει. Υποθέτει πως θα του δωθεί ένα ανοικτό stream και δεν κάνει κανένα έλεγχο τέτοιου είδους. Προχωράει τον stream pointer και επιστρέφει length-of-the-string blocks μνήμης στον caller για να τα κάνει ότι θέλει.
- Δεν ευθύνομαι για κακή είσοδο από το config file, η οποία θα προκαλέσει υπερβολικά αργό search (π.χ. πολλοί μαθητές αλλά πολύ μικρό πλήθος buckets από το configuration file).

List Module

Η κλασσίκη διπλά συνδεδεμένη λίστα, υλοποιημένη με generic item ως $void*$, αφήνοντας στον χρήστη να δεσμεύει την μνήμη και να εισάγει κάθε τύπο αντικειμένου.

Παρέχει συναρτήσεις εισαγωγής στο τέλος ή την αρχή της (bool last), sorted εισαγωγής με βάση δοσμένης compare function, διαγραφής, αναζήτησης, επιστροφής μήκους, access σε βασικά nodes (head, tail). Ακόμη επιτρέπεται η ευρεία χρήση συναρτήσεων που κάνουν traverse ή/και modify τα List Nodes με την ευθύνη πάντα του χρήστη από το πως θα τα χρησιμοποιήσει.

Επίσης υπάρχουν και κάποιες ειδικές συναρτήσεις (αναγράφονται στο *List.h*), οι οποίες έχουν αποκλειστική χρήση την διευκόλυνση της χρησιμοποίησης λιστών στην υλοποίηση των ερωτημάτων της εφαρμογής.

Hash Table Module

Το **Hash Table** υλοποιήθηκε με generic τροπο αφήνοντας στον χρήστη να δεσμεύει την μνήμη και να εισάγει κάθε τύπο αντικειμένου, χρησει $void*$, το οποίο έχει γίνει typedef σε $Pointer$.

Το πλήθος των buckets έχουν καθοριστεί είτε βάσει του configuration file, είτε μέσω μιας εισαγωγής των $max\_entries$ και ισούται με $\frac{max\_entries \times 365}{22}$, βασισμένοι στο παράδοξο του $Misse$ (23 άτομα έχουν πιθανότητα πάνω απο 50% να έχουν γενέθλια την ίδια μέρα).

Η υλοποίηση του hash table έχει γίνει με την χρήση του module $List$, παρέχοντας ένα array από Lists ($List^*array$), το οποίο όπως δηλώσαμε έχει στατικό μέγεθος που δεν αλλάζει κατά το life span του Objec HT. Παρέχουμε Constructor, συν/ση εισαγωγής, συν/ση διαγραφής, συν/ση αναζήτησης, και destructor του hash table ώστε ο χρήστης να μην ασχολείται καθοόλου με τα σχετικά.

- Η contains κάνει την πολύ απλή αναζήτηση με βάση το hash και το compare value.
- Η insert, κάνει τα τυπικά: ψάχνει το item και αν δεν υπάρχει το εισάγει, αν υπάρχει το κάνει update (για αυτό αν δεν θελω την αλλαγή καλώ πρώτα την contains).
- Η delete ψάχνει το item και αν υπάρχει το διαγράφει.

Student Manager Module/API

Η βασική δομή της εφαρμογής. Αποτελείται από το Hash Table των μαθητών και από το Inverted Index και μία δομή που κρατάει το πλήθος των μαθητών σε ένα zip code.

