HY255 Εργαστήριο Λογισμικού - L09

(a) Abstract Data Types (ADTs)
(b) Assignment 3 – Symbol Table

HY255 Εργαστήριο Λογισμικού
Άνοιξη 2021
Άγγελος Μπίλας

Abstract Data Types (ATDs) - Αφηρημένοι Τύποι Δεδομένων

- Τα modules έχουν ένα σημαντικό περιορισμό
  - Π.χ. στο παράδειγμά μας, τι γίνεται αν ένα πρόγραμμα θέλει να χρησιμοποιήσει 2 αντίγραφα μιας στοίβας για διαφορετικούς σκοπούς;
  - Το stack module μας ορίζει και χρησιμοποιεί πάντα ένα stack και το ίδιο κομμάτι μνήμης
- Ένα module που μας επιτρέπει να χρησιμοποιούμε πολλαπλά, διαφορετικά αντίγραφα του service που ορίζει, π.χ. πολλές στοίβες, ονομάζεται abstract data type
- Είναι ουσιαστικά σαν να δημιουργούμε ένα νέο τύπο που δεν υπάρχει στη γλώσσα, π.χ. στοίβα, και να μπορεί κανείς να δηλώνει αντικείμενα/μεταβλητές/ονόματα αυτού του νέου τύπου και να τα χρησιμοποιούμε μέσω των πράξεων (operations) του ADT
- Τι χρειάζεται για να μετατρέψουμε το stack module μας σε ADT;
  - 1. Να δηλώσουμε στο interface τον τύπο που θα έχουν οι μεταβλητές τύπου stack που θα ορίζει το κάθε πρόγραμμα
    - Οπότε με κάποιο τρόπο η δήλωση π.χ δύο stacks s1,s2 από ένα πρόγραμμα θα είναι κάτι σαν

      struct stack_s s1;

      struct stack_2 s2;
  - 2. Να αλλάξουμε αυτό το interface ώστε κάθε operation να μπορεί να αναφέρεται πλέον σε ένα συγκεκριμένο από τα πολλά stacks
- Ας δούμε πως φαίνεται το νέο interface σε αυτό το σημείο

Stack ADT interface: stack.h

- stack.h

```
#define STACK_SZ 100
struct stack_s {
    int contents[STACK_SZ];
    int top;
};
void make_empty(struct stack_s *s);
int is_empty(struct stack_s *s);
void push(struct stack_s *s, int i);
int pop(struct stack_s *s);
```

- Παρατηρούμε ότι πλέον έχουμε αποκαλύψει στον χρήστη πως υλοποιείται η στοίβα (ως ένα array και μια μεταβλητή στην κορυφή της)
- Χρειάζεται απαραίτητα να το κάνουμε αυτό;
- Εμείς θα θέλαμε ιδανικά ο χρήστης να γνωρίζει το όνομα του τύπου (απαραίτητο) χωρίς να γνωρίζει την υλοποίησή του
- Ένας τέτοιος τύπος ονομάζεται αδιαφανής (opaque)
- Αυτό είναι μια καινούργια δυνατότητα που δεν την έχουμε μέχρι τώρα
- Η C μας δίνει την δυνατότητα να ορίζουμε opaque types με την χρήση ενός νέου keyword: typedef
  - Σημείωση: Παρότι πολλές φορές νομίζουμε ότι το typedef χρησιμοποιείται για λόγους συντακτικού, να ορίσουμε νέα ονόματα για τύπους, η κύρια και ουσιαστική χρήση του είναι η δήλωση opaque τύπων

Νέο stack ADT interface: stack.h

- stack.h

```
typedef struct stack_s Stack_T;
void make_empty(Stack_T *s);
int is_empty(Stack_T *s);
void push(Stack_T *s, int i);
int pop(Stack_T *s);
```

