ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

Δομές Δεδομένων (Εργ.)
Ακ. Έτος 2018-19
Διδάσκων: Ευάγγελος Σπύρου

Εργαστήριο 1 – Δομές στη C και Στοίβα (υλοποίηση με πίνακα)

1. Στόχος του εργαστηρίου

Στόχος του τέταρτου εργαστηρίου είναι η εξοικοίωση με τις δομές (structs) της γλώσσας C, εξοικείωση με τη δομή της στοίβας και μια πρώτη υλοποίηση σε C με χρήση πινάκων.

2. Δομές στη C

Η δομή είναι μια συλλογή πεδίων που χρησιμοποιούνται για την ομαδοποίηση πληροφορίας που περιγράφει μια λογική οντότητα. Για παράδειγμα μια δομή μπορέι να περιέχει πληροφορίες για μια εταιρείας όπως την επωνυμία της, το αντικείμενο εργασιών της, το έτος ίδρυσης, ΑΦΜ, αριθμό υπαλλήλων, στοιχεία επικοινωνίας... και άλλα δεδομένα.

Η δήλωση μιας δομής στη C αρχίζει με τη δεσμευμένη λέξη struct και στη γενική της μορφή έχει την ακόλουθη σύνταξη:

```
struct [ετικέτα_δομής]
{
     Δηλώσεις πεδίων;
} [λίστα_μεταβλητών];
```

Μια δήλωση `struct` ορίζει έναν τύπο. Αν και η `ετικέτα_δομής` είναι προαιρετική, θα τη χρησιμοποιούμε για να δίνουμε όνομα σε κάθε τύπο δομής. Έτσι θα μπορούμε να δηλώνουμε μεταβλητές. Τα πεδία ή μέλη της δομής χρησιμοποιούνται όπως και οι μεταβλητές του αντίστοιχου τύπου. Μια δομή μπορεί να αποτελείται από πεδία διαφορετικού τύπου. Η λογική συσχέτισή τους είνα ότι περιέχουν την πληροφορία που απαιτείται για την περιγραφή μιας συγκεκριμένης οντότητας.

Όπως θα δούμε στη συνέχεια, μπορούμε προαιρετικά να δηλώσουμε μια `[λίστα_μεταβλητών]` του ίδιου τύπου. Προσοχή, στο τέλος της δήλωσης το `;` είναι υποχρεωτικό. Για παράδειγμα, για να αποθηκεύσουμε πληροφορία για μια εταιρεία, θα μπορούσαμε να δηλώσουμε τον τύπο `company` ως εξής:

```
struct company
{
     char name[50];
     int start_year;
     int field;
     int tax_num;
```

```
     int num_employers;
     char addr[50];
     float balance;
}
```

Πεδία του ίδιου τύπου επιτρέπεται να δηλώνονται στην ίδια ίδια γραμμή, αλλά προτιμούμε να δηλώνονται ξεχωριστά, ώστε να φαίνεται πιο εύκολα η αντιστοίχιση με την πληροφορία που θέλουμε να αποθηκεύσουμε. Τα πεδία μιας δομής αποθηκεύονται σε θέσεις μνήμης που αυξάνονται με το πρώτο να αρχίζει από τη διεύθυνση της δομής.

Η δήλωση της δομής που δεν ακολουθείται από λίστα μεταβλητών δεν προκαλεί δέσμευση μνήμης. Απλά περιγράφει τη μορφή της δομής. Αν η δομή έχει ετικέτα, μπορούμε να τη χρησιμοποιήσουμε για δηλώσεις μεταβλητών:

```
struct company c1, c2;
```

Οι μεταβλητές `c1`, `c2` δηλώνονται σαν δομές τύπου `company`: κάθε μια έχει τα δικά της ξεχωριστά "αντίγραφα" πεδίων. Εναλλακτικά, μπορούμε να δηλώσουμε μεταβλητές στη `[λίστα_μεταβλητών]`. Για παράδειγμα:

```
struct book
{
     char title[100];
     int year;
     float price;
} b1, b2;
```