- $mngstd\_create(compare, destroy, hash, filename)$: Constructor της δομής παίρνει ως είσοδο ένα αρχείο ή NULL καθώς και τις απαραίτητες συναρτήσεις για το manipulation των Student structs.
- $mngstd\_run(manager, expr, value)$: βασισμένη σε ένα numerical expression index το οποίο υπάρχει στο αντίστοιχο array βρίσκουμε ποιά εντολή μας είπε το tty να εκτελέσουμε και με βάση αυτό και το value που δώσαμε πράττουμε αναλόγως στο manager struct. Η συνάρτηση υλοποιήθηκε με switch statement για καθαρότερη δομή (είναι αρκετά μεγάλη).
- $mngstd\_destroy(manager)$: απελευθερώνει όλη την μνήμη που καταλαμβάνει η δομή σταδιακά και με αρκετά abstract τρόπο ώστε να μην μπλέκει ο user με την μνημη στο hash ή στο list ή στο Inverted Index etc.

Επίσης υπάρχει και η boolean τιμή is_end, η οποία ενημερώνει τον caller οτι δώθηκε σήμα εξόδου.

Inverted Index Module

Η βασική δομή της οποίας είσοδος είναι object τύπου $Index$. Τα object περιέχουν ένα πεδίο τύπου List για τους μαθητες και ένα πεδίο για τα χρονιά που βρίσκονται στην σχολή. Βασιζόμαστε στο Current year. Το Inverted Index αποτελείται από μία λίστα από Indexed με την βοήθεια του τύπου List.

- *invidx_create*(...): Constructor του Inverted Index. Τυπικές αναθέσεις.
- *invidx_to_list*(*inv_index*): Επιστρέφει την λίστα με τα indexes από το header του Inverted Index.
- *invidx_insert*(..., *student*): Εισάγει το Student στο Inverted Index. Άν δεν υπάρχει το κατάλληλο Year Index, το δημιουργεί και τον εισάγει, αν υπάρχει τότε το βρίσκει και τον εισάγει εκεί.
- *invidx_delete*(...): Διαγράφει το student αν υπάρχει στο αντίστοιχο Index.
- *invidx_destroy*(...): Απελευθερώνει την μνήμη που είχε δεσμετεί για το Inverted Index σταδιακά.

Custom Command Window API

Το custom TTY API που δημιούργησα έχει την δυνατότητα να εκτυπώνει τα σωστά error messages και τα αντίστοιχα confirmations μέσω της συνάρτησης *mngstd_run*(...). Το TTY προσφέρει ένα parser το οποίο σπάει το expression σε command και value και, αφού το ελέγξει και βρεί το αντίστοιχο expression index του, περνάει, βάσει ένος πίνακα, το expression index και το value στην παραπάνω συνάρτηση για να ενεργήσει. Το input το παίρνει μέσω της *make_string* η οποία είναι αυτοσχέδια συνάρτηση για να διαβάζει μία γραμμή από τον file pointer που της περνιέται και να γυρνάει ένα malloced char* string. Η βασική λούπα του προγράμματος βρίσκεται στο file *main.c*, η οποία παίρνει, κάνει parse και ελέγχει το input expression και μετά καλεί την *mngstd_run*(...) αν το input command είναι valid, αλλίως εκτυπώνει error message.

Argument Parser Functions

Το Argument Parser αποτελείται επί της ουσίας από μία συνάρτηση (*args_parser*). Η τελευταία φροντίζει να διαβάζει τα Inputed Command Line Arguments τα οποία της έχει δώσει η main συνάρτηση του προγράμματος και με βάση ένα array από allowed arguments να μπορεί να κρίνει αν το τρέχον read argument είναι σωστό και αν δεν είναι να τυπώνει το αντίστοιχο error message (help section for the tty arguments proper usage). Επιστρέφει σε ένα απο τα ορίσματά της (*char*** *input*), ένα πίνακα από τις τιμές των arguments στο κατάλληλο index έτσι ώστε ο χρήστης να χρησιμοποιήσει τα values χωρίς κανένα πρόβλημα και χωρίς κανένα μπέρδεμα.

Παράλληλα παρέχουμε και μία αρκετά εξειδικευμένη συνάρτηση για το σωστό parsing configuration file, την $parse\_cnfg(char^* \ filename)$, η οποία ακολουθεί την λογική του configuration file format που διευκρινήσαμε στο *Notes Section*.