- Το Stack_T είναι πλέον ένας νέος τύπος για μεταβλητές που ονομάζουν μια στοίβα
- Παρατηρούμε ότι πλέον ο χρήστης μπορεί να δηλώνει μεταβλητές τύπου Stack_T χωρίς να γνωρίζει την υλοποίηση του τύπου της στοίβας, όπως φαίνεται στο απλό πρόγραμμα δεξιά
- Ποια είναι η υλοποίηση του ADT της στοίβας;

main.c

```
#include "stack.h"
int main(void){
   int t;
   Stack_T *s1;
   Stack_T *s2;

   make_empty(s1);
   make_empty(s2);
   push(s1, 1);
   push(s2, 2);
   t=pop(s1);
   if (is_empty(s2))
     push(s1, 1);

   return;
}
```

Υλοποίηση stack ADT: stack.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "stack.h"

#define STACK_SZ 100
struct stack_s {
  int contents[STACK_SZ];
  int top;
} Stack_T;

void make_empty(Stack_T **s){
  *s = malloc(sizeof(Stack_T));
  if (!(*s)) {
    printf("error – out of memory\n"));
    exit(EXIT_FAILURE);
  }
  (*s)->top = 0;
}

int is_empty(Stack_T *s){
  return (s->top == 0);
}
```

```c
static int is_full(Stack_T *s){
  return (s->top == STACK_SZ);
}
void push(Stack_T *s, int i){
  if (is_full(s)){
    printf("error – stack full\n");
    exit(EXIT_FAILURE);
  }
  s->contents[top++] = i;
  return;
}

int pop (Stack_T *s){
  if (is_empty(s)){
    printf("error – stack empty\n");
    exit(EXIT_FAILURE);
  }
  return (s->contents[--top];
}