Οι `b1`, `b2` δηλώνονται σαν δομές τύπου `book`. Αν δηλώσουμε όλες τις μεταβλητές που χρειαζόμαστε στη `[λίστα_μεταβλητών]` μπορούμε να παραλείψουμε την ετικέτα. Ωστόσο, επειδή η πιο συνηθισμένη περίπτωση είναι να χρειαστεί να δηλώσουμε και σε άλλα σημεία ενός μεγάλου προγράμματος τέτοιες μεταβλητές, καλό είναι να επιλέγουμε πάντα μια τέτοια ετικέτα.

Στη C κάθε τύπος δομής ορίζει έναν ξεχωριστό χώρο ονομάτων που έχει τη δική του εμβέλεια. Επομένως, επιτρέπεται να δηλώνουμε μεταβλητές που να έχουν το ίδιο όνομα με τα πεδία μιας δομής. Επίσης, διαφορετικοί τύποι δομών μπορούν να περιέχουν πεδία με το ίδιο όνομα. Για παράδειγμα:

```
struct person
{
     char name[50];
     int tax_num;
     char addr[50];
}
```

Τα τρία αυτά πεδία δεν έχουν καμία σχέση με τα αντίστοιχα πεδία της δομής company. Παρόμοια μπορούμε να δηλώσουμε:

```
int a;
struct S1 {int a[5]; } s1;
struct S2 {int *a; } s2;
```

Και όπως θα δούμε στη συνέχεια, μπορούμε να δηλώσουμε `s1.a[0] = a;` ή `s2.a = &a` ή `s2.a = s1.a` χωρίς κανένα πρόβλημα.

Ο τύπος μιας δομής ολοκληρώνεται με το `}`. Μια δομή μπορεί να περιέχει έναν δείκτη σε ένα στιγμιότυπο του εαυτού της, γιατί επιτρέπεται η δήλωση δεικτών για ημιτελείς τύπους, π.χ.:

```
struct A
{
     struct A *p;
};
```

Η λέξη κλειδί `typedef` χρησιμοποιείται για να ορίσουμε ένα επιπλέον όνομα για κάποιον υπαρκτό τύπο δεδομένων. Για παράδειγμα στην περίπτωση των `struct`:

```
typedef struct Mybook
{
     char title[100];
     int year;
     float price;
} book;
```

Το `Mybook` είναι μια ετικέτα και πρέπει να προσθέτουμε τη λέξη `struct` όταν τη χρησιμοποιούμε. Αντίθετα, επειδή το `typedef` όνομα αποτελεί συνώνυμο τύπου, δεν επιτρέπεται να χρησιμοποιούμε τη λέξη `struct`. Για παράδειγμα:

```
book b1, b2;
```

Πρέπει να σημειωθεί ότι η ετικέτα της δομής μπορεί να έχει το ίδιο όνομα με το συνώνυμο: αντί για `Mybook` θα μπορούσαμε να είχαμε γράψει `book`, αλλά η σημασία τους θα ήταν διαφορετική. Επίσης, θα μπορούσαμε να παραλείψουμε την ετικέτα και να χρησιμοποιούμε το `typedef` όνομα για να δηλώσουμε μεταβλητές:

```
typedef struct
{
     char title[100];
     int year;
     float price;
} book;
```

Ο πιο συνηθισμένος τρόπο για να προσπελάσουμε ένα πεδίο μιας δομής είναι να γράψουμε το όνομα της δομής, να προσθέσουμε τον τελεστή τελεία (`.`) και μετά το όνομα του πεδίου. Δείτε π.χ.