void destroy(Stack_T **s){
  assert(*s);
  free(*s);
  *s = NULL;
  return;
}
```

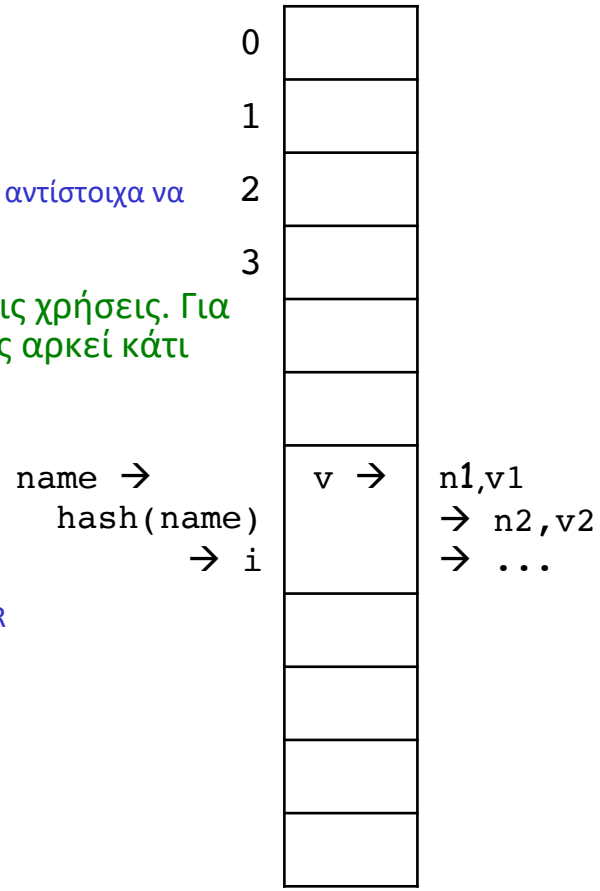
Assignment 3: Symbol Table ADT

- Στόχος της άσκησης είναι η κατανόηση των ADTs, interfaces, δυναμικής μνήμης και συμβολαίων
- Ένας symbol table (ST) είναι μια δομή που αποθηκεύει ζεύγη (key, value) και επιτρέπει τις βασικές λειτουργίες: insert, lookup, delete
  - Για μας τα ζεύγη θα είναι τύπου (string, pointer)
  - Σε μερικές περιπτώσεις είναι ιδιαίτερα σημαντικά επίσης τα update, scan
  - Οι symbol tables είναι μιας μορφής in-memory key-value store, που σήμερα χρησιμοποιούνται πάρα πολύ για την αποθήκευση δεδομένων σε πάρα πολλά συστήματα και εφαρμογές
- Θα υλοποιήσουμε ένα ST ADT με δύο τρόπους: λίστες και hash tables
  - Για την υλοποίηση με τον hash table, δεν μας ενδιαφέρει τόσο ο συγκεκριμένος hash table – αυτά τα συζητάτε στο μάθημα των δομών
  - Symbol tables, με την μορφή των hash tables, χρησιμοποιούνται σε πολλές περιπτώσεις εκτενώς στην διαδικασία της μετάφρασης (compiler, assembler, cpp) για να κρατούν την αντιστοίχιση (ονόματος, διεύθυνσης) για κάθε μεταβλητή, σε λειτουργικά συστήματα, storage systems, και πολλές πολλές άλλες περιπτώσεις
- Γιατί είναι χρήσιμοι οι hash tables σε όλες αυτές τις περιπτώσεις;
  - Μας επιτρέπουν να βρίσκουμε στοιχεία γρήγορα
  - Δουλεύουν καλά σε πολλές περιπτώσεις
    - Αλλά και σε πολλές άλλες δεν μας εξυπηρετούν π.χ. για γρήγορα scans
- Ποιος είναι ο ταχύτερος τρόπος στην γλώσσα για να αποθηκεύσουμε και να διαβάσουμε μια τιμή με βάση κάποιο index;
  - Τα arrays
  - Το index είναι ένας integer και το store/lookup είναι ένα operation στην μνήμη
- Πρόβλημα 1: Το index σε arrays της C είναι ένας integer ενώ εμείς θέλουμε να είναι string
  - Λύση: ας μετατρέψουμε το string σε integer με μια συνάρτηση: int hash(char *)

Sketch of table lookup

- Τι θα συμβεί αν δύο names αντιστοιχιστούν στην ίδια θέση του array από την συνάρτηση hash?
  - Μπορεί να συμβεί αυτό;
  - Φυσικά, αφού ο αριθμός των δυνατών ονομάτων είναι πολύ μεγαλύτερος από τον αριθμό των θέσεων του array
- Ας βάλουμε όλα τα names που θα "πέσουν" στην ίδια θέση του array σε μια λίστα
  - Οπότε πλέον για να βρούμε ένα στοιχείο πρέπει να πάμε στην αντίστοιχη θέση του πίνακα και μετά να διατρέξουμε την λίστα
- Ελπίζουμε ότι στην περίπτωση που το KV store περιέχει τόσα στοιχεία όσα περίπου είναι οι θέσεις του, τότε η hash function θα τα μοιράσει περίπου ισοδύναμα, και άρα οι πιο πολλές θέσεις θα έχουν ένα στοιχείο και επομένως δεν θα χρειαστεί να διατρέξουμε κάποια λίστα
- Όταν ο αριθμός των στοιχείων μεγαλώνει σε σχέση με το μέγεθος του array είναι αναπόφευκτο ότι σε αυτή την περίπτωση κάποιες λίστες θα αρχίσουν να μεγαλώνουν
  - Για να λύσουμε αυτό το θέμα μπορούμε να μεγαλώσουμε το μέγεθος του array
    - allocate ένα μεγαλύτερο array
    - insert όλα τα στοιχεία του παλιού array στο νέο
    - free το παλιό array
  - Όταν το array είναι αρκετά μεγαλύτερο από τον αριθμό των στοιχείων που περιέχει, μπορούμε αντίστοιχα να αντικαταστήσουμε με κάποιο πιο μικρό array
  - Οι λειτουργίες αυτές είναι extra credit στην Άσκηση
- Ποια είναι μια καλή hash function; Δεν υπάρχει "ιδανική" hash function για όλες τις χρήσεις. Για διαφορετικές περιπτώσεις χρησιμοποιούνται διαφορετικές hash functions. Για μας αρκεί κάτι σαν:

```
unsigned int hash (char *s) {
  unsigned int h = 0;
  unsigned char *p;
  for (p=s; *p != '\0'; p++) {
                h = MULTIPLIER * h + *p;      # empirically 31 is a good value
  }                                                          # for the MULTIPLIER
  return (h % HTSIZE);
}
```



Reading - Εργασία

- King Ch. 19.{3,4,5}
- Διαβάστε την εκφώνηση για την Άσκηση 3
- Υλοποιήστε την Άσκηση 3