το παρακάτω πρόγραμμα, το οποίο αρχικοποιεί και εμφανίζει τις τιμές των πεδίων της b. Προσπαθήστε να εξηγήσετε την έξοδό του:

```
#include <stdio.h>

struct student
{
     char *name;
     float *avg_grd;
};

int main(void)
{
     float grd = 8.5;
     struct student s1, s2;

     s1.name = "somebody";
     s1.avg_grd = &grd;
     printf("%s %.2f\n", s1.name+3, *s1.avg_grd);

     s2 = s1;
     grd = 3.4;
     printf("%s %.2f\n", s2.name, *s2.avg_grd);

     return 0;
}
```

Μια δομή μπορεί να περιέχει μία ή περισσότερες ένθετες δομές. Μια ένθετη δομή πρέπει να ορίζεται πριν από τον ορισμό της δομής στην οποία περιέχεται. Διαφορετικά, ο μεταγλωττιστής θα εμφανίσει μήνυμα λάθους. Δείτε το παρακάτω παράδειγμα, στο οποίο η δομή `prod` περιέχει τις ένθετες δομές `s_date` και `e_date`:

```
#include <stdio.h>
#include <string.h>

struct date
{
     int day;
     int month;
     int year;
};

struct product
{
     char name[50];
     double price;
```

```
     struct date s_date;
     struct date e_date;
};

int main(void)
{
     struct product prod;

     strcpy(prod.name, "product");

     prod.s_date.day = 1;
     prod.s_date.month = 9;
     prod.s_date.year = 2012;

     prod.e_date.day = 1;
     prod.e_date.month = 9;
     prod.e_date.year = 2015;

     prod.price = 7.5;
     printf("The product's life is %d years\n", prod.e_date.year -
prod.s_date.year);

     return 0;
}

#include <stdio.h>
#include <string.h>

struct date
{
     int day;
     int month;
     int year;
};

struct product
{
     char name[50];
     double price;
     struct date s_date;
     struct date e_date;
};

int main(void)
{
     struct product prod;
```

```
     strcpy(prod.name, "product");

     prod.s_date.day = 1;
     prod.s_date.month = 9;
     prod.s_date.year = 2012;

     prod.e_date.day = 1;
     prod.e_date.month = 9;
     prod.e_date.year = 2015;

     prod.price = 7.5;
     printf("The product's life is %d years\n", prod.e_date.year -
prod.s_date.year);

     return 0;

}
```

Όπως βλέπουμε, για να προσπελάσουμε τα πεδία μιας ένθετης δομής, χρησιμοποιούμε δύο φορές τον τελεστή τελεία (`.`). Γενικά, επιτρέπονται πολλαπλές δηλώσεις δομών μέσα σε δομές, αλλά συνήθως δε γίνεται υπέρβαση των δύο επιπέδων.

Ένας δείκτης σε μια δομή χρησιμοποιείται με τον ίδιο τρόπο όπως και κάθε άλλος δείκτης. Για παράδειγμα, δείτε το παρακάτω πρόγραμμα:

```
#include <stdio.h>
#include <string.h>

struct student
{
     char name[50];
     float grd;
};

int main(void)
{
     struct student *ptr, stud;

     ptr = &stud;
     strcpy((*ptr).name, "somebody");
     (*ptr).grd = 6.7;

     printf("N: %s G: = %.2f\n", stud.name, stud.grd);

     return 0;
```

```
}
```

Η `ptr` δηλώνεται σαν δείκτης σε δομή τύπου `student`. Με την εντολή `ptr = &stud`, ο `ptr` δείχνει στη διεύθυνση μνήμης της δομής `stud`. Συγκεκριμένα, δείχνει στη διεύθυνση μνήμης του πρώτου πεδίου της δομής. Αφού ο `ptr` δείχνει στη διεύθυνση μνήμης της δομής `stud`, το `*ptr` είναι ισοδύναμο με τη δομή `stud` και με τον τελεστή `.` μπορούμε να αποκτήσουμε πρόσβαση στα πεδία της δομής. Η έκφραση `*ptr` πρέπει να περικλείεται σε παρενθέσεις, γιατί ο τελεστής `.` έχει μεγαλύτερη προτεραιότητα από τον τελεστή `*`. Εναλλακτικά μπορούμε να χρησιμοποιούμε τον τελεστή `->`:

```
#include <stdio.h>
#include <string.h>

struct student
{
     char name[50];
     float grd;
};

int main(void)
{
     struct student *ptr, stud;

     ptr = &stud;
     strcpy(ptr->name, "somebody");
     ptr->grd = 6.7;

     printf("N: %s G: = %.2f\n", stud.name, stud.grd);

     return 0;
}
```

Η έκφραση `ptr->name` είναι ισοδύναμη με `(*ptr).name` και η `ptr->grd` με `(*ptr).grd`. Η χρήση του τελεστή `->` θεωρείται απλούστερη.

Μια δομή μπορεί να μεταβιβαστεί σε μια συνάρτηση όπως οποιαδήποτε άλλη μεταβλητή, δηλαδή είτε η ίδια η δομή, είτε η διεύθυνση μνήμης της δομής. Δείτε το παρακάτω πρόγραμμα:

```
#include <stdio.h>
#include <string.h>

struct student
{
     char name[50];
     int code;
```

```
     float grd;
};

void test(struct student stud_1);

int main(void)
{
     struct student stud = {"somebody", 20, 5};

     test(stud);

     printf("N: %s C:%d G:%.2f\n", stud.name, stud.code, stud.grd);

     return 0;
}

void test(struct student stud_1)
{
     strcpy(stud_1.name, "new_name");
     stud_1.code = 30;
     stud_1.grd = 7;
}
```

Όταν καλείται η `test()`, οι τιμές των πεδίων της `stud` αντιγράφονται στα αντίστοιχα πεδία της `stud_1`. Αφού οι μεταβλητές `stud_1` και `stud` αποθηκεύονται σε διαφορετικές θέσεις μνήμης, οποιαδήποτε αλλαγή γίνει στα πεδία της `stud_1` δεν επηρεάζει τις αντίστοιχες τιμές των πεδίων της `stud`. Υπενθυμίζεται ότι ακόμη κι αν χρησιμοποιούσαμε το όνομα `stud` αντί για `stud_1`, το αποτέλεσμα θα ήταν ίδιο, αφού εξακολουθούν να είναι διαφορετικές μεταβλητές.

Αντίθετα, όταν σε μια συνάρτηση μεταβιβάζεται η διεύθυνση μνήμης μιας μεταβλητής, η συνάρτηση μπορεί να αλλάξει την τιμή της. Για παράδειγμα, δείτε μια τροποποιημένη `test()`:

```
#include <stdio.h>
#include <string.h>

struct student
{
     char name[50];
     int code;
     float grd;
};

void test(struct student *ptr);

int main(void)
{
```

```
    struct student stud = {"somebody", 20, 5};

    test(&stud);

    printf("N: %s C:%d G:%.2f\n", stud.name, stud.code, stud.grd);

    return 0;
}

void test(struct student *ptr)
{
    strcpy(ptr->name, "new_name");
    ptr->code = 30;
    ptr->grd = 7;
}
```

Όταν καλείται η `test()`, έχουμε `ptr=&stud`. Άρα, αφού ο `ptr` δείχνει στη διεύθυνση μνήμης της `stud`, η συνάρτηση μπορεί να αλλάξει τις τιμές των πεδίων της.

3. Η Στοίβα

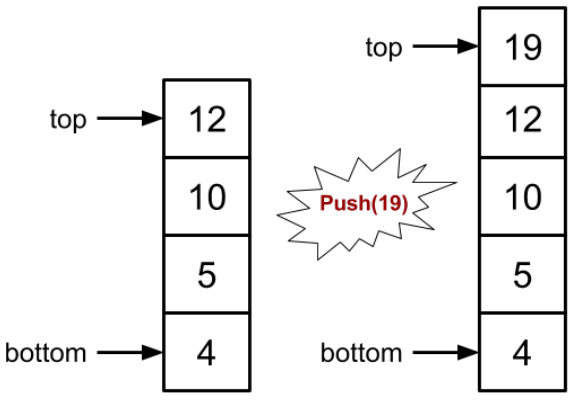
Η στοίβα αποτελεί μια απλή γραμμική δομή δεδομένων που λειτουργεί σαν μια "συλλογή" από στοιχεία. Διαθέτει δύο βασικές λειτουργίες, `push` και `pop`.

- Η `push` προσθέτει ένα στοιχείο στη συλλογή
- Η `pop` αφαιρεί από τη συλλογή το τελευταίο στοιχείο που προστέθηκε

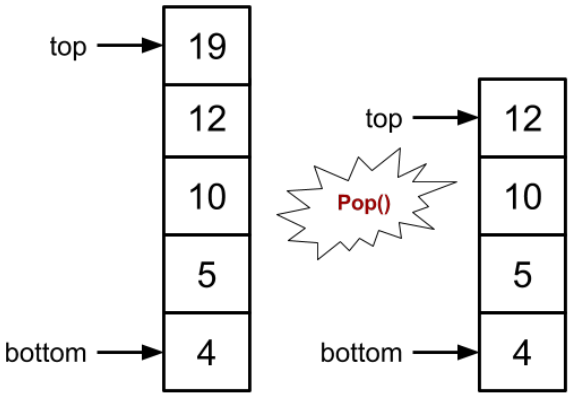
Η στοίβα ακολουθεί τη σειρά LIFO (Last In First Out): Το τελευταίο στοιχείο που προστέθηκε είναι αυτό που θα βγει πρώτο. Συνήθως σε μια στοίβα χρησιμοποιούμε δύο "δείκτες", τον `top` που δείχνει το τελευταίο στοιχείο που προστέθηκε στη στοίβα (το "πιο πάνω") και τον `bottom` που δείχνει το πρώτο στοιχείο που προστέθηκε στη στοίβα (το "πιο κάτω"). Ωστόσο, ο bottom μπορεί να παραλείπεται.

Δείτε τα παρακάτω παραδείγματα για τις `push`, `pop`:

- Αρχικά, μια συγκεκριμένη στιγμή, εισάγεται στη στοίβα ο αριθμός `19`. Προσέξτε ότι αλλάζει ο δείκτης `top`.



- Έπειτα, αφαιρείται το τελευταίο στοιχείο που προστέθηκε στη στοίβα. Προσέξτε ότι αλλάζει ο δείκτης `top`.



Οι συνήθεις λειτουργίες που υλοποιούνται σε μια στοίβα είναι οι εξής:

- `Push(x)`, εισαγωγή (ώθηση) στοιχείου με τιμή `x` στην κορυφή της στοίβας
- `Pop()`, εξαγωγή (απώθηση) στοιχείου από την κορυφή της στοίβας
- `Top()`, επιστροφή στοιχείου (χωρίς διαγραφή) από την κορυφή της στοίβας

Συχνά, υλοποιούνται και οι ακόλουθες βοηθητικές λειτουργίες:

- `isEmpty()`: ελέγχει αν η στοίβα είναι άδεια (πρόληψη υποχείλισης - ο πίνακας μπορεί να είναι άδειος και να επιχειρηθεί άντληση από τη "θέση" `-1`)
- `isFull()`: ελέγχει αν η στοίβα είναι γεμάτη (πρόληψη υπερχείλισης - ο πίνακας μπορεί να γεμίσει και να επιχειρηθεί προσθήκη σε θέση εκτός του)
- `size()`: επιστρέφει το μέγεθος της στοίβας

4. Υλοποίηση της Στοίβας με πίνακες

Να υλοποιηθεί μια στοίβα με χρήση πινάκων, η οποία να υποστηρίζει τις παραπάνω λειτουργίες. Η εισαγωγή στοιχείων να γίνεται από αριστερά προς δεξιά, το κορυφαίο στοιχείο να βρίσκεται στη θέση `top` και η στοίβα να περιέχει το πολύ `SIZE` στοιχεία. Να θεωρήσετε ότι στη στοίβα θα αποθηκεύονται θετικοί ακέραιοι αριθμοί. Στη `main` να συμπεριληφθούν παραδείγματα όλων των λειτουργιών.